PAC22140

Battery Management System

Multi-Mode Power Manager™

Configurable Analog Front End™

Application Specific Power Drivers™

Arm® Cortex®-M0 Controller Core

TABLE OF CONTENTS

LIST OF TABLES

LIST OF REGISTERS

LIST OF FIGURES

1. STYLES AND FORMATTING CONVENTIONS

1.1. Overview

This chapter describes formatting and styles used through the document.

1.2. Number Representation

Numbers in a base other than decimal have a prefix or postfix as indicator. All numbers use little endian formatting, most significant bit/digit is to the left. Digits for binary and hexadecimal representation are grouped with a single space every four digits to improve readability. Binary numbers use “b” as postfix, hexadecimal numbers use “0x” as prefix.

For example 1011b binary = 0xB hexadecimal = 11 decimal.

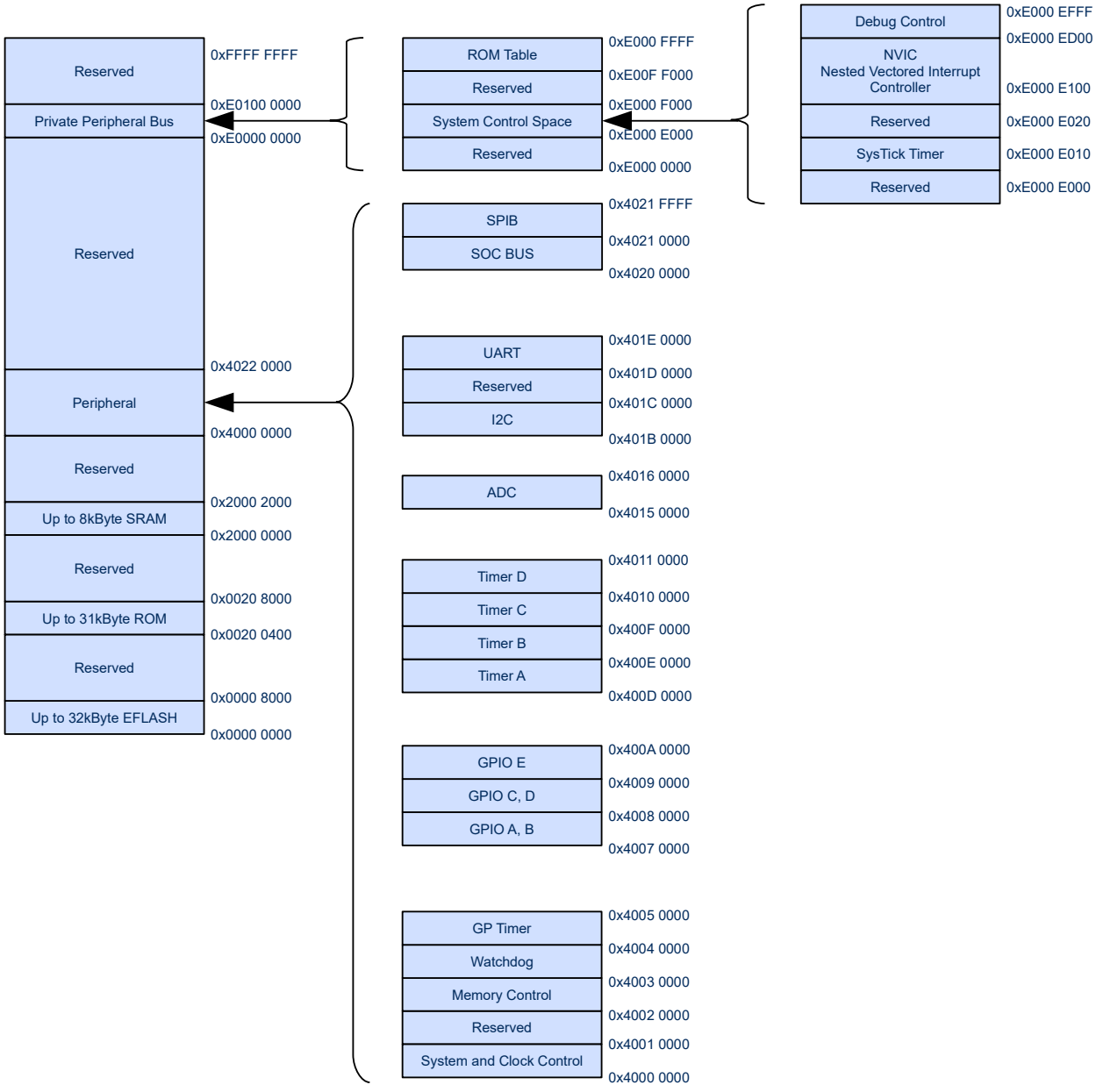
1.3. Formatting Styles

TYPE	EXAMPLE	DESCRIPTION
Register Name	**RTCCTL**	Register names use capital letter and bold formatting.
Register Bit(s)	**RTCCTL.RTCCLKDIV**	Register bits are always represented with the register name separated with a period
Function selected by Register bit(s)	**[RTCCTL.RTCCLKDIV]**	Within text blocks, functions selected with a register bit setting are set in brackets. For example **[RTCCTL.RTCCLKDIV]** means divider settings /2 to / 65536.
Pin Function	XIN	Pin functions use capital letters
Formulas	`CLK = FCLK / DIV`	Formulas use Teletype font.
Links	Number Representation	Clickable Links are underlined and blue
CPU Mnemonic	`MRS`	CPU Mnemonic use Teletype font.
Operands	`{`*Rd*`,}` *Rn*, *Rm*	Operands use Italic
Code examples	`B loopA`	Code examples use Teletype font.

2. MEMORY AND REGISTER MAP

2.1. Memory Map

Figure 2-1. Memory Map



2.2. Register Map

Table 2-1. Embedded FLASH Register Map

ADDRESS	NAME	DESCRIPTION
Embedded FLASH		
0x0000 0000 – 0x0000 03FF	**EFLASHP0**	EFLASH page 0
0x0000 0400 – 0x0000 07FF	**EFLASHP1**	EFLASH page 1
0x0000 0800 – 0x0000 0BFF	**EFLASHP2**	EFLASH page 2
0x0000 0C00 – 0x0000 0FFF	**EFLASHP3**	EFLASH page 3
0x0000 1000 – 0x0000 13FF	**EFLASHP4**	EFLASH page 4
0x0000 1400 – 0x0000 17FF	**EFLASHP5**	EFLASH page 5
0x0000 1800 – 0x0000 1BFF	**EFLASHP6**	EFLASH page 6
0x0000 1C00 – 0x0000 1FFF	**EFLASHP7**	EFLASH page 7
0x0000 2000 – 0x0000 23FF	**EFLASHP8**	EFLASH page 8
0x0000 2400 – 0x0000 27FF	**EFLASHP9**	EFLASH page 9
0x0000 2800 – 0x0000 2BFF	**EFLASHP10**	EFLASH page 10
0x0000 2C00 – 0x0000 2FFF	**EFLASHP11**	EFLASH page 11
0x0000 3000 – 0x0000 33FF	**EFLASHP12**	EFLASH page 12
0x0000 3400 – 0x0000 37FF	**EFLASHP13**	EFLASH page 13
0x0000 3800 – 0x0000 3BFF	**EFLASHP14**	EFLASH page 14
0x0000 3C00 – 0x0000 3FFF	**EFLASHP15**	EFLASH page 15
0x0000 4000 – 0x0000 43FF	**EFLASHP16**	EFLASH page 16
0x0000 4400 – 0x0000 47FF	**EFLASHP17**	EFLASH page 17
0x0000 4800 – 0x0000 4BFF	**EFLASHP18**	EFLASH page 18
0x0000 4C00 – 0x0000 4FFF	**EFLASHP19**	EFLASH page 19
0x0000 5000 – 0x0000 53FF	**EFLASHP20**	EFLASH page 20
0x0000 5400 – 0x0000 57FF	**EFLASHP21**	EFLASH page 21
0x0000 5800 – 0x0000 5BFF	**EFLASHP22**	EFLASH page 22
0x0000 5C00 – 0x0000 5FFF	**EFLASHP23**	EFLASH page 23
0x0000 6000 – 0x0000 63FF	**EFLASHP24**	EFLASH page 24
0x0000 6400 – 0x0000 67FF	**EFLASHP25**	EFLASH page 25
0x0000 6800 – 0x0000 6BFF	**EFLASHP26**	EFLASH page 26
0x0000 6C00 – 0x0000 6FFF	**EFLASHP27**	EFLASH page 27
0x0000 7000 – 0x0000 73FF	**EFLASHP28**	EFLASH page 28
0x0000 7400 – 0x0000 77FF	**EFLASHP29**	EFLASH page 29
0x0000 7800 – 0x0000 7BFF	**EFLASHP30**	EFLASH page 30
0x0000 7C00 – 0x0000 7FFF	**EFLASHP31**	EFLASH page 31

Table 2-2. ROM Register Map

ADDRESS	NAME	DESCRIPTION
INFO ROM		
0x0010 0000 – 0x0010 000F	**Reserved**	Reserved
0x0010 0010	**ROSC11**	ROSC frequency in Hz at 11b setting (8 MHz)
0x0010 0014	**Reserved**	Reserved
0x0010 0018	**Reserved**	Reserved
0x0010 001C	**Reserved**	Reserved
0x0010 0020	**ADCGAIN**	ADC gain * 65536 in ADC counts/V.
0x0010 0024	**ADCOFF**	ADC offset (Two's complement) * 65536 in ADC counts.
0x0010 0028	**FTTEMP**	Test temperature for internal temp sensor in °C
0x0010 002A	**TEMPS**	Internal temp sensor reading at FTTEMP temperature in ADC counts
0x0010 002C	**CLKREF**	4MHz CLKREF frequency in Hz
0x0010 0030	**Reserved**	Reserved
0x0010 0034	**Reserved**	Reserved
0x0010 0038	**Reserved**	Reserved
0x0010 003C	**Reserved**	Reserved
0x0010 0040	**Reserved**	Reserved
0x0010 0044	**PACIDR**	Device part number and revision
0x0010 0048 – 0x0010 00FF	**Reserved**	Reserved
DATA ROM		
0x0020 0400 – 0x0020 7FFF	**ROM**	ROM Area

Table 2-3. System Clock Control Register Map

ADDRESS	NAME	DESCRIPTION
System Clock Control		
0x4000 0000	**SCCTL**	System clock control
0x4000 0004	**PLLCTL**	PLL control
0x4000 0008	**ROSCCTL**	Ring oscillator control
0x4000 000C	**XTALCTL**	Crystal driver control

Table 2-4. FLASH Memory Controller Register Map

ADDRESS	NAME	DESCRIPTION
FLASH Memory Controller		
0x4002 0000	**FLASHLOCK**	FLASH lock
0x4002 0004	**FLASHCTL**	FLASH Control
0x4002 0008	**FLASHPAGE**	FLASH page selection
0x4002 000C	**Reserved**	Reserved
0x4002 0010	**Reserved**	Reserved
0x4002 0014	**FLASHPERASE**	FLASH page erase
0x4002 0018	**Reserved**	Reserved
0x4002 001C	**Reserved**	Reserved
0x4002 0020	**Reserved**	Reserved
0x4002 0024	**SWDACCESS**	SWD access control
0x4002 0028	**FLASHWSTATE**	FLASH wait state control
0x4002 002C	**FLASHBWRITE**	FLASH buffered write enable
0x4002 0030	**FLASHBWDATA**	FLASH buffered write data and address

Table 2-5. Watchdog Timer Register Map

ADDRESS	NAME	DESCRIPTION
Watchdog Timer		
0x4003 0000	**WDTCTL**	Watchdog timer control
0x4003 0004	**WDTCDV**	Watchdog timer count-down value
0x4003 0008	**WDTCTR**	Watchdog timer counter

Table 2-6. General Purpose Timer Register Map

ADDRESS	NAME	DESCRIPTION
Real Time Clock		
0x4004 0000	**RTCCTL**	General purpose timer control
0x4004 0004	**RTCCDV**	General purpose timer count-down value
0x4004 0008	**RTCCTR**	General purpose timer counter

Table 2-7. GPIO Port A Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port A		
0x4007 0000	**GPIOAOUT**	GPIO Port A output
0x4007 0004	**GPIOAOUTEN**	GPIO Port A output enable
0x4007 0008	**GPIOADS**	GPIO Port A output drive strength
0x4007 000C	**GPIOAPU**	GPIO Port A output weak pull up
0x4007 0010	**GPIOAPD**	GPIO Port A output weak pull down
0x4007 0014	**GPIOAIN**	GPIO Port A input
0x4007 0018	**Reserved**	Reserved
0x4007 001C	**GPIOAPSEL**	GPIO Port A peripheral select
0x4007 0020	**GPIOAINTP**	GPIO Port A interrupt polarity select
0x4007 0024	**GPIOAINTE**	GPIO Port A interrupt enable select
0x4007 0028	**GPIOAINTF**	GPIO Port A interrupt flag
0x4007 002C	**GPIOAINTM**	GPIO Port A interrupt mask

Table 2-8. GPIO Port B Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port B		
0x4007 0040	**GPIOBOUT**	GPIO Port B output
0x4007 0044	**GPIOBOUTEN**	GPIO Port B output enable
0x4007 0048	**GPIOBODS**	GPIO Port B output drive strength
0x4007 004C	**GPIOBPU**	GPIO Port B output weak pull up
0x4007 0050	**GPIOBPD**	GPIO Port B output weak pull down
0x4007 0054	**GPIOBIN**	GPIO Port B input
0x4007 0058	**Reserved**	Reserved
0x4007 005C	**GPIOBPSEL**	GPIO Port B peripheral select
0x4007 0060	**GPIOBINTP**	GPIO Port B interrupt polarity select
0x4007 0064	**GPIOBINTE**	GPIO Port B interrupt enable select
0x4007 0068	**GPIOBINTF**	GPIO Port B interrupt flag
0x4007 006C	**GPIOBINTM**	GPIO Port B interrupt mask

Table 2-9. GPIO Port AB Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port AB		
0x4007 0080	**GPIOABOUT**	GPIO Port AB output
0x4007 0084	**GPIOABOUTEN**	GPIO Port AB output enable
0x4007 0088	**GPIOABODS**	GPIO Port AB output drive strength
0x4007 008C	**GPIOABPU**	GPIO Port AB output weak pull up
0x4007 0090	**GPIOABPD**	GPIO Port AB output weak pull down
0x4007 0094	**GPIOABIN**	GPIO Port AB input
0x4007 0098	**Reserved**	Reserved
0x4007 009C	**GPIOABPSEL**	GPIO Port AB peripheral select
0x4007 00A0	**GPIOABINTP**	GPIO Port AB interrupt polarity select
0x4007 00A4	**GPIOABINTE**	GPIO Port AB interrupt enable select
0x4007 00A8	**GPIOABINTF**	GPIO Port AB interrupt flag
0x4007 00AC	**GPIOABINTM**	GPIO Port AB interrupt mask

Table 2-10. GPIO Port C Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port C		
0x4008 0000	**GPIOCOUT**	GPIO Port C output
0x4008 0004	**GPIOCOUTEN**	GPIO Port C output enable
0x4008 0008	**Reserved**	Reserved
0x4008 000C	**Reserved**	Reserved
0x4008 0010	**Reserved**	Reserved
0x4008 0014	**GPIOCIN**	GPIO Port C input
0x4008 0018	**GPIOCINE**	GPIO Port C input enable
0x4008 001C	**Reserved**	Reserved
0x4008 0020	**GPIOCINTP**	GPIO Port C interrupt polarity select
0x4008 0024	**GPIOCINTE**	GPIO Port C interrupt enable select
0x4008 0028	**GPIOCINTF**	GPIO Port C interrupt flag
0x4008 002C	**GPIOCINTM**	GPIO Port C interrupt mask

Table 2-11. GPIO Port D Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port D		
0x4008 0040	**GPIODOUT**	GPIO Port D output
0x4008 0044	**GPIODOUTEN**	GPIO Port D output enable
0x4008 0048	**GPIODODS**	GPIO Port D output drive strength
0x4008 004C	**GPIODPU**	GPIO Port D output weak pull up
0x4008 0050	**GPIODPD**	GPIO Port D output weak pull down

ADDRESS	NAME	DESCRIPTION
0x4008 0054	**GPIODIN**	GPIO Port D input
0x4008 0058	**Reserved**	Reserved
0x4008 005C	**GPIODPSEL**	GPIO Port D peripheral select
0x4008 0060	**GPIODINTP**	GPIO Port D interrupt polarity select
0x4008 0064	**GPIODINTE**	GPIO Port D interrupt enable select
0x4008 0068	**GPIODINTF**	GPIO Port D interrupt flag
0x4008 006C	**GPIODINTM**	GPIO Port D interrupt mask

Table 2-12. GPIO Port CD Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port CD		
0x4008 0080	**GPIOCDOUT**	GPIO Port CD output
0x4008 0084	**GPIOCDOUTEN**	GPIO Port CD output enable
0x4008 0088	**Reserved**	Reserved
0x4008 008C	**Reserved**	Reserved
0x4008 0090	**Reserved**	Reserved
0x4008 0094	**GPIOCDIN**	GPIO Port CD input
0x4008 0098	**Reserved**	Reserved
0x4008 009C	**GPIOCDPSEL**	GPIO Port CD peripheral select
0x4008 00A0	**GPIOCDINTP**	GPIO Port CD interrupt polarity select
0x4008 00A4	**GPIOCDINTE**	GPIO Port CD interrupt enable select
0x4008 00A8	**GPIOCDINTF**	GPIO Port CD interrupt flag
0x4008 00AC	**GPIOCDINTM**	GPIO Port CD interrupt mask

Table 2-13. GPIO Port E Register Map

ADDRESS	NAME	DESCRIPTION
GPIO Port E		
0x4009 0000	**GPIOEOUT**	GPIO Port E output
0x4009 0004	**GPIOEOUTEN**	GPIO Port E output enable
0x4009 0008	**GPIOEODS**	GPIO Port E output drive strength
0x4009 000C	**GPIOEPU**	GPIO Port E output weak pull up
0x4009 0010	**GPIOEPD**	GPIO Port E output weak pull down
0x4009 0014	**GPIOEIN**	GPIO Port E input
0x4009 0018	**Reserved**	Reserved
0x4009 001C	**GPIOEPSEL**	GPIO Port E peripheral select
0x4009 0020	**GPIOEINTP**	GPIO Port E interrupt polarity select
0x4009 0024	**GPIOEINTE**	GPIO Port E interrupt enable select
0x4009 0028	**GPIOEINTF**	GPIO Port E interrupt flag
0x4009 002C	**GPIOEINTM**	GPIO Port E interrupt mask

Table 2-14. Timer A Register Map

ADDRESS	NAME	DESCRIPTION
Timer A		
0x400D 0000	**TACTL**	Timer A control
0x400D 0004	**TAPRD**	Timer A period
0x400D 0008	**TACTR**	Timer A counter
Timer A PWMA Capture and Compare		
0x400D 0040	**TACCTRL0**	Timer A capture and compare 0 control
0x400D 0044	**TACTR0**	Timer A counter 0
0x400D 0048	**TACCTRL1**	Timer A capture and compare 1 control
0x400D 004C	**TACTR1**	Timer A counter 1
0x400D 0050	**TACCTRL2**	Timer A capture and compare 2 control
0x400D 0054	**TACTR2**	Timer A counter 2
0x400D 0058	**TACCTRL3**	Timer A capture and compare 3 control
0x400D 005C	**TACTR3**	Timer A counter 3
0x400D 0060	**TACCTRL4**	Timer A capture and compare 4 control
0x400D 0064	**TACTR4**	Timer A counter 4
0x400D 0068	**TACCTRL5**	Timer A capture and compare 5 control
0x400D 006C	**TACTR5**	Timer A counter 5
0x400D 0070	**TACCTRL6**	Timer A capture and compare 6 control
0x400D 0074	**TACTR6**	Timer A counter 6
0x400D 0078	**TACCTRL7**	Timer A capture and compare 7 control
0x400D 007C	**TACTR7**	Timer A counter 7
Timer A Dead Time Generator		
0x400D 00A0	**DTGA0CTL**	Timer A dead time generator 0 control
0x400D 00A4	**DTGA0LED**	Timer A dead time generator 0 leading edge delay
0x400D 00A8	**DTGA0TED**	Timer A dead time generator 0 trailing edge delay
0x400D 00B0	**DTGA1CTL**	Timer A dead time generator 1 control
0x400D 00B4	**DTGA1LED**	Timer A dead time generator 1 leading edge delay
0x400D 00B8	**DTGA1TED**	Timer A dead time generator 1 trailing edge delay
0x400D 00C0	**DTGA2CTL**	Timer A dead time generator 2 control
0x400D 00C4	**DTGA2LED**	Timer A dead time generator 2 leading edge delay
0x400D 00C8	**DTGA2TED**	Timer A dead time generator 2 trailing edge delay
0x400D 00D0	**DTGA3CTL**	Timer A dead time generator 3 control
0x400D 00D4	**DTGA3LED**	Timer A dead time generator 3 leading edge delay
0x400D 00D8	**DTGA3TED**	Timer A dead time generator 3 trailing edge delay

Table 2-15. Timer B Register Map

ADDRESS	NAME	DESCRIPTION
Timer B		
0x400E 0000	**TBCTL**	Timer B control
0x400E 0004	**TBPRD**	Timer B period
0x400E 0008	**TBCTR**	Timer B counter
Timer B PWMB Capture and Compare		
0x400E 0040	**TBCCTRL0**	Timer B capture and compare 0 control
0x400E 0044	**TBCTR0**	Timer B counter 0
0x400E 0048	**TBCCTRL1**	Timer B capture and compare 1 control
0x400E 004C	**TBCTR1**	Timer B counter 1
0x400E 0050	**TBCCTRL2**	Timer B capture and compare 2 control
0x400E 0054	**TBCTR2**	Timer B counter 2
0x400E 0058	**TBCCTRL3**	Timer B capture and compare 3 control
0x400E 005C	**TBCTR3**	Timer B counter 3
Timer B Dead Time Generator		
0x400E 00A0	**DTGB0CTL**	Timer B dead time generator 0 control
0x400E 00A4	**DTGB0LED**	Timer B dead time generator 0 leading edge delay
0x400E 00A8	**DTGB0TED**	Timer B dead time generator 0 trailing edge delay

Table 2-16. Timer C Register Map

ADDRESS	NAME	DESCRIPTION
Timer C		
0x400F 0000	**TCCTL**	Timer C control
0x400F 0004	**TCPRD**	Timer C period
0x400F 0008	**TCCTR**	Timer C counter
Timer C PWMC Capture and Compare		
0x400F 0040	**TCCCTRL0**	Timer C capture and compare 0 control
0x400F 0044	**TCCTR0**	Timer C counter 0
0x400F 0048	**TCCCTRL1**	Timer C capture and compare 1 control
0x400F 004C	**TCCTR1**	Timer C counter 1
Timer C Dead Time Generator		
0x400F 00A0	**DTGC0CTL**	Timer C dead time generator 0 control
0x400F 00A4	**DTGC0LED**	Timer C dead time generator 0 leading edge delay
0x400F 00A8	**DTGC0TED**	Timer C dead time generator 0 trailing edge delay

Table 2-17. Timer D Register Map

ADDRESS	NAME	DESCRIPTION
Timer D		
0x4010 0000	**TDCTL**	Timer D control
0x4010 0004	**TDPRD**	Timer D period
0x4010 0008	**TDCTR**	Timer D counter
Timer D PWMD Capture and Compare		
0x4010 0040	**TDCCTL0**	Timer D capture and compare 0 control
0x4010 0044	**TDCTR0**	Timer D counter 0
0x4010 0048	**TDCCTRL1**	Timer D capture and compare 1 control
0x4010 004C	**TDCTR1**	Timer D counter 1
Timer D Dead Time Generator		
0x4010 00A0	**DTGD0CTL**	Timer D dead time generator 0 control
0x4010 00A4	**DTGD0LED**	Timer D dead time generator 0 leading edge delay
0x4010 00A8	**DTGD0TED**	Timer D dead time generator 0 trailing edge delay

Table 2-18. EMUX Register Map

ADDRESS	NAME	DESCRIPTION
EMUX		
0x4015 0000	**EMUXCTL**	ADC external MUX control
0x4015 0004	**EMUXDATA**	ADC external MUX data

Table 2-19. ADC Register Map

ADDRESS	NAME	DESCRIPTION
ADC		
0x4015 0008	**ADCCTL**	ADC control
0x4015 000C	**ADCR**	ADC conversion result
0x4015 0010	**ADCINT**	ADC interrupt

Table 2-20. ADC Auto-Sampling Sequencer 0 Register Map

ADDRESS	NAME	DESCRIPTION
ADC Auto-Sampling Sequencer 0		
0x4015 0040	**ASCTL0**	Auto-sampling sequencer 0 control
0x4015 0044	**AS0S0**	Auto-sampling sequencer 0 sample 0 control
0x4015 0048	**AS0R0**	Auto-sampling sequencer 0 sample 0 result
0x4015 004C	**AS0S1**	Auto-sampling sequencer 0 sample 1 control
0x4015 0050	**AS0R1**	Auto-sampling sequencer 0 sample 1 result
0x4015 0054	**AS0S2**	Auto-sampling sequencer 0 sample 2 control
0x4015 0058	**AS0R2**	Auto-sampling sequencer 0 sample 2 result
0x4015 005C	**AS0S3**	Auto-sampling sequencer 0 sample 3 control

ADDRESS	NAME	DESCRIPTION
0x4015 0060	**AS0R3**	Auto-sampling sequencer 0 sample 3 result
0x4015 0064	**AS0S4**	Auto-sampling sequencer 0 sample 4 control
0x4015 0068	**AS0R4**	Auto-sampling sequencer 0 sample 4 result
0x4015 006C	**AS0S5**	Auto-sampling sequencer 0 sample 5 control
0x4015 0070	**AS0R5**	Auto-sampling sequencer 0 sample 5 result
0x4015 0074	**AS0S6**	Auto-sampling sequencer 0 sample 6 control
0x4015 0078	**AS0R6**	Auto-sampling sequencer 0 sample 6 result
0x4015 007C	**AS0S7**	Auto-sampling sequencer 0 sample 7 control
0x4015 0080	**AS0R7**	Auto-sampling sequencer 0 sample 7 result

Table 2-21. ADC Auto-Sampling Sequencer 1 Register Map

ADDRESS	NAME	DESCRIPTION
ADC Auto-Sampling Sequencer 1		
0x4015 0100	**ASCTL1**	Auto-sampling sequencer 1 control
0x4015 0104	**AS1S0**	Auto-sampling sequencer 1 sample 0 control
0x4015 0108	**AS1R0**	Auto-sampling sequencer 1 sample 0 result
0x4015 010C	**AS1S1**	Auto-sampling sequencer 1 sample 1 control
0x4015 0110	**AS1R1**	Auto-sampling sequencer 1 sample 1 result
0x4015 0114	**AS1S2**	Auto-sampling sequencer 1 sample 2 control
0x4015 0118	**AS1R2**	Auto-sampling sequencer 1 sample 2 result
0x4015 011C	**AS1S3**	Auto-sampling sequencer 1 sample 3 control
0x4015 0120	**AS1R3**	Auto-sampling sequencer 1 sample 3 result
0x4015 0124	**AS1S4**	Auto-sampling sequencer 1 sample 4 control
0x4015 0128	**AS1R4**	Auto-sampling sequencer 1 sample 4 result
0x4015 012C	**AS1S5**	Auto-sampling sequencer 1 sample 5 control
0x4015 0130	**AS1R5**	Auto-sampling sequencer 1 sample 5 result
0x4015 0134	**AS1S6**	Auto-sampling sequencer 1 sample 6 control
0x4015 0138	**AS1R6**	Auto-sampling sequencer 1 sample 6 result
0x4015 013C	**AS1S7**	Auto-sampling sequencer 1 sample 7 control
0x4015 0140	**AS1R7**	Auto-sampling sequencer 1 sample 7 result

Table 2-22. I²C Register Map

ADDRESS	NAME	DESCRIPTION
I²C		
0x40B0 0000	**I2CCFG**	I²C configuration
0x40B0 0004	**I2CSTATUS**	I²C interrupt and status
0x40B0 0008	**I2CIE**	I²C interrupt enable
0x40B0 0030	**I2CMCTRL**	I²C master access control
0x40B0 0034	**I2CMRXDATA**	I²C master receive data

ADDRESS	NAME	DESCRIPTION
0x40B0 0038	**I2CMTXDATA**	I^2C master transmit data
0x40B0 0040	**I2CBAUD**	I^2C master baud rate
0x40B0 0070	**I2CSRXDATA**	I^2C slave receive data
0x40B0 0074	**I2CSTXDATA**	I^2C slave transmit data
0x40B0 0078	**I2CSADDR**	I^2C slave address

Table 2-23. UART Register Map

ADDRESS	NAME	DESCRIPTION
UART		
0x401D 0000	**UARTRXTX**	UART receive/transmit FIFO (available only if **UARTLCR.DLAB** = 0b)
	UARTDL_L	UART divisor latch low (available only if **UARTLCR.DLAB** = 1b)
0x401D 0004	**UARTIER**	UART interrupt enable (available only if **UARTLCR.DLAB** = 0b)
	UARTDL_H	UART divisor latch high (available only if **UARTLCB.DLAB** = 1b)
0x401D 0008	**UARTIIR**	UART interrupt identification (only for register read)
	UARTFCTL	UART FIFO control (only for register write)
0x401D 000C	**UARTLCR**	UART line control
0x401D 0010	**UARTMCR**	UART modem control
0x401D 0014	**UARTLSR**	UART line status
0x401D 0018	**UARTMSR**	UART modem status
0x401D 001C	**UARTSP**	UART Scratch Pad
0x401D 0020	**UARTFCTL2**	UART FIFO control
0x401D 0024	**UARTIER2**	UART interrupt enable
0x401D 0028	**UARTDL_L2**	UART divisor latch low byte
0x401D 002C	**UARTDL_H2**	UART divisor latch high byte
0x401D 0038	**UARTFD_F**	UART fractional divisor value
0x401D 003C	**Reserved**	Reserved
0x401D 0040	**UARTSTAT**	UART FIFO status

Table 2-24. SOC Bus Bridge Register Map

ADDRESS	NAME	DESCRIPTION
SOC Bus Bridge		
0x4020 0000	**SOCBCTL**	SOC Bus Bridge control
0x4020 0004	**SOCBCFG**	SOC Bus Bridge configuration
0x4020 0008	**SOCBCLKDIV**	SOC Bus Bridge clock divider
0x4000 000C	**Reserved**	Reserved
0x4000 0010	**Reserved**	Reserved
0x4020 0014	**SOCBSTAT**	SOC Bus Bridge status
0x4020 0018	**SOCBSSTR**	SOC Bus Bridge Chip Select steering
0x4020 001C	**SOCBD**	SOC Bus Bridge data

ADDRESS	NAME	DESCRIPTION
0x4020 0020	**SOCBINT_EN**	SOC Bus Bridge interrupt enable

Table 2-25. SPI Register Map

ADDRESS	NAME	DESCRIPTION
SPI		
0x4021 0000	**SPICTL**	SPI control
0x4021 0004	**SPICFG**	SPI configuration
0x4021 0008	**SPICLKDIV**	SPI clock divider
0x4021 000C	**Reserved**	Reserved
0x4021 0010	**Reserved**	Reserved
0x4021 0014	**SPISTAT**	SPI status
0x4021 0018	**SPICSSTR**	SPI chip select steering
0x4021 001C	**SPID**	SPI data
0x4021 0020	**SPIINT_EN**	SPI interrupt enable

3. INFORMATION BLOCK

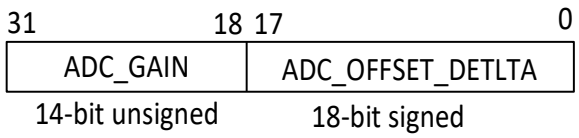
3.1. Register

3.1.1. Register Map

Table 3-1. Information Block Register Map

ADDRESS	BYTE OFFSET							
	0		4		8		12	
0x0010 0000	Reserved		Reserved		Reserved		CLKOUT	Reserved
0x0010 0010	ROSC11		Reserved		Reserved		Reserved	
0x0010 0020	ADCGAIN		ADCOFF		FTTEMP	TEMPS	CLKREF	
0x0010 0030	Reserved				Reserved	SCLK	Reserved	
0x0010 0040	Reserved		PACIDR		Reserved			
0x0010 0050	Reserved							
0x0010 0060	UNIQUEID						Reserved	
0x0010 0070	VB01ADCCAL		VB02ADCCAL		VB03ADCCAL		VB04ADCCAL	
0x0010 0080	VB05ADCCAL		VB06ADCCAL		VB07ADCCAL		VB08ADCCAL	
0x0010 0090	VB09ADCCAL		VB10ADCCAL		VB11ADCCAL		VB12ADCCAL	
0x0010 00A0	VB13ADCCAL		VB14ADCCAL		VB15ADCCAL		VB16ADCCAL	
0x0010 00B0	VB17ADCCAL		VB18ADCCAL		VB19ADCCAL		VB20ADCCAL	
0x0010 00C0	IADC_GAIN		IADC_OFFSET		Reserved		Reserved	
0x0010 00D0- 0x0010 00FF	Reserved							

Each of the 32-bit VBxxADCCAL values consist of a 14-bit ADC_GAIN and 18-bit ADC_OFFSET_DELTA as illustrated below:



Where:

ADC_GAIN = ADC Gain * 2^{28}

ADC_OFFSET_DELTA = (-6.25V – ADC Offset) * 2^{15}

And

ADC Gain is in V/count

ADC Offset is in V

These values can be used in the VADC calibration equation:

VADC Cal Result (in V) = VADC Result (counts) * ADC Gain (V/count) + ADC Offset (V)

For the IADC calibration factors, we have the following:

IADC_GAIN = IADC Gain * 2^{28}

IADC_OFFSET = IADC Offset * 2^{28}

3.1.2. CLKOUT

Register 3-1. CLKOUT (CLKOUT Frequency Value, 0x0010 000C)

BIT	NAME	ACCESS	DESCRIPTION
16:0	**CLKOUT**	R	CLKOUT frequency in Hz for a 250Hz CLKOUT configuration.

3.1.3. ROSC11

Register 3-2. ROSC11 (ROSC11 Frequency Value, 0x0010 0010)

BIT	NAME	ACCESS	DESCRIPTION
31:0	**ROSC11**	R	ROSC frequency in Hz at setting 11b (8MHz).

3.1.4. ADCGAIN

Register 3-3. ADCGAIN (ADC Gain Value, 0x0010 0020)

BIT	NAME	ACCESS	DESCRIPTION
31:0	**ADCGAIN**	R	ADC gain (in ADC counts/volt) * 65536.

3.1.5. ADCOFF

Register 3-4. ADCOFF (ADC Offset, 0x0010 0024)

BIT	NAME	ACCESS	DESCRIPTION
31:0	**ADCOFF**	R	Two's complement of the ADC offset * 65536. The value of this field is in ADC counts.

3.1.6. FTTEMP

Register 3-5. FTTEMP (FT Temp value, 0x0010 0028)

BIT	NAME	ACCESS	DESCRIPTION
15:0	**FTTEMP**	R	Test temperature for internal temp sensor in °C[1]

3.1.7. TEMPS

Register 3-6. TEMPS (Temperature Sensor reading, 0x0010 002A)

BIT	NAME	ACCESS	DESCRIPTION
15:0	**TEMPS**	R	Internal temp sensor ADC reading at FTTEMP[2]

3.1.8. CLKREF

Register 3-7. CLKREF (CLKREF Frequency Value, 0x0010 002C)

BIT	NAME	ACCESS	DESCRIPTION
31:0	**CLKREF**	R	4MHz CLKREF frequency in Hz.

1Note that this field may be set to 0xFFFF on some devices. If this is the case, use a value of 25.
2Note that this field may be set to 0xFFFF on some devices. If this is the case, use a value of 614.

3.1.9. SCLK

Register 3-8. SCLK (SCLK Frequency Value, 0x0010 003A)

BIT	NAME	ACCESS	DESCRIPTION
15:0	**SCLK**	R	32kHz clock frequency in Hz (e.g., 0x7DC6 = 32,198 Hz)

3.1.10. PACIDR

Register 3-9. PACIDR (PAC part number and revision, 0x0010 0044)

BIT	NAME	ACCESS	DESCRIPTION
23:0	**PACIDR**	R	Device part number and revision.

3.1.11. UNIQUEID

Register 3-10. UNIQUEID (96-bit Unique ID, 0x0010 0060)

BIT	NAME	ACCESS	DESCRIPTION
95:0	**PACIDR**	R	96-bit device unique ID.

4. SYSTEM CLOCK CONTROL

4.1. Register

4.1.1. Register Map

Table 4-1. System Clock Control Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
System Clock Control			
0x4000 0000	**CCSCTL**	System clock control	0x0000 0000
0x4000 0004	**PLLCTL**	PLL control	0x0000 0000
0x4000 0008	**OSCCTL**	Ring oscillator control	0x0000 0007
0x4000 000C	**XTALCTL**	Crystal driver control	0x0000 0000

4.1.2. CCSCTL

Register 4-1. CCSCTL (System Clock Control, 0x4000 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7	**FCLK**	RW	0x0	FCLK input clock select 1b: PLLOUT clock 0b: FRCLK
6:5	**HCLKDIV**	RW	0x0	HCLK divider 11b = FCLK / 8 10b = FCLK / 4 01b = FCLK / 2 00b = FCLK / 1
4:2	**ACLKDIV**	RW	0x0	ACLK divider 111b = FCLK / 128 110b = FCLK / 44 101b = FCLK / 32 100b = FCLK / 16 011b = FCLK / 8 010b = FCLK / 4 001b = FCLK / 2 000b = FCLK / 1
1:0	**CLKIN**	R/W	0x0	FRCLK input clock select 11b = XTAL driver XIN/XOUT 10b = EXTCLK input 01b = CLKREF input (4MHz trimmed RC oscillator) 00b = internal ring oscillator ROSC

4.1.3. PLLCTL

Register 4-2. PLLCTL (PLL Control, 0x4000 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**Reserved**	R	0x0	Reserved
23:20	**Reserved**	RW	0x0	Reserved, must be set to 0x0
19:16	**PLLOUTDIV**	RW	0x0	PLL output divider 1111b: / 15 ... 0001b: / 1 0000b: reserved
15:7	**PLLFBDIV**	RW	0x0	PLL feedback divider 1 1111 1111b: / 513 ... 0 0000 0001b: / 3 0 0000 0000b: / 2
6:2	**PLLINDIV**	RW	0x0	PLL input divider 1 1111b: / 33 ... 0 0001b: / 3 0 0000b: / 2
1	**Reserved**	RW	0x0	Reserved, must be set to 0x0
0	**PLLEN**	RW	0x0	PLL oscillator 1b: enable PLL 0b: disable PLL

4.1.4. OSCCTL

Register 4-3. OSCCTL (Ring Oscillator Control, 0x4000 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:3	**Reserved**	R	0x0	Reserved
2:1	**ROSCP**	RW	0x3	Ring oscillator frequency setting 11b = 8.3MHz 10b = 10.7MHz 01b = 15.3MHz 00b = 28.7MHz
0	**ROSCEN**	RW	0x1	Enable Ring oscillator 1b: enable ROSC 0b: disable ROSC

4.1.5. XTALCTL

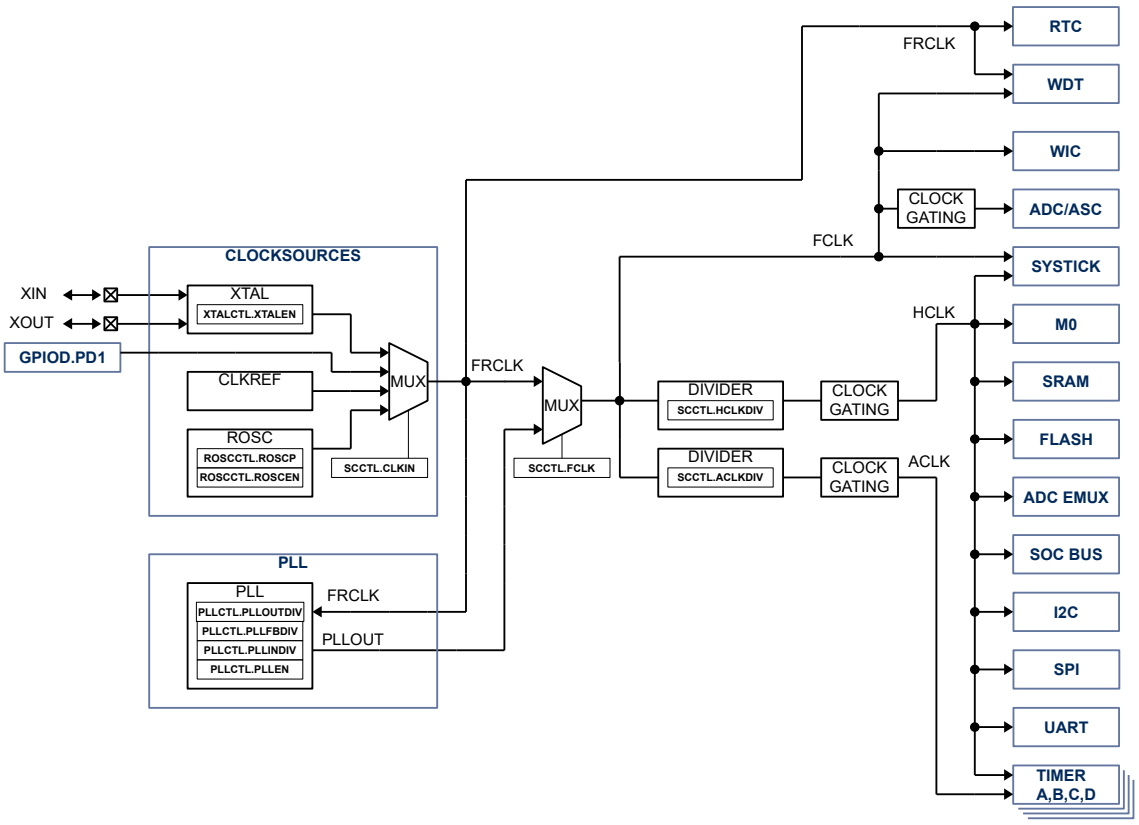
Register 4-4. XTALCTL (Crystal Driver Control, 0x4000 000C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:1	**Reserved**	R	0x0	Reserved
0	**XTALEN**	RW	0x0	Enable XTAL driver 1b: enable crystal driver 0b: disable crystal driver

4.2. Details of Operation

4.2.1. Block Diagram

Figure 4-1. System Clock Control



4.2.2. Configuration

Following blocks need to be configured for correct use of the system clock control:

- GPIOD.PD1
- RTC
- WDT
- WIC
- ADC/ASC
- SYSTICK
- SRAM
- FLASH
- ADC EMUX
- SOC BUS
- I2C

- SPI
- UART
- Timer A, B, C, D

4.2.3. ROSC

The internal ring oscillator has four frequency settings controllable with **OSCCTL.ROSCP** from 8.3MHz to 28.7MHz in four steps. The ROSC can also be disabled using **OSCCTL.ROSCEN**.

4.2.4. CLKREF

The CLKREF provides a 2% trimmed 4MHz clock.

4.2.5. XTAL

The crystal driver supports a range of crystal frequencies from 2MHz to 10MHz. The crystal driver can also be used to input an external clock using XIN.

The crystal driver can be enabled with **XTALCTL.XTALEN**.

4.2.6. EXTCLK

PD1 can be configured as clock input, EXTCLK.

4.2.7. PLL

The clock input to the PLL is FRCLK.

The PLL can be enabled with **PLLCTL.PLLEN**.

The PLL output clock PLLOUT is following following equation:

$$PLLOUT = \frac{PLLIN * PLLFBDIV}{PLLINDIV * PLLOUTDIV * 2} \quad (1)$$

Where:

PLLOUT: PLL output frequency in MHz

PLLIN: PLL input frequency in MHz (FRCLK)

PLLINDIV: PLL input divider (2 to 33) **PLLCTL.PLLINDIV**

PLLFBDIV: PLL feedback divider (2 to 513) **PLLCTL.PLLFBDIV**

PLLOUTDIV: PLL output divider (1 to 16) **PLLCTL.PLLOUTDIV**

The input clock frequency and input clock divider selection must follow formula below for correct operation of PLL:

$$1\text{MHz} \le \frac{PLLIN}{PLLINDIV} \le 25\text{MHz} \quad (2)$$

The output clock frequency and output clock divider selection must following formula below for correct operation of the PLL

$$\begin{gathered} 100\text{MHz} \le PLLOUT * PLLOUTDIV \le 250\text{MHz} \\ PLLOUTDIV \ge 1 \end{gathered} \quad (3)$$

The table below shows pre-calculated PLL output frequency settings using the 4MHz ROSC as input.

Table 4-5. PLL output frequency settings using 4MHz ROSC as input

PLL output	PLLOUTDIV	PLLFBDIV	PLLINDIV
10MHz	0x01 (/[1*2])	0x008 (*10)	0x0 (/2)
16.8MHz	0x05 (/[5*2])	0x052 (*84)	0x0 (/2)
20MHz	0x01 (/[1*2])	0x012 (*20)	0x0 (/2)
25MHz	0x01 (/[1*2])	0x019 (*25)	0x0 (/2)
30MHz	0x01 (/[1*2])	0x01C (*30)	0x0 (/2)
33.333MHz	0x01 (/[1*2])	0x030 (*50)	0x1 (/3)
40MHz	0x01 (/[1*2])	0x026 (*40)	0x0 (/2)
50MHz	0x01 (/[1*2])	0x030 (*50)	0x0 (/2)
60MHz	0x01 (/[1*2])	0x03A (*60)	0x0 (/2)
70MHz	0x01 (/[1*2])	0x044 (*70)	0x0 (/2)
80MHz	0x01 (/[1*2])	0x04E (*80)	0x0 (/2)
90MHz	0x01 (/[1*2])	0x058 (*90)	0x0 (/2)
100MHz	0x01 (/[1*2])	0x062 (*100)	0x0 (/2)

4.2.8. FRCLK

The free running clock FRCLK clock source can be selected with **CCSCTL.CLKIN**. From XTAL, EXTCLK, CLKREF, or ROSC.

4.2.9. FCLK

The fast clock FCLK clock source can be selected with **CCSCTL.FCLK** to be either PLLOUT of FRCLK.

4.2.10. HCLK

The high speed clock HCLK clock source input is FCLK. The HCLK can be divided down from FCLK using **CCSCTL.HCLKDIV** from /1 to /8 in 4 steps.

4.2.11. ACLK

The auxiliary clock ACLK clock source input is FCLK. The ACLK can be divided down from FCLK from ./1 to / 128 in 8 steps.

4.2.12. Clock Gating

When the CPU enters deep sleep mode, the HCLK, ACLK and FCLK to the ADC are clock gated and stopped to save power. Only FRCLK and FCLK continue to run to supply WIC, RTC, and WDT.

5. WATCHDOG TIMER

5.1. Register

5.1.1. Register Map

Table 5-1. Watchdog Timer Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Watchdog Timer			
0x4003 0000	**WDTCTL**	Watchdog timer control	0x6300 0000
0x4003 0004	**WDTCDV**	Watchdog timer count-down value	0x63FF FFFF
0x4003 0008	**WDTCTR**	Watchdog timer counter	0x00FF FFFF

5.1.2. WDTCTL

Register 5-1. WDTCTL (Watchdog Timer Control, 0x4003 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**KEY**	RW	0x63	Key for writing WDTCTL register 0x4A: allow writes to WDTCTL register 0x63: read value of WDTCTL.KEY
23:12	**Reserved**	R	0x0	Reserved
11	**WRBUSY**	RW	0x0	WDTCTL write busy 1b = write to WDTCTL still being processed. Any register writes received while this bit is set to a 1b will be dropped and not written to the given register. Any reads will return indeterminate data. 0b = WDT registers not busy
10	**WDTCLKSEL**	RW	0x0	WDT input clock select 1b: FCLK 0b: FRCLK
9:6	**WDTCLKDIV**	RW	0x0	WDT clock input divider 1111b: /65536 1110b: /32768 1101b: /16384 1100b: /8192 1011b: /4096 1010b: /2048 1001b: /1024 1000b: /512 0111b: /256 0110b: /128 0101b: /64 0100b: /32 0011b: /16 0010b: /8 0001b: /4 0000b: /2
5	**WDTRESETEN**	RW	0x0	Watchdog device RESET enable 1b = WDT trigger device RESET when WDTCTR register counts to 0x0 0b = WDT trigger device RESET disabled
4	**WDTINT**	RW	0x0	Watchdog interrupt 1b = WDT interrupt 0b = no WDT interrupt
3	**WDTINTEN**	RW	0x0	Watchdog interrupt enable 1b = enable WDT interrupt 0b = disable WDT interrupt

BIT	NAME	ACCESS	RESET	DESCRIPTION
2:0	**WDTCTRRST**	RW	0x0	WDTCTR counter reset 101b = write of 101b reset WDTCTR to WDTCDV value

5.1.3. WDTCDV

Register 5-2. WDTCDV (Watchdog Timer Count-Down Value, 0x4003 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**KEY**	RW	0x63	Key for writing WDTCDV register 0x4A: allow writes to WDTCDV register 0x63: read value of WDTCDV.KEY
23:0	**RSTVALUE**	RW	0xFF FFFF	24-bit WDT count-down value

5.1.4. WDTCTR

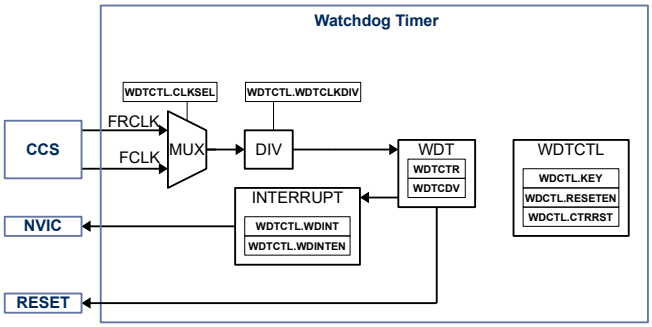
Register 5-3. WDTCTR (Watchdog Timer Counter, 0x4003 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**Reserved**	R	0x0	Reserved
23:0	**CTR**	RW	0xFF FFFF	Current value of WDT

5.2. Details of Operation

5.2.1. Block Diagram

Figure 5-1. WDT



5.2.2. Configuration

Following blocks need to be configured for correct use of the WDT:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)

5.2.3. Watchdog Timer

The Watchdog timer consist of a 24-bit timer and reset logic. The WDT can be used as general purpose 16bit timer or as watchdog timer that need to be serviced periodically to avoid device reset.

5.2.4. Access WDT Registers

The **WDTCTL** and **WDTCDV** registers can only be written to if **WDTCTL.KEY** or **WDTCDV.KEY** are set to 0x4A.

The read back value of **WDTCTL.KEY** or **WDTCDV.KEY** is always 0x63. The watchdog timer has 2 clock domains, HCLK and WDT clock domain set by **WDTCTL.CLKSEL** and **WDTCTL.WDTCLKDIV**. Writing to any WDT registers may take up to 1 clock cycle on the WDT clock domain to finish. Any ongoing writes to WDT registers are shown with **WDTCTL.WRBUSY**. As long as **WDTCTL.WRBUSY** is 1b, any subsequent writes to WDT registers are ignored and reads only provide undetermined data.

5.2.5. WDT Clock Setting

The WDT can use 2 clocks FCLK or FRCLK, selectable with **WDTCTL.CLKSEL**. For applications where the WDT need to run in CPU sleep mode, FRCLK should be used. The clock input can be further divided down from /2 to /65536 using **WDTCTL.WDTCLKDIV**.

5.2.6. General Purpose Timer Mode

Set **WDTCTL.WDTRESETEN** to 0b to use the WDT as general purpose 24-bit timer. Set the desired count value in WDT clocks in **WDTCDV.RSTVALUE**, set **WDTCTL.WDTCTRRST** to 101b to load **WDTCTR.CTR** with **WDTCDV.RSTVALUE**. To start the GPT timer set **GPTCTL.INTEN**. When **WDTCTR.CTR** reaches 0x0, the timer automatic reloads **WDTCTR.CTR** with **WDTCDV.RSTVALUE** and continues countdown. The WDT is stopped when **WDTCTL.INTEN** is cleared.

5.2.7. Watchdog Timer Mode

Set **WDTCTL.WDTRESETEN** to 1b to use the WDT as 24-bit watchdog timer with device reset capability. Set **WDTCTL.WDTINTEN** to 1b to enable interrupt when WDT counts to 0x0. Set the desired count value in WDT clocks in **WDTCDV.RSTVALUE**. To start the WDT count down, set **WDTCTL.WDTCTRRST** to 101b. The WDT will copy the **WDTCTL.RSTVALUE** to **WDTCTR.CTR** and start counting down. When **WDTCTR.CTR** reaches 0x0, the WDT will automatic copy **WDTCTL.RSTVALUE** to **WDTCTR.CTR**, restart the counter and set the interrupt flag if enabled is set. During the second count down, set **WDTCTL.WDTCTRRST** to 101b to restart the 1st WDT countdown and avoid device reset. If the **WDTCTR.CTR** reaches 0x0 during the second count down without being reloaded, the WDT will toggle device reset.

6. GENERAL PURPOSE TIMER

6.1. Register

6.1.1. Register Map

Table 6-1. General Purpose Timer Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Real Time Clock			
0x4004 0000	**RTCCTL**	Real-time clock timer control	0x6300 0000
0x4004 0004	**RTCCDV**	Real-time clock timer count-down value	0x63FF FFFF
0x4004 0008	**RTCCTR**	Real-time clock timer counter	0x00FF FFFF

6.1.2. RTCCTL

Register 6-1. RTCCTL (Real Time Clock Control, 0x4004 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**KEY**	RW	0x63	Key for writing **GPTCTL** register 0x4A: allow writes to **GPTCTL** register 0x63: read value of **GPTCTL.KEY**
23:12	**Reserved**	R	0x0	Reserved
11	**WRBUSY**	RW	0x0	**GPTCTL** write busy 1b = write to **GPTCTL** still being processed. Any register writes received while this bit is set to a 1b will be dropped and not written to the given register. Any reads will return indeterminate data. 0b = GPT registers not busy
10	**Reserved**	RW	0x0	Reserved, write as 0
9:6	**GPTCLKDIV**	RW	0x0	GPT clock input divider 1111b: /65536 1110b: /32768 1101b: /16384 1100b: /8192 1011b: /4096 1010b: /2048 1001b: /1024 1000b: /512 0111b: /256 0110b: /128 0101b: /64 0100b: /32 0011b: /16 0010b: /8 0001b: /4 0000b: /2
5	**Reserved**	R	0x0	Reserved
4	**GPTINT**	RW	0x0	Real time clock interrupt 1b = GPT interrupt 0b = no GPT interrupt
3	**GPTINTEN**	RW	0x0	Real time clock interrupt enable 1b = enable GPT interrupt 0b = disable GPT interrupt
2:0	**GPTCTRRST**	RW	0x0	**GPTCTR** counter reset 101b = write of 101b reset **GPTCTR** to **GPTCDV** value

6.1.3. RTCCDV

Register 6-2. RTCCDV (Real Time Clock Count-Down Value, 0x4004 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**KEY**	RW	0x63	Key for writing **GPTCTL** register 0x4A: allow writes to **GPTCTL** register 0x63: read value of **GPTCTL.KEY**
23:0	**RSTVALUE**	RW	0xFF FFFF	24bit GPT count-down value

6.1.4. RTCCTR

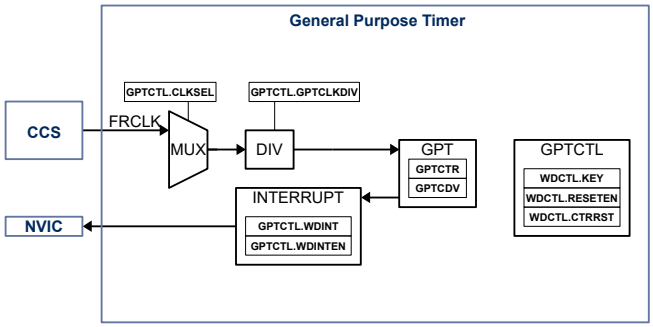
Register 6-3. RTCCTR (Real Time Clock Counter, 0x4004 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**Reserved**	R	0x0	Reserved
23:0	**CTR**	RW	0xFF FFFF	Current value of GPT

6.2. Details of Operation

6.2.1. Block Diagram

Figure 6-1. GPT



6.2.2. Configuration

Following blocks need to be configured for correct use of the GPT:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)

6.2.3. General Purpose Timer

The General purpose timer consist of a 24-bit timer, can can also run in device sleep mode if FRLCK is used.

6.2.4. Access GPT Registers

The **RTCCTL** and **RTCCDV** registers can only be written to if **RTCCTL.KEY** or **RTCCDV.KEY** are set to 0x4A.

The read back value of **RTCCTL.KEY** or **RTCCDV.KEY** is always 0x63. The general purpose timer is supplied by the FRCLK. The GPT may divide this input clock by using the **RTCCTL.GPTCLKDIV**. Writing to any GPT registers may take up to 1 clock cycle on the GPT clock domain to finish. Any ongoing writes to GPT registers are shown with **RTCCTL.WRBUSY**. As long as **RTCCTL.WRBUSY** is 1b, any subsequent writes to GPT registers are ignored and reads only provide undetermined data.

6.2.5. GPT Clock

The GPT uses FRCLK as its input clock. For applications where the GPT need to run in CPU sleep mode, FRCLK should be used. The clock input can be further divided down from /2 to /65536 using **RTCCTL.GPTCLKDIV**.

6.2.6. General Purpose Timer Mode

Set **RTCCTL.GPTRESETEN** to 0b to use the GPT as general purpose 24-bit timer. Set the desired count value in GPT clocks in **RTCCDV.RSTVALUE**, set **RTCCTL.GPTCTRRST** to 101b to load **RTCCTR.CTR** with **RTCCDV.RSTVALUE**. To start the GPT timer set **RTCCTL.INTEN**. When **RTCCTR.CTR** reaches 0x0, the timer automatic reloads **RTCCTR.CTR** with **RTCCDV.RSTVALUE** and continues countdown. The GPT is stopped when **RTCCTL.INTEN** is cleared.

7. GPIO PORT A

7.1. Register

7.1.1. Register Map

Table 7-1. GPIO Port A Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
GPIO Port A			
0x4007 0000	**GPIOAOUT**	GPIO Port A output	0x0000 0000
0x4007 0004	**GPIOAOUTEN**	GPIO Port A output enable	0x0000 0000
0x4007 0008	**GPIOADS**	GPIO Port A output drive strength	0x0000 0000
0x4007 000C	**GPIOAPU**	GPIO Port A output weak pull up	0x0000 0000
0x4007 0010	**GPIOAPD**	GPIO Port A output weak pull down	0x0000 0000
0x4007 0014	**GPIOAIN**	GPIO Port A input	0x0000 0000
0x4007 0018	**Reserved**	Reserved	0x0000 0000
0x4007 001C	**GPIOAPSEL**	GPIO Port A peripheral select	0x0000 0000
0x4007 0020	**GPIOAINTP**	GPIO Port A interrupt polarity select	0x0000 0000
0x4007 0024	**GPIOAINTE**	GPIO Port A interrupt enable select	0x0000 0000
0x4007 0028	**GPIOAINTF**	GPIO Port A interrupt flag	0x0000 0000
0x4007 002C	**GPIOAINTM**	GPIO Port A interrupt mask	0x0000 0000

7.1.2. GPIOAO

Register 7-1. GPIOAOUT (GPIO Port A Output, 0x4007 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A output 7 1b: set output high if **GPIOAOUTEN.P7** = 1b 0b: set output low if **GPIOAOUTEN.P7** = 1b
6	**P6**	RW	0x0	Port A output 6 1b: set output high if **GPIOAOUTEN.P6** = 1b 0b: set output low if **GPIOAOUTEN.P6** = 1b
5	**P5**	RW	0x0	Port A output 5 1b: set output high if **GPIOAOUTEN.P5** = 1b 0b: set output low if **GPIOAOUTEN.P5** = 1b
4	**P4**	RW	0x0	Port A output 4 1b: set output high if **GPIOAOUTEN.P4** = 1b 0b: set output low if **GPIOAOUTEN.P4** = 1b
3	**P3**	RW	0x0	Port A output 3 1b: set output high if **GPIOAOUTEN.P3** = 1b 0b: set output low if **GPIOAOUTEN.P3** = 1b
2	**P2**	RW	0x0	Port A output 2 1b: set output high if **GPIOAOUTEN.P2** = 1b 0b: set output low if **GPIOAOUTEN.P2** = 1b
1	**P1**	RW	0x0	Port A output 1 1b: set output high if **GPIOAOE.P1** = 1b 0b: set output low if **GPIOAOE.P1** = 1b

BIT	NAME	ACCESS	RESET	DESCRIPTION
0	**P0**	RW	0x0	Port A output 0 1b: set output high if **GPIOAOE.P0** = 1b 0b: set output low if **GPIOAOE.P0** = 1b

7.1.3. GPIOAOUTEN

Register 7-2. GPIOAOUTEN (GPIO Port A Output Enable, 0x4007 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A output 7 enable 1b: output state set by **GPIOAOUT.P7** 0b: output disabled, high-impedance state
6	**P6**	RW	0x0	Port A output 6 enable 1b: output state set by **GPIOAOUT.P6** 0b: output disabled, high-impedance state
5	**P5**	RW	0x0	Port A output 5 enable 1b: output state set by **GPIOAOUT.P5** 0b: output disabled, high-impedance state
4	**P4**	RW	0x0	Port A output 4 enable 1b: output state set by **GPIOAOUT.P4** 0b: output disabled, high-impedance state
3	**P3**	RW	0x0	Port A output 3 enable 1b: output state set by **GPIOAOUT.P3** 0b: output disabled, high-impedance state
2	**P2**	RW	0x0	Port A output 2 enable 1b: output state set by **GPIOAOUT.P2** 0b: output disabled, high-impedance state
1	**P1**	RW	0x0	Port A output 1 enable 1b: output state set by **GPIOAOUT.P1** 0b: output disabled, high-impedance state
0	**P0**	RW	0x0	Port A output 0 enable 1b: output state set by **GPIOAOUT.P0** 0b: output disabled, high-impedance state

7.1.4. GPIOADS

Register 7-3. GPIOADS (GPIO Port A Output Drive Strength, 0x4007 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A output 7 drive strength select 1b: high 0b: low
6	**P6**	RW	0x0	Port A output 6 drive strength select 1b: high 0b: low
5	**P5**	RW	0x0	Port A output 5 drive strength select 1b: high 0b: low
4	**P4**	RW	0x0	Port A output 4 drive strength select 1b: high 0b: low

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**P3**	RW	0x0	Port A output 3 drive strength select 1b: high 0b: low
2	**P2**	RW	0x0	Port A output 2 drive strength select 1b: high 0b: low
1	**P1**	RW	0x0	Port A output 1 drive strength select 1b: high 0b: low
0	**P0**	RW	0x0	Port A output 0 drive strength select 1b: high 0b: low

7.1.5. GPIOAPU

Register 7-4. GPIOAPU (GPIO Port A Weak Pull Up, 0x4007 000C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
6	**P6**	RW	0x0	Port A 6 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
5	**P5**	RW	0x0	Port A 5 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
4	**P4**	RW	0x0	Port A 4 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
3	**P3**	RW	0x0	Port A 3 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
2	**P2**	RW	0x0	Port A 2 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
1	**P1**	RW	0x0	Port A 1 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
0	**P0**	RW	0x0	Port A 0 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO

7.1.6. GPIOAPD

Register 7-5. GPIOAPD (GPIO Port A Weak Pull Down, 0x4007 0010)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
6	**P6**	RW	0x0	Port A 6 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
5	**P5**	RW	0x0	Port A 5 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
4	**P4**	RW	0x0	Port A 4 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
3	**P3**	RW	0x0	Port A 3 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
2	**P2**	RW	0x0	Port A 2 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
1	**P1**	RW	0x0	Port A 1 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
0	**P0**	RW	0x0	Port A 0 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS

7.1.7. GPIOAIN

Register 7-6. GPIOAIN (GPIO Port A Input, 0x4007 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 input state 1b: input high 0b: input low
6	**P6**	RW	0x0	Port A 6 input state 1b: input high 0b: input low
5	**P5**	RW	0x0	Port A 5 input state 1b: input high 0b: input low
4	**P4**	RW	0x0	Port A 4 input state 1b: input high 0b: input low
3	**P3**	RW	0x0	Port A 3 input state 1b: input high 0b: input low
2	**P2**	RW	0x0	Port A 2 input state 1b: input high 0b: input low

BIT	NAME	ACCESS	RESET	DESCRIPTION
1	**P1**	RW	0x0	Port A 1 input state 1b: input high 0b: input low
0	**P0**	RW	0x0	Port A 0 input state 1b: input high 0b: input low

7.1.8. GPIOAPSEL

Register 7-7. GPIOAPSEL (GPIO Port A Peripheral Select, 0x4007 001C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RW	0x0	Reserved
15:14	**P7**	RW	0x0	Port A 7 peripheral select 11b: PWMC1 / DTGC0HS output or CC1 capture and compare input 10b: PWMA7 / DTGA3HS output or CA7 capture and compare input 01b: PWMA5 / DTGA1HS output or CA5 capture and compare input 00b: I/O mode PA7
13:12	**P6**	RW	0x0	Port A 6 peripheral select 11b: reserved 10b: PWMB0 / DTGB0LS output or CB0 capture and compare input 01b: PWMA4 / DTGA0HS output or CA4 capture and compare input 00b: I/O mode PA6
11:10	**P5**	RW	0x0	Port A 5 peripheral select 11b: reserved 10b: PWMD0 / DTGD0LS output or CD0 capture and compare input / IBCTL5 01b: PWMA6 / DTGA2HS output or CA6 capture and compare input / IBCTL5 00b: I/O mode PA5
9:8	**P4**	RW	0x0	Port A 4 peripheral select 11b: reserved 10b: PWMC0 / DTGC0LS output or CC0 capture and compare input / IBCTL4 01b: PWMA5 / DTGA1HS output or CA5 capture and compare input IBCTL4 00b: I/O mode PA4
7:6	**P3**	RW	0x0	Port A 3 peripheral select 11b: PWMB0 / DTGB0LS output or CB0 capture and compare input / IBCTL3 10b: PWMA4 / DTGA0HS output or CA4 capture and compare input / IBCTL3kl 01b: PWMA3 / DTGA3LS output or CA3 capture and compare input / IBCTL3 00b: I/O mode PA3
5:4	**P2**	RW	0x0	Port A 2 peripheral select 11b: reserved 10b: reserved 01b: PWMA2 / DTGA2LS output or CA2 capture and compare input / IBCTL2 00b: I/O mode PA2
3:2	**P1**	RW	0x0	Port A 1 peripheral select 11b: reserved 10b: reserved 01b: PWMA1 / DTGA1LS output or CA1 capture and compare input / IBCTL1 00b: I/O mode PA1

BIT	NAME	ACCESS	RESET	DESCRIPTION
1:0	**P0**	RW	0x0	Port A 0 peripheral select 11b: reserved 10b: reserved 01b: PWMA0 / DTGA0LS output or CA0 capture and compare input / IBCTL0 00b: I/O mode PA0

7.1.9. GPIOAINTP

Register 7-8. GPIOAINTP (GPIO Port A Interrupt Polarity, 0x4007 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
6	**P6**	RW	0x0	Port A 6 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
5	**P5**	RW	0x0	Port A 5 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
4	**P4**	RW	0x0	Port A 4 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
3	**P3**	RW	0x0	Port A 3 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
2	**P2**	RW	0x0	Port A 2 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
1	**P1**	RW	0x0	Port A 1 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
0	**P0**	RW	0x0	Port A 0 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

7.1.10. GPIOAINTE

Register 7-9. GPIOAINTE (GPIO Port A Interrupt Enable, 0x4007 0024)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 interrupt enable 1b: enabled interrupt 0b: disable interrupt
6	**P6**	RW	0x0	Port A 6 interrupt enable 1b: enabled interrupt 0b: disable interrupt
5	**P5**	RW	0x0	Port A 5 interrupt enable 1b: enabled interrupt 0b: disable interrupt

BIT	NAME	ACCESS	RESET	DESCRIPTION
4	**P4**	RW	0x0	Port A 4 interrupt enable 1b: enabled interrupt 0b: disable interrupt
3	**P3**	RW	0x0	Port A 3 interrupt enable 1b: enabled interrupt 0b: disable interrupt
2	**P2**	RW	0x0	Port A 2 interrupt enable 1b: enabled interrupt 0b: disable interrupt
1	**P1**	RW	0x0	Port A 1 interrupt enable 1b: enabled interrupt 0b: disable interrupt
0	**P0**	RW	0x0	Port A 0 interrupt enable 1b: enabled interrupt 0b: disable interrupt

7.1.11. GPIOAINTF

Register 7-10. GPIOAINTF (GPIO Port A Interrupt Flag, 0x4007 0028)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port A 7 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
6	**P6**	RW	0x0	Port A 6 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
5	**P5**	RW	0x0	Port A 5 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
4	**P4**	RW	0x0	Port A 4 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
3	**P3**	RW	0x0	Port A 3 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
2	**P2**	RW	0x0	Port A 2 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
1	**P1**	RW	0x0	Port A 1 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
0	**P0**	RW	0x0	Port A 0 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending

7.1.12. GPIOAINTM

Register 7-11. GPIOAINTM (GPIO Port A Interrupt Mask, 0x4007 002C)

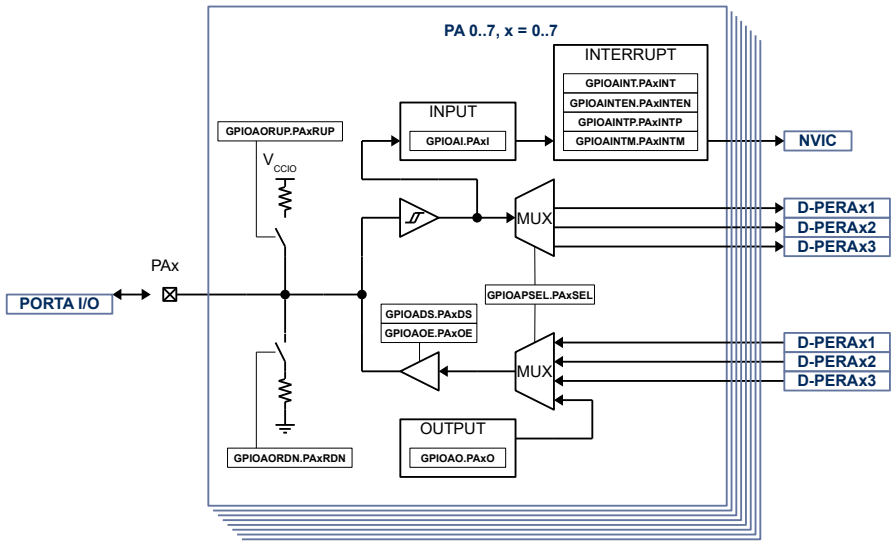
BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**P7**	RW	0x0	Port A 7 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
6	**P6**	RW	0x0	Port A 6 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
5	**P5**	RW	0x0	Port A 5 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
4	**P4**	RW	0x0	Port A 4 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
3	**P3**	RW	0x0	Port A 3 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
2	**P2**	RW	0x0	Port A 2 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
1	**P1**	RW	0x0	Port A 1 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
0	**P0**	RW	0x0	Port A 0 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

7.2. Details of Operation

7.2.1. Block Diagram

Figure 7-1. GPIO Port A



7.2.2. Configuration

Following blocks need to be configured for correct use of the GPIO A:

- Nested Vectored Interrupt Controller (NVIC)
- Gate Driver
- Timer A, PWMA, DTGA
- Timer B, PWMB, DTGB
- Timer C, PWMC, DTGC
- Timer D, PWMD, DTGD
- General Purpose Gate Drivers

7.2.3. GPIO A Block

The GPIO A block consists of up to 8 general purpose input output (GPIO). Each GPIO has interrupt capabilities, weak pull-up or pull-down, programmable output drive strength, High-Z output operation. Some of the GPIO can be configured as PWM output, or capture and compare input.

7.2.4. Input

The input state of GPIOA can be monitored with **GPIOAIN.Px**. The input state can be monitored regardless of the peripheral select setting **GPIOAPSEL**.

7.2.5. Output and Output Enable

When **GPIOAOUTEN.Px** is enabled, the output state is controlled by **GPIOAOUT.Px**.

When **GPIOAOUTEN.Px** is disabled, the output is in High-Z state.

7.2.6. Output Drive Strength

The output drive strength can be adjusted using **GPIOADS** to meet application needs. Set **GPIOADS.Px** to enable high current drive strength, reset to enable low current drive strength.

7.2.7. Weak Pull Up and Pull Down

Independent from the output settings, weak pull up can be enabled with **GPIOAPU** and weak pull down can be enabled with **GPIOAPD**.

NOTE:

GPIOAPU.Pxor **GPIOAPD.Px** should never be enabled at the same time for a single GPIO. If switching from weak pull-up to weak pull-down is required, disable weak pull-up first before enablle weak pull-down and vice versa.

7.2.8. Peripheral Select

Each GPIO is connected to up to 4 digital peripherals, selectable with **GPIOAPSEL**. When a different function than IO is selected the input state can still be read with **GPIOAIN** and the pull-up and pull-down is still controllable.

7.2.9. Interrupt

The interrupt for each GPIO can be enabled with **GPIOAINTE**. The interrupt can be configured to be rising signal edge or falling signal edge using **GPIOAINTP**. The state of the interrupt can be read from **GPIOAINTF**. The individual interrupt bits can be cleared by writing to 1.

When the GPIO interrupts are enabled for the first time after device start-up, it may be in an uncertain state and generate an interrupt. To avoid this the **GPIOAINTM** mask bit need to be set before enabled interrupt bits.

To allow interrupt to be recognized by the CPU the GPIO interrupt need also be enabled in the NVIC.

8. GPIO PORT B

8.1. Register

8.1.1. Register Map

Table 8-1. GPIO Port B Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
GPIO Port B			
0x4007 0040	**GPIOBOUT**	GPIO Port B output	0x0000 0000
0x4007 0044	**GPIOBOUTEN**	GPIO Port B output enable	0x0000 0000
0x4007 0048	**GPIOBODS**	GPIO Port B output drive strength	0x0000 0000
0x4007 004C	**GPIOBPU**	GPIO Port B output weak pull up	0x0000 0000
0x4007 0050	**GPIOBPD**	GPIO Port B output weak pull down	0x0000 0000
0x4007 0054	**GPIOBIN**	GPIO Port B input	0x0000 0000
0x4007 0058	**Reserved**	Reserved	0x0000 0000
0x4007 005C	**GPIOBPSEL**	GPIO Port B peripheral select	0x0000 0000
0x4007 0060	**GPIOBINTP**	GPIO Port B interrupt polarity select	0x0000 0000
0x4007 0064	**GPIOBINTE**	GPIO Port B interrupt enable select	0x0000 0000
0x4007 0068	**GPIOBINTF**	GPIO Port B interrupt flag	0x0000 0000
0x4007 006C	**GPIOBINTM**	GPIO Port B interrupt mask	0x0000 0000

8.1.2. GPIOBOUT

Register 8-1. GPIOBOUT (GPIO Port B Output, 0x4007 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Reserved, must be written to 0b
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Reserved, must be written to 0b

8.1.3. GPIOBOUTEN

Register 8-2. GIOBOUTEN (GPIO Port B Output Enable, 0x4007 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0	Reserved
7	**P7**	RW	0x0	Reserved, must be written to 0b
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b

BIT	NAME	ACCESS	RESET	DESCRIPTION
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Reserved, must be written to 0b

8.1.4. GPIOBDS

Register 8-3. GPIOBDS (GPIO Port B Output Drive Strength, 0x4007 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Reserved, must be written to 0b
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Reserved, must be written to 0b

8.1.5. GPIOBPU

Register 8-4. GPIOBPU (GPIO Port B Weak Pull Up, 0x4007 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Reserved, must be written to 0b
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Reserved, must be written to 0b

8.1.6. GPIOBPD

Register 8-5. GPIOBPD (GPIO Port B Weak Pull Down, 0x4007 0050)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Reserved, must be written to 0b
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b

BIT	NAME	ACCESS	RESET	DESCRIPTION
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Reserved, must be written to 0b

8.1.7. GPIOBIN

Register 8-6. GPIOBIN (GPIO Port B Input, 0x4007 0054)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	R	0x0	Port B 7 input state 1b: input high 0b: input low
6	**P6**	R	0x0	Reserved
5	**P5**	R	0x0	Reserved
4	**P4**	R	0x0	Reserved
3	**P3**	R	0x0	Reserved
2	**P2**	R	0x0	Reserved
1	**P1**	R	0x0	Reserved
0	**P0**	R	0x0	Port B 0 input state 1b: input high 0b: input low

8.1.8. GPIOBPSEL

Register 8-7. GPIOBPSEL (GPIO Port B Peripheral Select, 0x4007 005C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RW	0x0	Reserved
15:14	**P7**	RW	0x0	Port B 7 peripheral select 11b: reserved 10b: reserved 01b: reserved 00b: IRQ2 / POS
13:12	**P6**	RW	0x0	Port B 6 peripheral select 11b: reserved 10b: reserved 01b: EMUXDATA 00b: reserved
11:10	**P5**	RW	0x0	Port B 5 peripheral select 11b: reserved 10b: reserved 01b: EMUXCLK 00b: reserved
9:8	**P4**	RW	0x0	Port B 4 peripheral select 11b: reserved 10b: reserved 01b: SOC Bus SCLK 00b: reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:6	**P3**	RW	0x0	Port B 3 peripheral select 11b: reserved 10b: reserved 01b: SOC Bus MOSI 00b: reserved
5:4	**P2**	RW	0x0	Port B 2 peripheral select 11b: reserved 10b: reserved 01b: SOC Bus MISO 00b: reserved
3:2	**P1**	RW	0x0	Port B 1 peripheral select 11b: reserved 10b: reserved 01b: SOC Bus CS 00b: reserved
1:0	**P0**	RW	0x0	Port B 0 peripheral select 11b: reserved 10b: reserved 01b: reserved 00b: IRQ1

8.1.9. GPIOBINTP

Register 8-8. GPGPIOIOBINTP (GPIO Port B Interrupt Polarity, 0x4007 0060)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	RW	0x0	Port B 7 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Port B 0 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

8.1.10. GPIOBINTE

Register 8-9. GPIOBINTE (GPIO Port B Interrupt Enable, 0x4007 0064)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	RW	0x0	Port B 7 interrupt enable 1b: enabled interrupt 0b: disable interrupt
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Port B 0 interrupt enable 1b: enabled interrupt 0b: disable interrupt

8.1.11. GPIOBINTF

Register 8-10. GPIOBINTF (GPIO Port B Interrupt Flag, 0x4007 0068)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	RW	0x0	Port B 7 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Port B 0 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending

8.1.12. GPIOBINTM

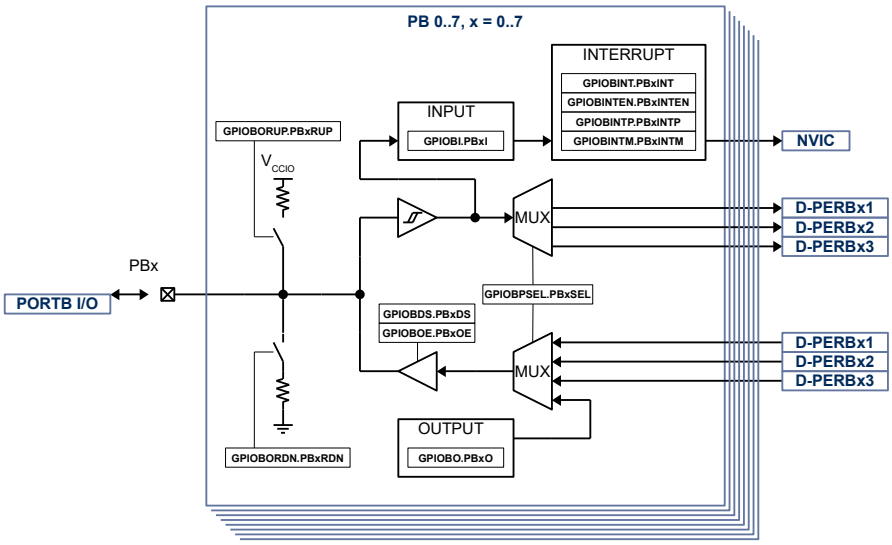
Register 8-11. GPIOBINTM (GPIO Port B Interrupt Mask, 0x4007 006C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port B 7 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
6	**P6**	RW	0x0	Reserved, must be written to 0b
5	**P5**	RW	0x0	Reserved, must be written to 0b
4	**P4**	RW	0x0	Reserved, must be written to 0b
3	**P3**	RW	0x0	Reserved, must be written to 0b
2	**P2**	RW	0x0	Reserved, must be written to 0b
1	**P1**	RW	0x0	Reserved, must be written to 0b
0	**P0**	RW	0x0	Port B 0 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

8.2. Details of Operation

8.2.1. Block Diagram

Figure 8-1. GPIO Port B



8.2.2. Configuration

Following blocks need to be configured for correct use of the GPIO B:

- Nested Vectored Interrupt Controller (NVIC)
- SOC Bus

8.2.3. GPIO B Block

The GPIO B block consists of up to 8 general purpose input output (GPIO). Each GPIO has interrupt capabilities, weak pull-up or pull-down, programmable output drive strength, High-Z output operation. Some of the GPIO can be configured as PWM output, or capture and compare input.

8.2.4. Input

The input state of GPIOB can be monitored with **GPIOBIN.Px**. The input state can be monitored regardless of the peripheral select setting **GPIOBPSEL**.

8.2.5. Output and Output Enable

When **GPIOBOUTEN.Px** is enabled, the output state is controlled by **GPIOBOUT.Px**.

When **GPIOBOUTEN.Px** is disabled, the output is in High-Z state.

8.2.6. Output Drive Strength

The output drive strength can be adjusted using **GPIOBDS** to meet application needs. Set **GPIOBDS.Px** to enable high current drive strength, reset to enable low current drive strength.

8.2.7. Weak Pull Up and Pull Down

Independent from the output settings, weak pull up can be enabled with **GPIOBPU** and weak pull down can be enabled with **GPIOBPD**.

NOTE:

GPIOBPU.Pxor **GPIOBPD.Px** should never be enabled at the same time for a single GPIO. If switching from weak pull-up to weak pull-down is required, disable weak pull-up first before enablle weak pull-down and vice versa.

8.2.8. Peripheral Select

Each GPIO is connected to up to 4 digital peripherals, selectable with **GPIOBPSEL**. When a different function than IO is selected the input state can still be read with **GPIOBIN** and the pull-up and pull-down is still controllable.

8.2.9. Interrupt

The interrupt for each GPIO can be enabled with **GPIOBINTE**. The interrupt can be configured to be rising signal edge or falling signal edge using **GPIOBINTP**. The state of the interrupt can be read from **GPIOBINTF**. The individual interrupt bits can be cleared by writing to 1.

When the GPIO interrupts are enabled for the first time after device start-up, it may be in an uncertain state and generate an interrupt. To avoid this the **GPIOBINTM** mask bit need to be set before enabled interrupt bits.

To allow interrupt to be recognized by the CPU the GPIO interrupt need also be enabled in the NVIC.

9. GPIO PORT C

9.1. Register

9.1.1. Register Map

Table 9-1. GPIO Port C Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
GPIO Port C			
0x4008 0000	**GPIOCOUT**	GPIO Port C output	0x0000 0000
0x4008 0004	**GPIOCOUTEN**	GPIO Port C output enable	0x0000 0000
0x4008 0008	**Reserved**	Reserved	0x0000 0000
0x4008 000C	**Reserved**	Reserved	0x0000 0000
0x4008 0010	**Reserved**	Reserved	0x0000 0000
0x4008 0014	**GPIOCIN**	GPIO Port C input	0x0000 0000
0x4008 0018	**GPIOCINE**	GPIO Port C input enable	0x0000 0000
0x4008 001C	**Reserved**	Reserved	0x0000 0000
0x4008 0020	**GPIOCINTP**	GPIO Port C interrupt polarity select	0x0000 0000
0x4008 0024	**GPIOCINTE**	GPIO Port C interrupt enable select	0x0000 0000
0x4008 0028	**GPIOCINTF**	GPIO Port C interrupt flag	0x0000 0000
0x4008 002C	**GPIOCINTM**	GPIO Port C interrupt mask	0x0000 0000

9.1.2. GPIOCOUT

Register 9-1. GPIOCOUT (GPIO Port C Output, 0x4008 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C output 7 1b: set output high if **GPIOCOUTEN.Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
6	**P6**	RW	0x0	Port C output 6 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
5	**P5**	RW	0x0	Port C output 5 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
4	**P4**	RW	0x0	Port C output 4 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
3	**P3**	RW	0x0	Port C output 3 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
2	**P2**	RW	0x0	Port C output 2 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b
1	**P1**	RW	0x0	Port C output 1 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b

BIT	NAME	ACCESS	RESET	DESCRIPTION
0	**P0**	RW	0x0	Port C output 0 1b: set output high if **GPIOCOUTEN. Px** = 1b 0b: set output low if **GPIOCOUTEN. Px** = 1b

9.1.3. GPIOCOUTEN

Register 9-2. GPIOCOUTEN (GPIO Port C Output Enable, 0x4008 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0	Reserved
7	**P7**	RW	0x0	Port C output 7 enable 1b: output state set by **GPIOCO.PCO7** 0b: output disabled, high-impedance state
6	**P6**	RW	0x0	Port C output 6 enable 1b: output state set by **GPIOCO.PCO6** 0b: output disabled, high-impedance state
5	**P5**	RW	0x0	Port C output 5 enable 1b: output state set by **GPIOCO.PCO5** 0b: output disabled, high-impedance state
4	**P4**	RW	0x0	Port C output 4 enable 1b: output state set by **GPIOCO.PCO4** 0b: output disabled, high-impedance state
3	**P3**	RW	0x0	Port C output 3 enable 1b: output state set by **GPIOCO.PCO3** 0b: output disabled, high-impedance state
2	**P2**	RW	0x0	Port C output 2 enable 1b: output state set by **GPIOCO.PCO2** 0b: output disabled, high-impedance state
1	**P1**	RW	0x0	Port C output 1 enable 1b: output state set by **GPIOCO.PCO1** 0b: output disabled, high-impedance state
0	**P0**	RW	0x0	Port C output 0 enable 1b: output state set by **GPIOCO.PCO0** 0b: output disabled, high-impedance state

9.1.4. GPIOCIN

Register 9-3. GPIOCIN (GPIO Port C Input, 0x4008 0018)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 input state 1b: input high 0b: input low
6	**P6**	RW	0x0	Port C 6 input state 1b: input high 0b: input low
5	**P5**	RW	0x0	Port C 5 input state 1b: input high 0b: input low
4	**P4**	RW	0x0	Port C 4 input state 1b: input high 0b: input low

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**P3**	RW	0x0	Port C 3 input state 1b: input high 0b: input low
2	**P2**	RW	0x0	Port C 2 input state 1b: input high 0b: input low
1	**P1**	RW	0x0	Port C 1 input state 1b: input high 0b: input low
0	**P0**	RW	0x0	Port C 0 input state 1b: input high 0b: input low

9.1.5. GPIOCINE

Register 9-4. GPIOCINE (GPIO Port C Input Enable, 0x4008 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 input enable 1b: input enabled, for I/O operation 0b: input disabled
6	**P6**	RW	0x0	Port C 6 input enable 1b: input enabled, for I/O operation 0b: input disabled
5	**P5**	RW	0x0	Port C 5 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD5 ADC input operation
4	**P4**	RW	0x0	Port C 4 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD4 ADC input operation
3	**P3**	RW	0x0	Port C 3 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD3 ADC input operation
2	**P2**	RW	0x0	Port C 2 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD2 ADC input operation
1	**P1**	RW	0x0	Port C 1 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD1 ADC input operation
0	**P0**	RW	0x0	Port C 0 input enable 1b: input enabled, for I/O operation 0b: input disabled, for AD0 ADC input operation

9.1.6. GPIOCINTP

Register 9-5. GPIOCINTP (GPIO Port C Interrupt Polarity, 0x4008 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

BIT	NAME	ACCESS	RESET	DESCRIPTION
6	**P6**	RW	0x0	Port C 6 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
5	**P5**	RW	0x0	Port C 5 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
4	**P4**	RW	0x0	Port C 4 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
3	**P3**	RW	0x0	Port C 3 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
2	**P2**	RW	0x0	Port C 2 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
1	**P1**	RW	0x0	Port C 1 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
0	**P0**	RW	0x0	Port C 0 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

9.1.7. GPIOCINTE

Register 9-6. GPIOCINTE (GPIO Port C Interrupt Enable, 0x4008 0024)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 interrupt enable 1b: enabled interrupt 0b: disable interrupt
6	**P6**	RW	0x0	Port C 6 interrupt enable 1b: enabled interrupt 0b: disable interrupt
5	**P5**	RW	0x0	Port C 5 interrupt enable 1b: enabled interrupt 0b: disable interrupt
4	**P4**	RW	0x0	Port C 4 interrupt enable 1b: enabled interrupt 0b: disable interrupt
3	**P3**	RW	0x0	Port C 3 interrupt enable 1b: enabled interrupt 0b: disable interrupt
2	**P2**	RW	0x0	Port C 2 interrupt enable 1b: enabled interrupt 0b: disable interrupt
1	**P1**	RW	0x0	Port C 1 interrupt enable 1b: enabled interrupt 0b: disable interrupt
0	**P0**	RW	0x0	Port C 0 interrupt enable 1b: enabled interrupt 0b: disable interrupt

9.1.8. GPIOCINTF

Register 9-7. GPIOCINTF (GPIO Port C Interrupt, 0x4008 0028)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
6	**P6**	RW	0x0	Port C 6 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
5	**P5**	RW	0x0	Port C 5 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
4	**P4**	RW	0x0	Port C 4 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
3	**P3**	RW	0x0	Port C 3 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
2	**P2**	RW	0x0	Port C 2 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
1	**P1**	RW	0x0	Port C 1 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
0	**P0**	RW	0x0	Port C 0 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending

9.1.9. GPIOCINTM

Register 9-8. GPIOCINTM (GPIO Port C Interrupt Mask, 0x4008 002C)

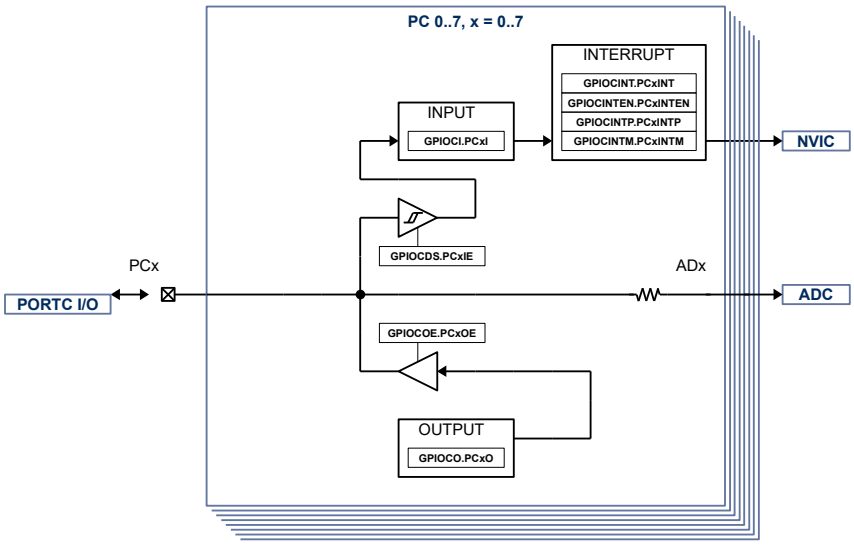
BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port C 7 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
6	**P6**	RW	0x0	Port C 6 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
5	**P5**	RW	0x0	Port C 5 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
4	**P4**	RW	0x0	Port C 4 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
3	**P3**	RW	0x0	Port C 3 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
2	**P2**	RW	0x0	Port C 2 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

BIT	NAME	ACCESS	RESET	DESCRIPTION
1	**P1**	RW	0x0	Port C 1 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
0	**P0**	RW	0x0	Port C 0 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

9.2. Details of Operation

9.2.1. Block Diagram

Figure 9-1. GPIO Port C



9.2.2. Configuration

Following blocks need to be configured for correct use of the GPIO C:

- Nested Vectored Interrupt Controller (NVIC)
- ADC

9.2.3. GPIO C Block

The GPIOC block consists of up to 8 general purpose input output (GPIO). Each GPIO has interrupt capabilities, High-Z output operation, analog ADx input to the ADC Mux. The GPIOC block IO voltage is different from the other GPIO blocks, it is supplied from VCC33.

9.2.4. Analog Input

The digital input state of GPIOC can be monitored with **GPIOCIN.Px** if **GPIOCINE.Px** is set.

Clear **GPIOCINE.Px** and **GPIOCOUTEN.Px** to disable digital input and output to allow use of analog input ADx to the ADC.

9.2.5. Output and Output Enable

When **GPIOCOUTEN.Px** is enabled, the output state is controlled by **GPIOCOUT.Px**.

When **GPIOCOUTEN. Px** is disabled, the output is in High-Z state.

9.2.6. Interrupt

The interrupt for each GPIO can be enabled with **GPIOCINTE**. The interrupt can be configured to be rising signal edge or falling signal edge using **GPIOCINTP**. The state of the interrupt can be read from **GPIOCINTF**. The individual interrupt bits can be cleared by writing to 1.

When the GPIO interrupts are enabled for the first time after device start-up, it may be in an uncertain state and generate an interrupt. To avoid this the **GPIOCINTM** mask bit need to be set before enabled interrupt bits.

To allow interrupt to be recognized by the CPU the GPIO interrupt need also be enabled in the NVIC.

10. GPIO PORT D

10.1. Register

10.1.1. Register Map

Table 10-1. GPIO Port D Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
GPIO Port D			
0x4008 0040	**GPIODOUT**	GPIO Port D output	0x0000 0000
0x4008 0044	**GPIODOUTEN**	GPIO Port D output enable	0x0000 0000
0x4008 0048	**GPIODODS**	GPIO Port D output drive strength	0x0000 0000
0x4008 004C	**GPIODPU**	GPIO Port D output weak pull up	0x0000 0000
0x4008 0050	**GPIODPD**	GPIO Port D output weak pull down	0x0000 0000
0x4008 0054	**GPIODIN**	GPIO Port D input	0x0000 0000
0x4008 0058	**Reserved**	Reserved	0x0000 0000
0x4008 005C	**GPIODPSEL**	GPIO Port D peripheral select	0x0000 0005
0x4008 0060	**GPIODINTP**	GPIO Port D interrupt polarity select	0x0000 0000
0x4008 0064	**GPIODINTE**	GPIO Port D interrupt enable select	0x0000 0000
0x4008 0068	**GPIODINTF**	GPIO Port D interrupt flag	0x0000 0000
0x4008 006C	**GPIODINTM**	GPIO Port D interrupt mask	0x0000 0000

10.1.2. GPIODO

Register 10-1. GPIODO (GPIO Port D Output, 0x4008 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D output 7 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
6	**P6**	RW	0x0	Port D output 6 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
5	**P5**	RW	0x0	Port D output 5 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
4	**P4**	RW	0x0	Port D output 4 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
3	**P3**	RW	0x0	Port D output 3 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
2	**P2**	RW	0x0	Port D output 2 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b
1	**P1**	RW	0x0	Port D output 1 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b

BIT	NAME	ACCESS	RESET	DESCRIPTION
0	**P0**	RW	0x0	Port D output 0 1b: set output high if **GPIODOUTEN.Px** = 1b 0b: set output low if **GPIODOUTEN.Px** = 1b

10.1.3. GPIODOUTEN

Register 10-2. GPIODOUTEN (GPIO Port D Output Enable, 0x4008 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D output 7 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
6	**P6**	RW	0x0	Port D output 6 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
5	**P5**	RW	0x0	Port D output 5 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
4	**P4**	RW	0x0	Port D output 4 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
3	**P3**	RW	0x0	Port D output 3 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
2	**P2**	RW	0x0	Port D output 2 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
1	**P1**	RW	0x0	Port D output 1 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state
0	**P0**	RW	0x0	Port D output 0 enable 1b: output state set by **GPIODOUT.Px** 0b: output disabled, high-impedance state

10.1.4. GPIODDS

Register 10-3. GPIODDS (GPIO Port D Output Drive Strength, 0x4008 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D output 7 drive strength select 1b: high 0b: low
6	**P6**	RW	0x0	Port D output 6 drive strength select 1b: high 0b: low
5	**P5**	RW	0x0	Port D output 5 drive strength select 1b: high 0b: low
4	**P4**	RW	0x0	Port D output 4 drive strength select 1b: high 0b: low

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**P3**	RW	0x0	Port D output 3 drive strength select 1b: high 0b: low
2	**P2**	RW	0x0	Port D output 2 drive strength select 1b: high 0b: low
1	**P1**	RW	0x0	Port D output 1 drive strength select 1b: high 0b: low
0	**P0**	RW	0x0	Port D output 0 drive strength select 1b: high 0b: low

10.1.5. GPIODPU

Register 10-4. GPIODPU (GPIO Port D Weak Pull Up, 0x4008 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
6	**P6**	RW	0x0	Port D 6 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
5	**P5**	RW	0x0	Port D 5 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
4	**P4**	RW	0x0	Port D 4 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
3	**P3**	RW	0x0	Port D 3 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
2	**P2**	RW	0x0	Port D 2 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
1	**P1**	RW	0x0	Port D 1 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
0	**P0**	RW	0x0	Port D 0 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO

10.1.6. GPIODPD

Register 10-5. GPIODPD (GPIO Port D Weak Pull Down, 0x4008 0050)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS

BIT	NAME	ACCESS	RESET	DESCRIPTION
6	**P6**	RW	0x0	Port D 6 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
5	**P5**	RW	0x0	Port D 5 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
4	**P4**	RW	0x0	Port D 4 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
3	**P3**	RW	0x0	Port D 3 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
2	**P2**	RW	0x0	Port D 2 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
1	**P1**	RW	0x0	Port D 1 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
0	**P0**	RW	0x0	Port D 0 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS

10.1.7. GPIODIN

Register 10-6. GPIODIN (GPIO Port D Input, 0x4008 0054)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 input state 1b: input high 0b: input low
6	**P6**	RW	0x0	Port D 6 input state 1b: input high 0b: input low
5	**P5**	RW	0x0	Port D 5 input state 1b: input high 0b: input low
4	**P4**	RW	0x0	Port D 4 input state 1b: input high 0b: input low
3	**P3**	RW	0x0	Port D 3 input state 1b: input high 0b: input low
2	**P2**	RW	0x0	Port D 2 input state 1b: input high 0b: input low
1	**P1**	RW	0x0	Port D 1 input state 1b: input high 0b: input low
0	**P0**	RW	0x0	Port D 0 input state 1b: input high 0b: input low

10.1.8. GPIODPSEL

Register 10-7. GPIODPSEL (GPIO Port D Peripheral Select, 0x4008 005C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:14	**P7**	RW	0x0	Port D 7 peripheral select 11b: reserved 10b: PWMD0 / DTGD0LS output or CD0 capture and compare input 01b: PWMA6 / DTGA2HS output or CA6 capture and compare input 00b: I/O mode PD7
13:12	**P6**	RW	0x0	Port D 6 peripheral select 11b: reserved 10b: PWMB1 / DTGB0HS output or CB1 capture and compare input 01b: PWMA7 / DTGA3HS output or CA7 capture and compare input 00b: I/O mode PD6
11:10	**P5**	RW	0x0	Port D 5 peripheral select 11b: reserved 10b: PWMC1 / DTGC0HS output or CC1 capture and compare input 01b: PWMA5 / DTGA1HS output or CA5 capture and compare input 00b: I/O mode PD5
9:8	**P4**	RW	0x0	Port D 4 peripheral select 11b: reserved 10b: reserved 01b: PWMD1 / DTGD0HS output or CD1 capture and compare input 00b: I/O mode PD4
7:6	**P3**	RW	0x0	Port D 3 peripheral select 11b: PWMB1 / DTGB0HS output or CB1 capture and compare input 10b: PWMA7 / DTGA3HS output or CA7 capture and compare input 01b: PWMA5 / DTGA1HS output or CA5 capture and compare input 00b: I/O mode PD3
5:4	**P2**	RW	0x0	Port D 2 peripheral select 11b: PWMB0 / DTGB0LS output or CB0 capture and compare input 10b: PWMA4 / DTGA0HS output or CA4 capture and compare input 01b: PWMA3 / DTGA3LS output or CA3 capture and compare input 00b: I/O mode PD2
3:2	**P1**	RW	0x1	Port D 1 peripheral select 11b: reserved 10b: EXTCLK input 01b: Serial wire debug SWDCLK 00b: I/O mode PD1
1:0	**P0**	RW	0x1	Port D 0 peripheral select 11b: reserved 10b: reserved 01b: Serial wire debug SWDDATA 00b: I/O mode PD0

10.1.9. GPIODINTP

Register 10-8. GPIODINTP (GPIO Port D Interrupt Polarity, 0x4008 0060)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
6	**P6**	RW	0x0	Port D 6 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

BIT	NAME	ACCESS	RESET	DESCRIPTION
5	**P5**	RW	0x0	Port D 5 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
4	**P4**	RW	0x0	Port D 4 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
3	**P3**	RW	0x0	Port D 3 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
2	**P2**	RW	0x0	Port D 2 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
1	**P1**	RW	0x0	Port D 1 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
0	**P0**	RW	0x0	Port D 0 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

10.1.10. GPIODINTE

Register 10-9. GPIODINTE (GPIO Port D Interrupt Enable, 0x4008 0064)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 interrupt enable 1b: enabled interrupt 0b: disable interrupt
6	**P6**	RW	0x0	Port D 6 interrupt enable 1b: enabled interrupt 0b: disable interrupt
5	**P5**	RW	0x0	Port D 5 interrupt enable 1b: enabled interrupt 0b: disable interrupt
4	**P4**	RW	0x0	Port D 4 interrupt enable 1b: enabled interrupt 0b: disable interrupt
3	**P3**	RW	0x0	Port D 3 interrupt enable 1b: enabled interrupt 0b: disable interrupt
2	**P2**	RW	0x0	Port D 2 interrupt enable 1b: enabled interrupt 0b: disable interrupt
1	**P1**	RW	0x0	Port D 1 interrupt enable 1b: enabled interrupt 0b: disable interrupt
0	**P0**	RW	0x0	Port D 0 interrupt enable 1b: enabled interrupt 0b: disable interrupt

10.1.11. GPIODINTF

Register 10-10. GPIODINTF (GPIO Port D Interrupt, 0x4008 0068)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
6	**P6**	RW	0x0	Port D 6 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
5	**P5**	RW	0x0	Port D 5 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
4	**P4**	RW	0x0	Port D 4 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
3	**P3**	RW	0x0	Port D 3 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
2	**P2**	RW	0x0	Port D 2 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
1	**P1**	RW	0x0	Port D 1 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
0	**P0**	RW	0x0	Port D 0 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending

10.1.12. GPIODINTM

Register 10-11. GPIODINTM (GPIO Port D Interrupt Mask, 0x4008 006C)

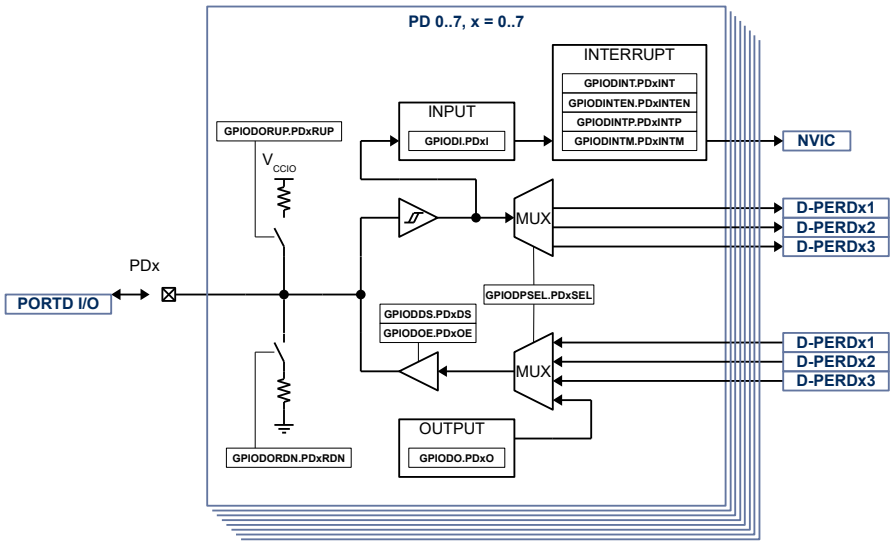
BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port D 7 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
6	**P6**	RW	0x0	Port D 6 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
5	**P5**	RW	0x0	Port D 5 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
4	**P4**	RW	0x0	Port D 4 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
3	**P3**	RW	0x0	Port D 3 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
2	**P2**	RW	0x0	Port D 2 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

BIT	NAME	ACCESS	RESET	DESCRIPTION
1	**P1**	RW	0x0	Port D 1 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
0	**P0**	RW	0x0	Port D 0 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

10.2. Details of Operation

10.2.1. Block Diagram

Figure 10-1. GPIO Port D



10.2.2. Configuration

Following blocks need to be configured for correct use of the GPIO D:

- Nested Vectored Interrupt Controller (NVIC)
- Gate Driver
- Timer A, PWMA, DTGA
- Timer B, PWMB, DTGB
- Timer C, PWMC, DTGC
- Timer D, PWMD, DTGD
- CCS
- SWD Debugger

10.2.3. GPIO D Block

The GPIO D block consists of up to 8 general purpose input output (GPIO). Each GPIO has interrupt capabilities, weak pull-up or pull-down, programmable output drive strength, High-Z output operation. Some of the GPIO can be configured as PWM output, or capture and compare input.

10.2.4. Input

The input state of GPIOD can be monitored with **GPIODIN.Px**. The input state can be monitored regardless of the peripheral select setting **GPIODPSEL**.

10.2.5. Output and Output Enable

When **GPIODOUTEN.Px** is enabled, the output state is controlled by **GPIODOUT.Px**.

When **GPIODOE.PDxOE** is disabled, the output is in High-Z state.

10.2.6. Output Drive Strength

The output drive strength can be adjusted using **GPIODDS** to meet application needs. Set **GPIODDS.Px** to enable high current drive strength, reset to enable low current drive strength.

10.2.7. Weak Pull Up and Pull Down

Independent from the output settings, weak pull up can be enabled with **GPIODPU** and weak pull down can be enabled with **GPIODPD**.

NOTE:

GPIODPU.Px or **GPIODPD.Px** should never be enabled at the same time for a single GPIO. If switching from weak pull-up to weak pull-down is required, disable weak pull-up first before enable weak pull-down and vice versa.

10.2.8. Peripheral Select

Each GPIO is connected to up to 4 digital peripherals, selectable with **GPIODPSEL**. When a different function than IO is selected the input state can still be read with **GPIODIN** and the pull-up and pull-down is still controllable.

10.2.9. Interrupt

The interrupt for each GPIO can be enabled with **GPIODINTE**. The interrupt can be configured to be rising signal edge or falling signal edge using **GPIODINTP**. The state of the interrupt can be read from **GPIODINTF**. The individual interrupt bits can be cleared by writing to 1.

When the GPIO interrupts are enabled for the first time after device start-up, it may be in an uncertain state and generate an interrupt. To avoid this the **GPIODINTM** mask bit need to be set before enabled interrupt bits.

To allow interrupt to be recognized by the CPU the GPIO interrupt need also be enabled in the NVIC.

11. GPIO PORT E

11.1. Register

11.1.1. Register Map

Table 11-1. GPIO Port E Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
GPIO Port E			
0x4009 0000	**GPIOEOUT**	GPIO Port E output	0x0000 0000
0x4009 0004	**GPIOEOUTEN**	GPIO Port E output enable	0x0000 0000
0x4009 0008	**GPIOEODS**	GPIO Port E output drive strength	0x0000 0000
0x4009 000C	**GPIOEPU**	GPIO Port E output weak pull up	0x0000 0000
0x4009 0010	**GPIOEPD**	GPIO Port E output weak pull down	0x0000 0000
0x4009 0014	**GPIOEIN**	GPIO Port E input	0x0000 0000
0x4009 0018	**Reserved**	Reserved	0x0000 0000
0x4009 001C	**GPIOEPSEL**	GPIO Port E peripheral select	0x0000 0000
0x4009 0020	**GPIOEINTP**	GPIO Port E interrupt polarity select	0x0000 0000
0x4009 0024	**GPIOEINTE**	GPIO Port E interrupt enable select	0x0000 0000
0x4009 0028	**GPIOEINTF**	GPIO Port E interrupt flag	0x0000 0000
0x4009 002C	**GPIOEINTM**	GPIO Port E interrupt mask	0x0000 0000

11.1.2. GPIOEOUT

Register 11-1. GPIOEOUT (GPIO Port E Output, 0x4009 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E output 7 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
6	**P6**	RW	0x0	Port E output 6 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
5	**P5**	RW	0x0	Port E output 5 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
4	**P4**	RW	0x0	Port E output 4 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
3	**P3**	RW	0x0	Port E output 3 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
2	**P2**	RW	0x0	Port E output 2 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b
1	**P1**	RW	0x0	Port E output 1 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b

BIT	NAME	ACCESS	RESET	DESCRIPTION
0	**P0**	RW	0x0	Port E output 0 1b: set output high if **GPIOEOUTEN.Px** = 1b 0b: set output low if **GPIOEOUTEN.Px** = 1b

11.1.3. GPIOEOUTEN

Register 11-2. GPIOEOUTEN (GPIO Port E Output Enable, 0x4009 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0	Reserved
7	**P7**	RW	0x0	Port E output 7 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
6	**P6**	RW	0x0	Port E output 6 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
5	**P5**	RW	0x0	Port E output 5 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
4	**P4**	RW	0x0	Port E output 4 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
3	**P3**	RW	0x0	Port E output 3 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
2	**P2**	RW	0x0	Port E output 2 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
1	**P1**	RW	0x0	Port E output 1 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state
0	**P0**	RW	0x0	Port E output 0 enable 1b: output state set by **GPIOEOUT.Px** 0b: output disabled, high-impedance state

11.1.4. GPIOEDS

Register 11-3. GPIOEDS (GPIO Port E Output Drive Strength, 0x4009 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E output 7 drive strength select 1b: high 0b: low
6	**P6**	RW	0x0	Port E output 6 drive strength select 1b: high 0b: low
5	**P5**	RW	0x0	Port E output 5 drive strength select 1b: high 0b: low
4	**P4**	RW	0x0	Port E output 4 drive strength select 1b: high 0b: low

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**P3**	RW	0x0	Port E output 3 drive strength select 1b: high 0b: low
2	**P2**	RW	0x0	Port E output 2 drive strength select 1b: high 0b: low
1	**P1**	RW	0x0	Port E output 1 drive strength select 1b: high 0b: low
0	**P0**	RW	0x0	Port E output 0 drive strength select 1b: high 0b: low

11.1.5. GPIOEPU

Register 11-4. GPIOEPU (GPIO Port E Weak Pull Up, 0x4009 000C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
6	**P6**	RW	0x0	Port E 6 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
5	**P5**	RW	0x0	Port E 5 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
4	**P4**	RW	0x0	Port E 4 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
3	**P3**	RW	0x0	Port E 3 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
2	**P2**	RW	0x0	Port E 2 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
1	**P1**	RW	0x0	Port E 1 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO
0	**P0**	RW	0x0	Port E 0 weak pull up select 1b: enable weak pull-up to VCCIO 0b: disable weak pull-up to VCCIO

11.1.6. GPIOEPD

Register 11-5. GPIOEPD (GPIO Port E Weak Pull Down, 0x4009 0010)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS

BIT	NAME	ACCESS	RESET	DESCRIPTION
6	**P6**	RW	0x0	Port E 6 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
5	**P5**	RW	0x0	Port E 5 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
4	**P4**	RW	0x0	Port E 4 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
3	**P3**	RW	0x0	Port E 3 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
2	**P2**	RW	0x0	Port E 2 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
1	**P1**	RW	0x0	Port E 1 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS
0	**P0**	RW	0x0	Port E 0 weak pull down select 1b: enable weak pull-down to VSS 0b: disable weak pull-down to VSS

11.1.7. GPIOEIN

Register 11-6. GPIOEIN (GPIO Port E Input, 0x4009 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 input state 1b: input high 0b: input low
6	**P6**	RW	0x0	Port E 6 input state 1b: input high 0b: input low
5	**P5**	RW	0x0	Port E 5 input state 1b: input high 0b: input low
4	**P4**	RW	0x0	Port E 4 input state 1b: input high 0b: input low
3	**P3**	RW	0x0	Port E 3 input state 1b: input high 0b: input low
2	**P2**	RW	0x0	Port E 2 input state 1b: input high 0b: input low
1	**P1**	RW	0x0	Port E 1 input state 1b: input high 0b: input low
0	**P0**	RW	0x0	Port E 0 input state 1b: input high 0b: input low

11.1.8. GPIOEPSEL

Register 11-7. GPIOEPSEL (GPIO Port E Peripheral Select, 0x4009 001C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:14	**P7**	RW	0x0	Port E 7 peripheral select 11b: reserved 10b: reserved 01b: reserved 00b: I/O mode PE7
13:12	**P6**	RW	0x0	Port E 6 peripheral select 11b: reserved 10b: reserved 01b: reserved 00b: I/O mode PE6
11:10	**P5**	RW	0x0	Port E 5 peripheral select 11b: reserved 10b: I2C clock I2CSDA 01b: SPI chip select 2 SPICS2 00b: I/O mode PE5
9:8	**P4**	RW	0x0	Port E 4 peripheral select 11b: reserved 10b: I2C clock I2CSCL 01b: SPI chip select 1 SPICS1 00b: I/O mode PE4
7:6	**P3**	RW	0x0	Port E 3 peripheral select 11b: reserved 10b: Device Reset input nRESET1 01b: SPI chip select 0 SPICS0 00b: I/O mode PE3
5:4	**P2**	RW	0x0	Port E 2 peripheral select 11b: reserved 10b: UART Receive UARTRX 01b: SPI Master in Slave out SPIMISO 00b: I/O mode PE2
3:2	**P1**	RW	0x0	Port E 1 peripheral select 11b: reserved 10b: UART Transmit UARTTX 01b: SPI Master out Slave in SPIMOSI 00b: I/O mode PE1
1:0	**P0**	RW	0x0	Port E 0 peripheral select 11b: reserved 10b: reserved 01b: SPI Clock SPICLK 00b: I/O mode PE0

11.1.9. GPIOEINTP

Register 11-8. GPIOEINTP (GPIO Port E Interrupt Polarity, 0x4009 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
6	**P6**	RW	0x0	Port E 6 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

BIT	NAME	ACCESS	RESET	DESCRIPTION
5	**P5**	RW	0x0	Port E 5 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
4	**P4**	RW	0x0	Port E 4 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
3	**P3**	RW	0x0	Port E 3 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
2	**P2**	RW	0x0	Port E 2 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
1	**P1**	RW	0x0	Port E 1 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition
0	**P0**	RW	0x0	Port E 0 interrupt polarity select 1b: Rising edge, low to high transition 0b: Falling edge, high to low transition

11.1.10. GPIOEINTE

Register 11-9. GPIOEINTE (GPIO Port E Interrupt Enable, 0x4009 0024)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 interrupt enable 1b: enabled interrupt 0b: disable interrupt
6	**P6**	RW	0x0	Port E 6 interrupt enable 1b: enabled interrupt 0b: disable interrupt
5	**P5**	RW	0x0	Port E 5 interrupt enable 1b: enabled interrupt 0b: disable interrupt
4	**P4**	RW	0x0	Port E 4 interrupt enable 1b: enabled interrupt 0b: disable interrupt
3	**P3**	RW	0x0	Port E 3 interrupt enable 1b: enabled interrupt 0b: disable interrupt
2	**P2**	RW	0x0	Port E 2 interrupt enable 1b: enabled interrupt 0b: disable interrupt
1	**P1**	RW	0x0	Port E 1 interrupt enable 1b: enabled interrupt 0b: disable interrupt
0	**P0**	RW	0x0	Port E 0 interrupt enable 1b: enabled interrupt 0b: disable interrupt

11.1.11. GPIOEINTF

Register 11-10. GPIOEINTF (GPIO Port E Interrupt Flag, 0x4009 0028)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
6	**P6**	RW	0x0	Port E 6 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
5	**P5**	RW	0x0	Port E 5 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
4	**P4**	RW	0x0	Port E 4 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
3	**P3**	RW	0x0	Port E 3 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
2	**P2**	RW	0x0	Port E 2 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
1	**P1**	RW	0x0	Port E 1 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending
0	**P0**	RW	0x0	Port E 0 interrupt 1b: interrupt pending, clear with write to 1b 0b: no interrupt pending

11.1.12. GPIOEINTM

Register 11-11. GPIOEINTM (GPIO Port E Interrupt Mask, 0x4009 002C)

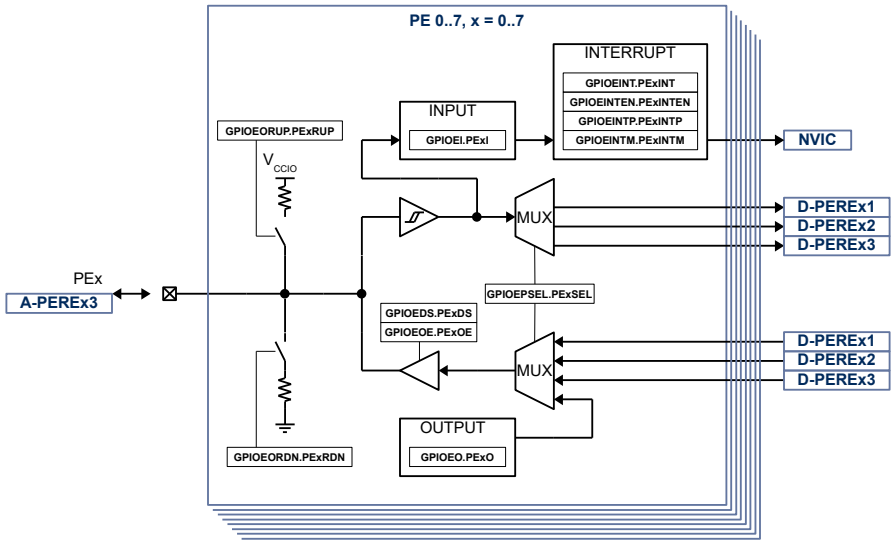
BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**P7**	RW	0x0	Port E 7 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
6	**P6**	RW	0x0	Port E 6 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
5	**P5**	RW	0x0	Port E 5 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
4	**P4**	RW	0x0	Port E 4 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
3	**P3**	RW	0x0	Port E 3 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
2	**P2**	RW	0x0	Port E 2 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

BIT	NAME	ACCESS	RESET	DESCRIPTION
1	**P1**	RW	0x0	Port E 1 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask
0	**P0**	RW	0x0	Port E 0 interrupt mask 1b: enable interrupt mask 0b: disable interrupt mask

11.2. Details of Operation

11.2.1. Block Diagram

Figure 11-1. GPIO Port E



11.2.2. Configuration

Following blocks need to be configured for correct use of the GPIO E:

- Nested Vectored Interrupt Controller (NVIC)
- SPI
- I2C
- UART

11.2.3. GPIO E Block

The GPIO E block consists of up to 8 general purpose input output (GPIO). Each GPIO has interrupt capabilities, weak pull-up or pull-down, programmable output drive strength, High-Z output operation.

11.2.4. Input

The input state of GPIOE can be monitored with **GPIOEIN.Px**. The input state can be monitored regardless of the peripheral select setting **GPIOEPSEL**.

11.2.5. Output and Output Enable

When **GPIOEOUTEN.Px**is enabled, the output state is controlled by **GPIOEOUT.Px**.

When **GPIOEOUTEN.Px** is disabled, the output is in High-Z state.

11.2.6. Output Drive Strength

The output drive strength can be adjusted using **GPIOEDS** to meet application needs. Set **GPIOEDS.Px** to enable high current drive strength, reset to enable low current drive strength.

11.2.7. Weak Pull Up and Pull Down

Independent from the output settings, weak pull up can be enabled with **GPIOEPU** and weak pull down can be enabled with **GPIOPD**.

NOTE:

GPIOEPU.Pxor **GPIOEPD.Px** should never be enabled at the same time for a single GPIO. If switching from weak pull-up to weak pull-down is required, disable weak pull-up first before enablle weak pull-down and vice versa.

11.2.8. Peripheral Select

Each GPIO is connected to up to 4 digital peripherals, selectable with **GPIOEPSEL**. When a different function than IO is selected the input state can still be read with **GPIOEIN** and the pull-up and pull-down is still controllable.

11.2.9. Interrupt

The interrupt for each GPIO can be enabled with **GPIOEINTE**. The interrupt can be configured to be rising signal edge or falling signal edge using **GPIOEINTP**. The state of the interrupt can be read from **GPIOEINTF**. The individual interrupt bits can be cleared by writing to 1.

When the GPIO interrupts are enabled for the first time after device start-up, it may be in an uncertain state and generate an interrupt. To avoid this the **GPIOEINTM** mask bit need to be set before enabled interrupt bits.

To allow interrupt to be recognized by the CPU the GPIO interrupt need also be enabled in the NVIC.

12. TIMER A

12.1. Register

12.1.1. Register Map

Table 12-1. Timer A Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Timer A			
0x400D 0000	**TACTL**	Timer A control	0x0000 0000
0x400D 0004	**TAPRD**	Timer A period	0x0000 0000
0x400D 0008	**TACTR**	Timer A counter	0x0000 0000
Timer A PWMA Capture and Compare			
0x400D 0040	**TACCTRL0**	Timer A capture and compare 0 control	0x0000 0000
0x400D 0044	**TACTR0**	Timer A counter 0	0x0000 0000
0x400D 0048	**TACCTRL1**	Timer A capture and compare 1 control	0x0000 0000
0x400D 004C	**TACTR1**	Timer A counter 1	0x0000 0000
0x400D 0050	**TACCTRL2**	Timer A capture and compare 2 control	0x0000 0000
0x400D 0054	**TACTR2**	Timer A counter 2	0x0000 0000
0x400D 0058	**TACCTRL3**	Timer A capture and compare 3 control	0x0000 0000
0x400D 005C	**TACTR3**	Timer A counter 3	0x0000 0000
0x400D 0060	**TACCTRL4**	Timer A capture and compare 4 control	0x0000 0000
0x400D 0064	**TACTR4**	Timer A counter 4	0x0000 0000
0x400D 0068	**TACCTRL5**	Timer A capture and compare 5 control	0x0000 0000
0x400D 006C	**TACTR5**	Timer A counter 5	0x0000 0000
0x400D 0070	**TACCTRL6**	Timer A capture and compare 6 control	0x0000 0000
0x400D 0074	**TACTR6**	Timer A counter 6	0x0000 0000
0x400D 0078	**TACCTRL7**	Timer A capture and compare 7 control	0x0000 0000
0x400D 007C	**TACTR7**	Timer A counter 7	0x0000 0000
Timer A Dead Time Generator			
0x400D 00A0	**DTGA0CTL**	Timer A dead time generator 0 control	0x0000 0080
0x400D 00A4	**DTGA0LED**	Timer A dead time generator 0 leading edge delay	0x0000 0000
0x400D 00A8	**DTGA0TED**	Timer A dead time generator 0 trailing edge delay	0x0000 0000
0x400D 00B0	**DTGA1CTL**	Timer A dead time generator 1 control	0x0000 0080
0x400D 00B4	**DTGA1LED**	Timer A dead time generator 1 leading edge delay	0x0000 0000
0x400D 00B8	**DTGA1TED**	Timer A dead time generator 1 trailing edge delay	0x0000 0000
0x400D 00C0	**DTGA2CTL**	Timer A dead time generator 2 control	0x0000 0080
0x400D 00C4	**DTGA2LED**	Timer A dead time generator 2 leading edge delay	0x0000 0000
0x400D 00C8	**DTGA2TED**	Timer A dead time generator 2 trailing edge delay	0x0000 0000
0x400D 00D0	**DTGA3CTL**	Timer A dead time generator 3 control	0x0000 0080
0x400D 00D4	**DTGA3LED**	Timer A dead time generator 3 leading edge delay	0x0000 0000
0x400D 00D8	**DTGA3TED**	Timer A dead time generator 3 trailing edge delay	0x0000 0000

12.1.2. TACTL

Register 12-1. TACTL (Timer A Control, 0x400D 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	RO	0x0	Reserved
13	**DTGCLK**	RW	0x0	DTGAx clock select 1b: DTGAx use clock after **TACTL.CLKDIV** 0b: DTGAx use clock selected by **TACTL.CLK**
12	**SYNC**	RW	0x0	Timer B synchronization 1b: Synchronize Timer A, enable SYNC_IN 0b: Do not synchronize Timer A, disabled SYNC_IN
11:10	**MODE**	RW	0x0	Timer A Mode 11b: reserved 10b: up / down 01b: up 00b: disabled
9	**CLK**	RW	0x0	Timer A clock input source 1b: ACLK 0b: HCLK
8:6	**CLKDIV**	RW	0x0	Timer A input clock divider 111b: / 128 110b: / 64 101b: /32 100b: /16 011b: /8 010b: /4 001b: /2 000b: /1
5	**INTEN**	RW	0x0	Timer A interrupt enable 1b: enable Timer A interrupt 0b: disable Timer A interrupt
4	**INT**	RW	0x0	Timer A interrupt 1b: interrupt, clear by write 1b 0b: no interrupt
3	**SS**	RW	0x0	Timer A single shot 1b: single shot mode 0b: continuous timer mode
2	**CLR**	RW	0x0	Timer A clear 1b: Clear Timer A, hold Timer A in reset and set SYNC_OUT 0b: Do not clear timer and clear SYNC_OUT
1	**Reserved**	RO	0x0	Reserved
0	**PRDL**	RW	0x0	Timer A **TAPRD** update 1b: Latch new **TAPRD** value when timer A counting down, **TACTR** value = 0x1 and **TACTL.MODE** = 10b 0b: Latch new **TAPRD** value when timer A counting up and **TACTR** value = TAPRD – 0x1.

12.1.3. TAPRD

Register 12-2. TAPRD (Timer A Period, 0x400D 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0	Reserved
15:0	**PERIOD**	RW	0x0	Timer A period value

12.1.4. TACTR

Register 12-3. TACTR (Timer A Counter, 0x400D 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CTR**	RO	0x0	Current Timer A counter value

12.1.5. TACCCTRL0

Register 12-4. TACCCTRL0 (Timer A PWMA0 Capture and Compare Control, 0x400D 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA0 input 0b: Compare mode PWMA0 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA0 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.6. TACCCTR0

Register 12-5. TACCCTR0 (Timer A PWMA0 Capture and Compare Counter, 0x400D 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA0 compare mode or counter value for PWMA0 capture mode

12.1.7. TACCCTRL1

Register 12-6. TACC1CTRL1 (Timer A PWMA1 Capture and Compare Control, 0x400D 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA1 input 0b: Compare mode PWMA1 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected

BIT	NAME	ACCESS	RESET	DESCRIPTION
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA1 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.8. TACCCTR1

Register 12-7. TACCCTR1 (Timer A PWMA1 Capture and Compare Counter, 0x400D 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA1 compare mode or counter value for PWMA1 capture mode

12.1.9. TACCCTRL2

Register 12-8. TACCCTRL2 (Timer A PWMA2 Capture and Compare Control, 0x400D 0050)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA2 input 0b: Compare mode PWMA2 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA2 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.10. TACC2CTR2

Register 12-9. TACCCTR2 (Timer A PWMA2 Capture and Compare Counter, 0x400D 0054)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA2 compare mode or counter value for PWMA2 capture mode

12.1.11. TACCCTRL3

Register 12-10. TACCCTRL3 (Timer A PWMA3 Capture and Compare Control, 0x400D 0058)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA3 input 0b: Compare mode PWMA3 output

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA3 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.12. TACCCTR3

Register 12-11. TACCCTR3 (Timer A PWMA3 Capture and Compare Counter, 0x400D 005C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA3 compare mode or counter value for PWMA3 capture mode

12.1.13. TACCCTRL4

Register 12-12. TACCCTRL4 (Timer A PWMA4 Capture and Compare Control, 0x400D 0060)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA4 input 0b: Compare mode PWMA4 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA4 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.14. TACCCTR4

Register 12-13. TACCCTR4 (Timer A PWMA4 Capture and Compare Counter, 0x400D 0064)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA4 compare mode or counter value for PWMA4 capture mode

12.1.15. TACCCTRL5

Register 12-14. TACCCTRL5 (Timer A PWMA5 Capture and Compare Control, 0x400D 0068)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA5 input 0b: Compare mode PWMA5 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA5 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.16. TACCCTR5

Register 12-15. TACCCTR5 (Timer A PWMA5 Capture and Compare Counter, 0x400D 006C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA5 compare mode or counter value for PWMA5 capture mode

12.1.17. TACCCTRL6

Register 12-16. TACCCTRL6 (Timer A PWMA6 Capture and Compare Control, 0x400D 0070)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA6 input 0b: Compare mode PWMA6 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA6 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.18. TACCCTR7

Register 12-17. TACCCTR7 (Timer A PWMA6 Capture and Compare Counter, 0x400D 0074)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA6 compare mode or counter value for PWMA6 capture mode

12.1.19. TACCCTRL7

Register 12-18. TACCCTRL7 (Timer A PWMA7 Capture and Compare Control, 0x400D 0078)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMA7 input 0b: Compare mode PWMA7 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMA7 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

12.1.20. TACCCTR7

Register 12-19. TACCCTR7 (Timer A PWMA7 Capture and Compare Counter, 0x400D 007C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMA7 compare mode or counter value for PWMA7 capture mode

12.1.21. DTGA0CTL

Register 12-20. DTGA0CTL (Timer A Dead Time Generator 0 Control, 0x400D 00A0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGA0 bypass active, DTGA0LS = PWMA0, DTGA0HS = PWMA4 0b: DTGA0 bypass inactive, dead time inserted to DTGA0LS and DTGA0HS, DTGA0LS = $\overline{\text{PWMA4}}$, DTGA0HS = PWMA4
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGA0HS high time is same as PWMA4 hightime and is shifted by **DTGA0LED** 0b: DTGA0HS high time is reduced by **DTGA0LED**
5	**INVHS**	RW	0x0	Invert DTGA0HS output signal 1b: invert DTGA0HS 0b: do not invert DTGA0HS

BIT	NAME	ACCESS	RESET	DESCRIPTION
4	**INVLS**	RW	0x0	Invert DTGA0LS output signal 1b: invert DTGA0LS 0b: do not invert DTGA0LS
3:0	**Reserved**	RO	0x0	Reserved

12.1.22. DTGA0LED

Register 12-21. DTGA0LED (Timer A Dead Time Generator 0 Leading Edge Delay, 0x400D 00A4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGA0 leading edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.23. DTGA0TED

Register 12-22. DTGA0TED (Timer A Dead Time Generator 0 Trailing Edge Delay, 0x400D 00A8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGA0 trailing edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.24. DTGA1CTL

Register 12-23. DTGA1CTL (Timer A Dead Time Generator 1 Control, 0x400D 00B0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGA1 bypass active, DTGA1LS = PWMA1, DTGA1HS = PWMA5 0b: DTGA1 bypass inactive, dead time inserted to DTGA1LS and DTGA1HS, DTGA1LS = $\overline{\text{PWMA5}}$, DTGA1HS = PWMA5
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGA1HS high time is same as PWMA5 hightime and is shifted by **DTGA1LED** 0b: DTGA1HS high time is reduced by **DTGA1LED**
5	**INVHS**	RW	0x0	Invert DTGA1HS output signal 1b: invert DTGA1HS 0b: do not invert DTGA1HS
4	**INVLS**	RW	0x0	Invert DTGA1LS output signal 1b: invert DTGA1LS 0b: do not invert DTGA1LS
3:0	**Reserved**	RO	0x0	Reserved

12.1.25. DTGA1LED

Register 12-24. DTGA1LED (Timer A Dead Time Generator 1 Leading Edge Delay, 0x400D 00B4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGA1 leading edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.26. DTGA1TED

Register 12-25. DTGA1TED (Timer A Dead Time Generator 1 Trailing Edge Delay, 0x400D 00B8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGA1 trailing edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.27. DTGA2CTL

Register 12-26. DTGA2CTL (Timer A Dead Time Generator 2 Control, 0x400D 00C0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGA2 bypass active, DTGA2LS = PWMA2, DTGA2HS = PWMA6 0b: DTGA2 bypass inactive, dead time inserted to DTGA2LS and DTGA2HS, DTGA2LS = $\overline{\text{PWMA6}}$, DTGA2HS = PWMA6
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGA2HS high time is same as PWMA6 hightime and is shifted by **DTGA2LED** 0b: DTGA2HS high time is reduced by **DTGA2LED**
5	**INVHS**	RW	0x0	Invert DTGA2HS output signal 1b: invert DTGA2HS 0b: do not invert DTGA2HS
4	**INVLS**	RW	0x0	Invert DTGA2LS output signal 1b: invert DTGA2LS 0b: do not invert DTGA2LS
3:0	**Reserved**	RO	0x0	Reserved

12.1.28. DTGA2LED

Register 12-27. DTGA2LED (Timer A Dead Time Generator 2 Leading Edge Delay, 0x400D 00C4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGA2 leading edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.29. DTGA2TED

Register 12-28. DTGA2TED (Timer A Dead Time Generator 2 Trailing Edge Delay, 0x400D 00C8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGA2 trailing edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.30. DTGA3CTL

Register 12-29. DTGA3CTL (Timer A Dead Time Generator 3 Control, 0x400D 00D0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGA3 bypass active, DTGA3LS = PWMA3, DTGA3HS = PWMA7 0b: DTGA3 bypass inactive, dead time inserted to DTGA3LS and DTGA3HS, DTGA3LS = $\overline{\text{PWMA7}}$, DTGA1HS = PWMA7
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGA3HS high time is same as PWMA7 high time and is shifted by **DTGA3LED** 0b: DTGA3HS high time is reduced by **DTGA3LED**
5	**INVHS**	RW	0x0	Invert DTGA3HS output signal 1b: invert DTGA3HS 0b: do not invert DTGA3HS
4	**INVLS**	RW	0x0	Invert DTGA3LS output signal 1b: invert DTGA3LS 0b: do not invert DTGA3LS
3:0	**Reserved**	RO	0x0	Reserved

12.1.31. DTGA3LED

Register 12-30. DTGA3LED (Timer A Dead Time Generator 3 Leading Edge Delay, 0x400D 00D4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGA3 leading edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.1.32. DTGA3TED

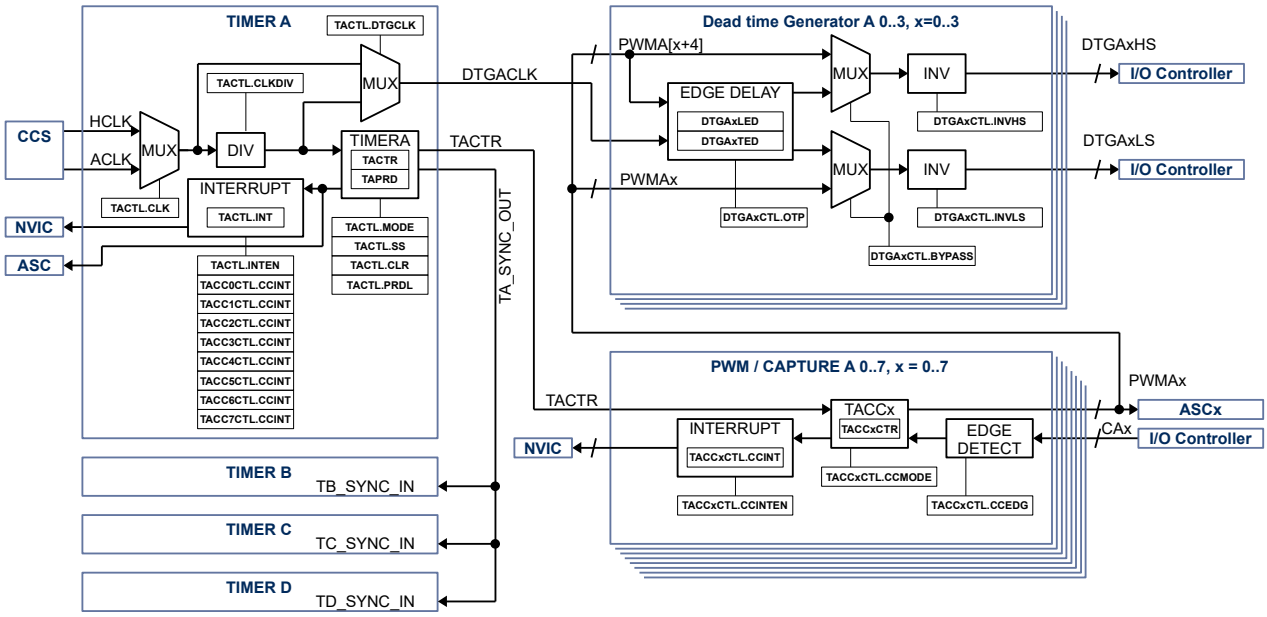
Register 12-31. DTGA3TED (Timer A Dead Time Generator 3 Trailing Edge Delay, 0x400D 00D8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGA3 trailing edge dead time in clock cycles defined by **TACTL.DTGCLK**

12.2. Details of Operation

12.2.1. Block Diagram

Figure 12-1. Timer A



12.2.2. Configuration

Following blocks need to be configured for correct use of the Timer A:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- I/O Controller
- Gate Driver
- Auto sequencing controller (ASC)
- Timer B
- Timer C
- Timer D

12.2.3. Timer A Block

The timer A block consist of a 16-bit timer with up mode or up/down mode with 8 PWM/capture units and 4 dead-time generator units.

12.2.4. Timer

Once enabled the timer counts up to the Timer A period value **TAPRD**. The **TAPRD** register can be written to while the timer is running, the new **TAPRD** value will be latched when the counter reaches old **TAPRD** value in

up mode. In up/down mode there is the option to latch the new **TAPRD** value when counter counts back to zero. **TACTL.PRDL** configures when the timer will be updated with the new **TAPRD** value in up/down mode.

The current timer value is accessible with the timer A counter value register **TACTR**.

12.2.5. Register update

The **TAPRD**, **TACCxCTR** register can be written to while the timer is running, the new **TAPRD, TACCxCTR** value will be latched when the counter reaches old **TAPRD** value in up mode. In up/down mode there is the option to latch the new **TAPRD, TACCxCTR** values when counter counts back to zero. **TACTL.PRDL** configures when the timer will be updated with the new **TAPRD**, **TACCxCTR** value in up/down mode.

12.2.6. Timer Modes

The timer supports 3 modes of operation: disabled, up and up/down using **TACTL.MODE**.

By default, the timer mode is disabled. When the timer is disabled, the timer counter does not increment or decrement. If the timer is disabled when previously in up or up/down mode, then the timer counter stops where it is. If the timer is re-enabled by putting it back into up or up/down modes, then the counter continues from the point at which it was disabled. To reset the current counter value **TACTR** to zero use **TACTL.CLR**.

In up mode, the timer starts counting from 0 up to the value of **TAPRD**. When the timer counter reaches the value of **TAPRD**, then the timer counter is reset to a value of 0.This mode is typically used for timed events or edge-aligned PWM output.

In up/down mode, the timer starts counting from 0 up to the value of **TAPRD**, and then back down to a value of 0. This timer mode is typically used for center-aligned PWM output. It can also be used for timed events, and will allow a longer timer range due to the fact it counts up and down

12.2.7. Single Shot Mode

The timer can be configured to run either once or continuously.

When the timer is configured in single shot mode using **TACTL.SS** the timer will only count to **TAPRD** value and stops in up mode. In up/down mode the timer will count to **TAPRD** and back to zero only once.

To start the timer in single-shot mode, **TACTL.SS** must be set. The timer will start when **TACTL.CLR** is set. To re-start a single-shot timer, **TACTL.CLR** must be reset, and then set again.

12.2.8. Input Clock And Pre-Scaler

The timer can be configured to use HCLK or ACLK using **TACTL.CLK**. The input clock for the can be divided further down from /1 to /128 using the **TACTL.CLKDIV**.

12.2.9. Timer Synchronization

The Timer A, B, C, D in the system have the ability to have synchronization between them. Each timer has a synchronization in signal (SYNC_IN) and the synchronization out signal (SYNC_OUT).

Timer A can be synchronized with Timer B, C, and D with timer A as master.

The timer asserts the SYNC_OUT pulse when the **TACTL.CLR** bit is set and de-asserts the SYNC_OUT pulse when the **TACTL.CLR** bit is cleared.

Each timer B, C, or D that need to be synchronized as slave with master timer A need to set the **TxCTL.SYNC** bit. If this is bit is not set, then the sync_in signal is ignored and the timer operates independently.

When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is asserted, the timer clears the timer counter. The timer counter is also cleared anytime the **TxCTL.CLR** bit is set to a 1. When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is de-asserted and the timer mode is either up or up/down, then the timer will start counting. The timer will not start counting when the mode is set to up or up/down unless the SYNC_IN signal is de-asserted when **TxCTL.SYNC** is set.

NOTE:

In order for this feature to work correctly, all timers that are synchronized must be set to the same mode (up or up/down), with the same timer pre-scaler, timer clock input and timer period.

To enable synchronized timers, the following steps should be followed:

1. All slave timers B, C, or D are configured with the selected timer input clock, timer pre-scaler, timer period and set the **TxCTL.SYNC** bit. The timer should still be set to disabled at this point.
2. The master timer A is configured with the same timer input clock, timer pre-scaler, timer period and sets the **TxCTL.CLR** bit. This should clear all timer counters of the master and slave timers.
3. The slave timers set the timer mode to the desired state (either up or up/down).
4. The master timer sets the timer mode to either up or up/down and clears the **TACTL.CLR** bit. This should start the master and all slave timers simultaneously based on the selected timer clock input.
5. Once configured as above, all timers can be disabled by the master setting **TACTL.CLR** signal, to assert the SYNC_OUT signal. The timers can be re-enabled by clearing the **TACTL.CLR** bit, which de-asserts the SYNC_OUT signal.

12.2.10. PWM/Compare Units

Timer A supports up to 8 PWM/Capture units PWMA0 to PWMA7. Each PWM/Compare unit can be configured independently in PWM mode or capture mode using **TACCxCTL.CCMODE**.

12.2.10.1. PWM Mode

The PWM mode is enabled with setting **TACCxCTRL.CCMODE** to 0.

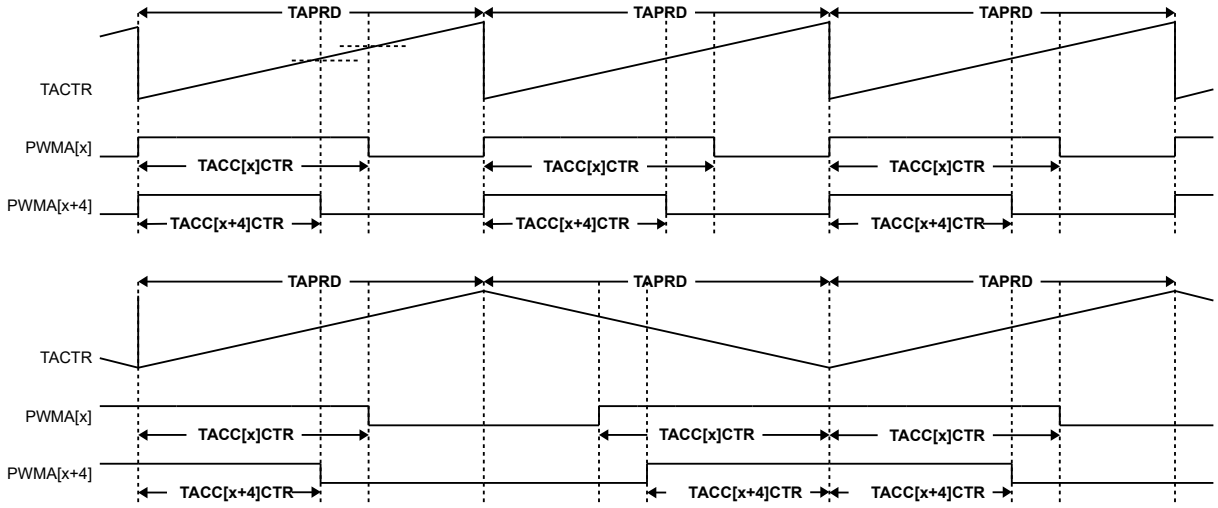
The timer configuration allows either edge-aligned (timer in up mode) or center-aligned (timer in up/down mode) modes of PWM operation.

In both edge-aligned and center-aligned modes of operation, the timer block outputs a PWM waveform that starts out high at a **TACTR** value of 0 and then transitions to low when **TACTR** counts up to **TACCxCTR** compare value.

To configure a duty cycle of 0%, the **TACCxCTR** should be set to 0; to configure a duty cycle of 100%, the **TACCxCTR** value should be set to a value greater than or equal to **TAPRD**.

The polarity of the timer PWM outputs are not configurable. Adjustments to the polarity of the PWM outputs may be adjusted in the Dead-Time Generator (DTG) unit connected to the timer peripheral for each output independently.

Figure 12-2. PWMA[x] and PWMA[x+4] Example Using Timer A Up Mode and Up/Down Mode

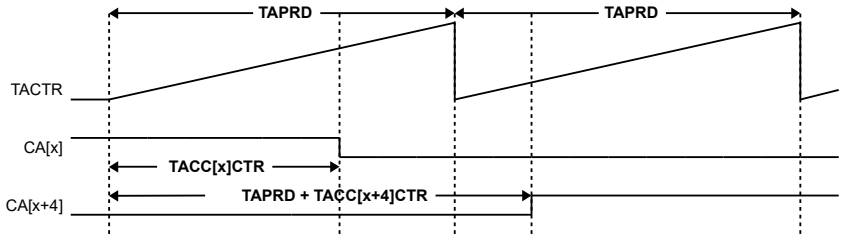


12.2.10.2. Capture Mode

The Capture mode is enabled with setting **TACCxCTRL.CCMODE** to 1. The trip condition for capture mode can be configured using **TACCxCTRL.CCEDGE**, high-to-low signal edge transition, low-to-high signal edge transition or both.

When trip condition is detected the actual **TACTR** value is copied into **TACCxCTR**.

Figure 12-3. CA[x] and CA[x+4] Capture Example



12.2.11. Timer and PWM/Capture Interrupt

The timer may generate interrupt based on the base timer wrap, or when a capture and compare event occurs.

In the base timer both up and up/down timer modes allow an interrupt to be generated when the count reaches 0. Each time the count reaches zero, the **TACTL.INT** interrupt flag is set. If the interrupt is enabled using the **TACTL.INTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TACTL.INT** interrupt flag bit.

In the capture and compare PWM units, each time a compare threshold is reached or each time a capture event is detected the **TACCxCTL.CCINT** bit will be set for that particular timer unit. If the interrupt is enabled via the **TACCxCTRL.CCINTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TACCxCTRL.CCINT** interrupt flag bit.

The timer IRQ signal will be asserted if any of the timer interrupt flags **TACTL.INT** or **TACCxCTRL.CCINT** are set. The Timer IRQ signal will be de-asserted when all of the timer interrupt flags are cleared.

12.2.12. Dead-Time Generator

The dead-time generator can be configured to introduce dead-time for a complementary PWM output. The Timer A block supports up to 4 dead time generators.

12.2.12.1. Dead Time Input Clock Selection

The clock source for the DTGAx can be selected using **TACTL.DTGCLK**.

Clear **TACTL.DTGCLK** to 0 to use clock source selected by **TACTL.CLK** directly to use higher resolution for dead time insertion.

Set **TACTL.DTGCLK** to 1 to use divided clock source selected by **TACTL.CLK** and **TACTL.CLKDIV** divider to use the same dead time resolution as Timer A.

12.2.12.2. Dead Time Range

The resolution for leading edge and trailing edge dead time is 12bits. Leading and trailing edge can be set independently for each DTG using **DTGAxLED** and **DTGAxTED**.

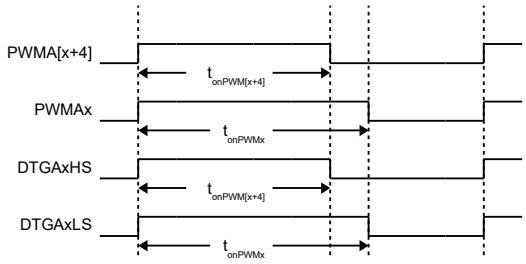
12.2.12.3. Bypass Mode

Set **DTGAxCTL.BYPASS** to 0 to enable dead time insertion.

Set **DTGAxCTL.BYPASS** to 1 to enable bypass mode, no deadtime is inserted, PWMA[x+4] is routed to DTGAxHS and PWMAx is routed to DTGAxLS.

The DTGAxHS and DTGAxLS signals can be inverted in bypass mode.

Figure 12-4. DTGAx Bypass Example



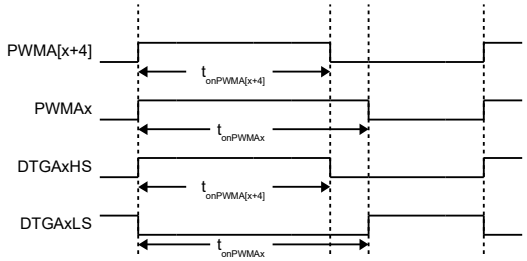
12.2.12.4. Inverting PWM Signal

The DTG output signals DTGAxHS and DTGAxLS can be inverted independently.

Set **DTGAxCTL.INVHS** to invert DTGAxHS signal.

Set **DTGAxCTL.INVLS** to invert DTGAxLS signal.

Figure 12-5. DTGAx Bypass and Inverting LS Example



12.2.12.5. Dead Time Insertion

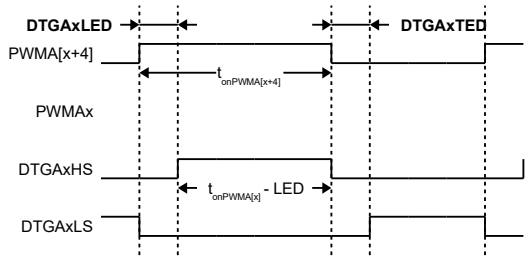
Set **DTGAxCTL.BYPASS** to 0 to enable dead time insertion. In dead time insertion mode only PWM[x+4] signal is used to generate DTGAxHS and DTGAxLS. PWMA[x] signal is ignored and can be used for other purposes.

Set **DTGAxLED** for desired leading-edge and **DTGAxTED** for desired trailing edge in clock-cycles defined by **TACTL.DTGCLK** clock source

NOTE:

In dead time insertion mode the DTGAxLS signal is automatically inverted compared to PWMA[x+4] signal. Set **DTGAxCTL.INVLS** to 0, if this is desired behavior.

Figure 12-6. DTGAx LED and TED Example



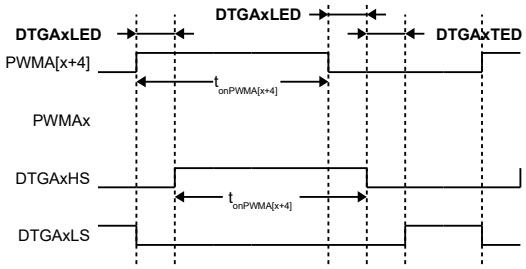
12.2.12.6. Dead Time Insertion with On Time Preservation

Set **DTGAxCTL.OTP** to 1 to enable on time preservation. In this mode the DTGAxHS is same as PWM[x+4] on time.

NOTE:

In dead time insertion mode the DTGAxLS signal is automatically inverted compared to PWMA[x+4] signal. Set **DTGAxCTL.INVLS** to 0, if this is desired behavior.

Figure 12-7. DTGAx LED and TED with On Time Preservation Example



12.2.13. PWM Output and Capture Input Pin Selection

Each of the DTGAxHS, DTGAxLS outputs, and CAx inputs can be routed to different I/Os, allowing great flexibility in pin assignments.

In capture mode only one I/O should be enabled as input to the capture. If more than one pin input is enabled, the capture might not work properly.

Note:

Not all pins are available pending package option, consult data sheet for available pins and signals.

Table 12-2. Timer A Signal to Pin Mapping

PWM	CAPTURE	DEADTIME	PINS
PWMA0	CA0	DTGA0LS	PA0
PWMA1	CA1	DTGA1LS	PA1
PWMA2	CA2	DTGA2LS	PA2
PWMA3	CA3	DTGA3LS	PA3, PD2
PWMA4	CA4	DTGA0HS	PA3, PA6, PD2
PWMA5	CA5	DTGA1HS	PA4, PA7, PD3, PD5
PWMA6	CA6	DTGA2HS	PA5, PD7
PWMA7	CA7	DTGA3HS	PA7, PD6, PD3

13. TIMER B

13.1. Register

13.1.1. Register Map

Table 13-1. Timer B Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Timer B			
0x400E 0000	**TBCTL**	Timer B control	0x0000 0000
0x400E 0004	**TBPRD**	Timer B period	0x0000 0000
0x400E 0008	**TBCTR**	Timer B counter	0x0000 0000
Timer B PWMB Capture and Compare			
0x400E 0040	**TBCCTRL0**	Timer B capture and compare 0 control	0x0000 0000
0x400E 0044	**TBCTR0**	Timer B counter 0	0x0000 0000
0x400E 0048	**TBCCTRL1**	Timer B capture and compare 1 control	0x0000 0000
0x400E 004C	**TBCTR1**	Timer B counter 1	0x0000 0000
0x400E 0050	**TBCCTRL2**	Timer B capture and compare 2 control	0x0000 0000
0x400E 0054	**TBCTR2**	Timer B counter 2	0x0000 0000
0x400E 0058	**TBCCTRL3**	Timer B capture and compare 3 control	0x0000 0000
0x400E 005C	**TBCTR3**	Timer B counter 3	0x0000 0000
Timer B Dead Time Generator			
0x400E 00A0	**DTGB0CTL**	Timer B dead time generator 0 control	0x0000 0080
0x400E 00A4	**DTGB0LED**	Timer B dead time generator 0 leading edge delay	0x0000 0000
0x400E 00A8	**DTGB0TED**	Timer B dead time generator 0 trailing edge delay	0x0000 0000

13.1.2. TBCTL

Register 13-1. TBCTL (Timer B Control, 0x400E 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	RO	0x0	Reserved
13	**DTGCLK**	RW	0x0	DTGBx clock select 1b: DTGBx use clock after **TBCTL.CLKDIV** 0b: DTGBx use clock selected by **TBCTL.CLK**
12	**SYNC**	RW	0x0	Timer B synchronization 1b: Synchronize Timer B, enable SYNC_IN 0b: Do not synchronize Timer B, disabled SYNC_IN
11:10	**MODE**	RW	0x0	Timer B Mode 11b: reserved 10b: up / down 01b: up 00b: disabled
9	**CLK**	RW	0x0	Timer B clock input source 1b: ACLK 0b: HCLK

BIT	NAME	ACCESS	RESET	DESCRIPTION
8:6	**CLKDIV**	RW	0x0	Timer B input clock divider 111b: / 128 110b: / 64 101b: /32 100b: /16 011b: /8 010b: /4 001b: /2 000b: /1
5	**INTEN**	RW	0x0	Timer B interrupt enable 1b: enable Timer B interrupt 0b: disable Timer B interrupt
4	**INT**	RW1C	0x0	Timer B interrupt 1b: interrupt, clear by write 1b 0b: no interrupt
3	**SS**	RW	0x0	Timer B single shot 1b: single shot mode 0b: continuous timer mode
2	**CLR**	RW	0x0	Timer B clear 1b: Clear Timer B, hold Timer B in reset and set SYNC_OUT 0b: Do not clear Timer and clear SYNC_OUT
1	**Reserved**	RO	0x0	Reserved
0	**PRDL**	RW	0x0	Timer B **TBPRD** update 1b: Latch new **TBPRD** value when Timer B counting down, **TBCTR** value = 0x1 and **TBCTL.MODE** = 10b 0b: Latch new **TBPRD** value when Timer B counting up and **TBCTR** value = TBPRD – 0x1.

13.1.3. TBPRD

Register 13-2. TBPRD (Timer B Period, 0x400E 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0	Reserved
15:0	**PERIOD**	RW	0x0	Timer B period value

13.1.4. TBCTR

Register 13-3. TBCTR (Timer B Counter, 0x400E 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CTR**	RO	0x0	Current Timer B counter value

13.1.5. TBCC0CTRL

Register 13-4. TBCC0CTRL (Timer B PWMB0 Capture and Compare Control, 0x400E 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMB0 input 0b: Compare mode PWMB0 output

BIT	NAME	ACCESS	RESET	DESCRIPTION
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMB0 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

13.1.6. TBCC0CTR

Register 13-5. TBCC0CTR (Timer B PWMB0 Capture and Compare Counter, 0x400E 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMB0 compare mode or counter value for PWMB0 capture mode

13.1.7. TBCC1CTRL

Register 13-6. TBCC1CTRL (Timer B PWMB1 Capture and Compare Control, 0x400E 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMB1 input 0b: Compare mode PWMB1 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMB1 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

13.1.8. TBCC1CTR

Register 13-7. TBCC1CTR (Timer B PWMB1 Capture and Compare Counter, 0x400E 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMB1 compare mode or counter value for PWMB1 capture mode

13.1.9. TBCC2CTRL

Register 13-8. TBCC2CTRL (Timer B PWMB2 Capture and Compare Control, 0x400E 0050)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMB2 input 0b: Compare mode PWMB2 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMB2 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

13.1.10. TBCC2CTR

Register 13-9. TBCC2CTR (Timer B PWMB2 Capture and Compare Counter, 0x400E 0054)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMB2 compare mode or counter value for PWMB2 capture mode

13.1.11. TBCC3CTRL

Register 13-10. TBCC3CTRL (Timer B PWMB3 Capture and Compare Control, 0x400E 0058)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMB3 input 0b: Compare mode PWMB3 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMB3 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

13.1.12. TBCC3CTR

Register 13-11. TBCC3CTR (Timer B PWMB3 Capture and Compare Counter, 0x400E 005C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMB3 compare mode or counter value for PWMB3 capture mode

13.1.13. DTGB0CTL

Register 13-12. DTGB0CTL (Timer B Dead Time Generator 0 Control, 0x400E 00A0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGB0 bypass active, DTGB0LS = PWMB0, DTGB0HS = PWMB1 0b: DTGB0 bypass inactive, dead time inserted to DTGB0LS and DTGB0HS, DTGB0LS = $\overline{PWMB1}$, DTGB0HS = PWMB1
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGB0HS high time is same as PWMB1 high time and is shifted by **DTGB0LED** 0b: DTGB0HS high time is reduced by **DTGB0LED**
5	**INVHS**	RW	0x0	Invert DTGB0HS output signal 1b: invert DTGB0HS 0b: do not invert DTGB0HS
4	**INVLS**	RW	0x0	Invert DTGB0LS output signal 1b: invert DTGB0LS 0b: do not invert DTGB0LS
3:0	**Reserved**	RO	0x0	Reserved

13.1.14. DTGB0LED

Register 13-13. DTGB0LED (Timer B Dead Time Generator 0 Leading Edge Delay, 0x400E 00A4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGB0 leading edge dead time in clock cycles defined by **TBCTL.DTGCLK**

13.1.15. DTGB0TED

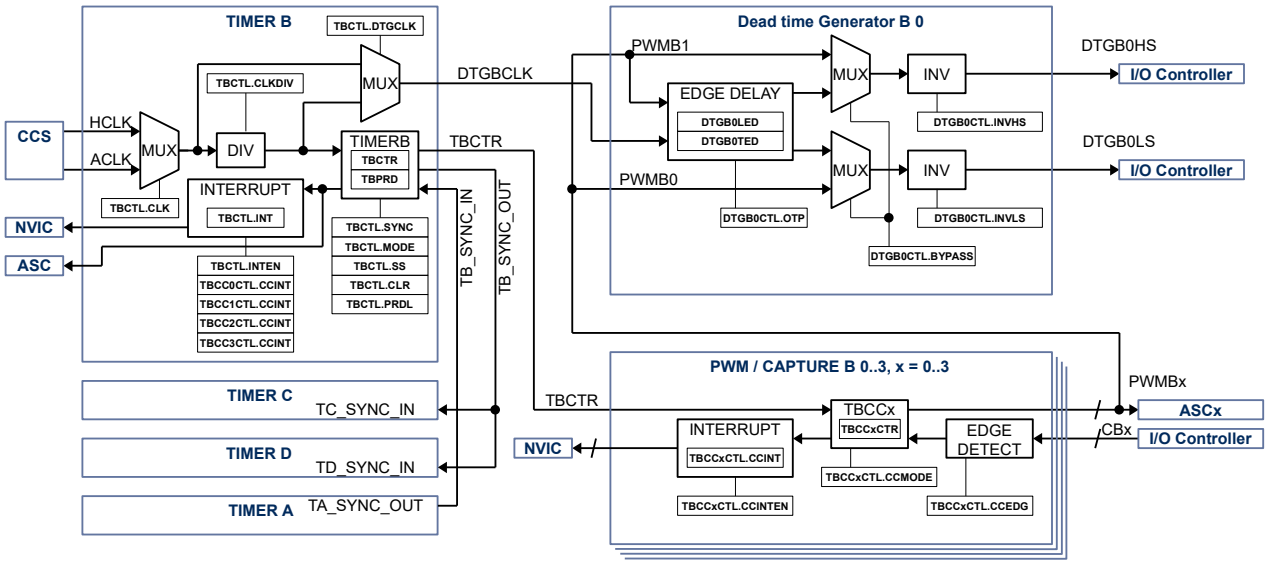
Register 13-14. DTGB0TED (Timer B Dead Time Generator 0 Trailing Edge Delay, 0x400E 00A8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGB0 trailing edge dead time in clock cycles defined by **TBCTL.DTGCLK**

13.2. Details of Operation

13.2.1. Block Diagram

Figure 13-1. Timer B



13.2.2. Configuration

Following blocks need to be configured for correct use of the Timer B:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller
- Gate Driver
- Auto sequencing controller (ASC)
- Timer A
- Timer C
- Timer D
-

13.2.3. Timer B Block

The timer B block consist of a 16-bit timer with up mode or up/down mode with 4 PWM/capture units and 1 dead-time generator unit.

13.2.4. Timer

Once enabled the timer counts up to the Timer B period value **TBPRD**. The **TBPRD** register can be written to while the timer is running, the new **TBPRD** value will be latched when the counter reaches old **TBPRD** value in up mode. In up/down mode there is the option to latch the new **TBPRD** value when counter counts back to zero. **TBCTL.PRDL** configures when the timer will be updated with the new **TBPRD** value in up/down mode.

The current timer value is accessible with the timer B counter value register **TBCTR**.

13.2.5. Register update

The **TBPRD**, **TBCCxCTR** register can be written to while the timer is running, the new **TBPRD**, **TBCCxCTR** value will be latched when the counter reaches old **TBPRD** value in up mode. In up/down mode there is the option to latch the new **TBPRD, TBCCxCTR** values when counter counts back to zero. **TBCTL.PRDL** configures when the timer will be updated with the new **TBPRD**, **TBCCxCTR** value in up/down mode.

13.2.6. Timer Modes

The timer supports 3 modes of operation: disabled, up and up/down using **TBCTL.MODE**.

By default, the timer mode is disabled. When the timer is disabled, the timer counter does not increment or decrement. If the timer is disabled when previously in up or up/down mode, then the timer counter stops where it is. If the timer is re-enabled by putting it back into up or up/down modes, then the counter continues from the point at which it was disabled. To reset the current counter value **TBCTR** to zero use **TBCTL.CLR**.

In up mode, the timer starts counting from 0 up to the value of **TBPRD**. When the timer counter reaches the value of **TBPRD**, then the timer counter is reset to a value of 0.This mode is typically used for timed events or edge-aligned PWM output.

In up/down mode, the timer starts counting from 0 up to the value of **TBPRD**, and then back down to a value of 0. This timer mode is typically used for center-aligned PWM output. It can also be used for timed events, and will allow a longer timer range due to the fact it counts up and down.

13.2.7. Single Shot Mode

The timer can be configured to run either once or continuously.

When the timer is configured in single shot mode using **TBCTL.SS** the timer will only count to **TBPRD** value and stops in up mode. In up/down mode the timer will count to **TBPRD** and back to zero only once.

To start the timer in single-shot mode, **TBCTL.SS** must be set. The timer will start when **TBCTL.CLR** is set. To re-start a single-shot timer, **TBCTL.CLR** must be reset, and then set again.

13.2.8. Input Clock And Pre-Scaler

The timer can be configured to use HCLK or ACLK using **TBCTL.CLK**. The input clock for the can be divided further down from /1 to /128 using the **TBCTL.CLKDIV**.

13.2.9. Timer Synchronization

The Timer A, B, C, D in the system have the ability to have synchronization between them. Each timer has a synchronization in signal (SYNC_IN) and the synchronization out signal (SYNC_OUT).

Timer B can be synchronized with Timer C, and D with timer B as master. Timer B can be synchronized with Timer A as slave.

The timer asserts the SYNC_OUT pulse when the **TBCTL.CLR** bit is set and de-asserts the SYNC_OUT pulse when the **TBCTL.CLR** bit is cleared.

Each timer C, or D that need to be synchronized as slave with master timer B need to set the **TxCTL.SYNC** bit. If this is bit is not set, then the sync_in signal is ignored and the timer operates independently.

When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is asserted, the timer clears the timer counter. The timer counter is also cleared anytime the **TxCTL.CLR** bit is set to a 1. When the **TxCTL.SYNC** bit is set and the

SYNC_IN signal is de-asserted and the timer mode is either up or up/down, then the timer will start counting. The timer will not start counting when the mode is set to up or up/down unless the SYNC_IN signal is de-asserted when **TxCTL.SYNC** is set.

NOTE:

In order for this feature to work correctly, all timers that are synchronized must be set to the same mode (up or up/down), with the same timer pre-scaler, timer clock input and timer period.

To enable synchronized timers, the following steps should be followed:

1. All slave timers B, C, or D are configured with the selected timer input clock, timer pre-scaler, timer period and set the **TxCTL.SYNC** bit. The timer should still be set to disabled at this point.
2. The master timer A, or B is configured with the same timer input clock, timer pre-scaler, timer period and sets the **TxCTL.CLR** bit. This should clear all timer counters of the master and slave timers.
3. The slave timers set the timer mode to the desired state (either up or up/down).
4. The master timer sets the timer mode to either up or up/down and clears the **TxCTL.CLR** bit. This should start the master and all slave timers simultaneously based on the selected timer clock input.
5. Once configured as above, all timers can be disabled by the master setting **TxCTL.CLR** signal, to assert the SYNC_OUT signal. The timers can be re-enabled by clearing the **TxCTL.CLR** bit, which de-asserts the SYNC_OUT signal.
6.

13.2.10. PWM/Compare Units

Timer B supports up to 4 PWM/Capture units PWMB0 to PWMB3. Each PWM/Compare unit can be configured independently in PWM mode or capture mode using **TBCCxCTRL.CCMODE**.

13.2.10.1. PWM Mode

The PWM mode is enabled with setting **TBCCxCTRL.CCMODE** to 0.

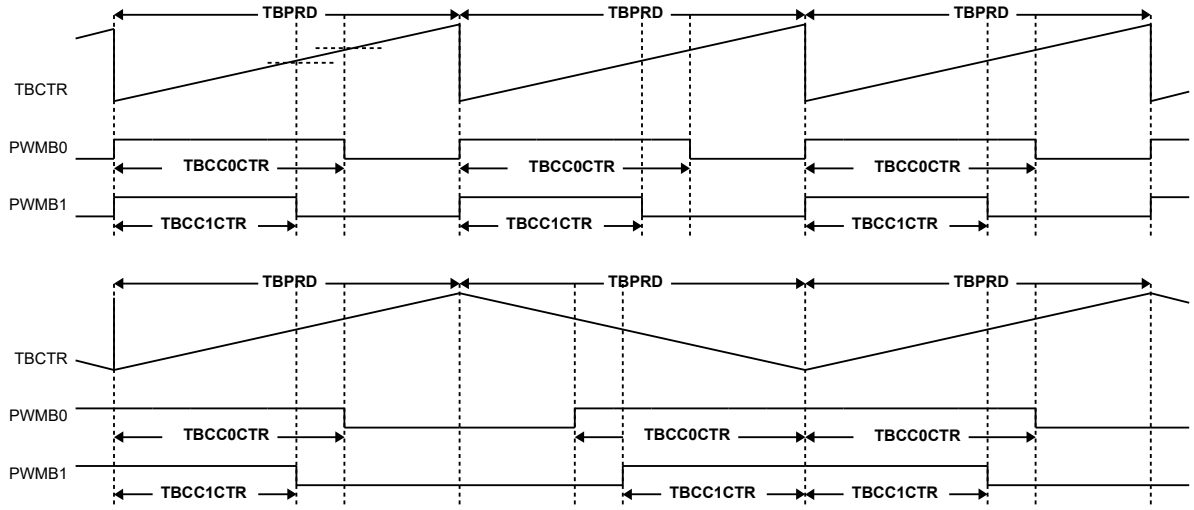
The timer configuration allows either edge-aligned (timer in up mode) or center-aligned (timer in up/down mode) modes of PWM operation.

In both edge-aligned and center-aligned modes of operation, the timer block outputs a PWM waveform that starts out high at a **TBCTR** value of 0 and then transitions to low when **TBCTR** counts up to **TBCCxCTR** compare value.

To configure a duty cycle of 0%, the **TBCCxCTR** should be set to 0; to configure a duty cycle of 100%, the **TBCCxCTR** value should be set to a value greater than or equal to **TBPRD**.

The polarity of the timer PWM outputs are not configurable. Adjustments to the polarity of the PWM outputs may be adjusted in the Dead-Time Generator (DTG) unit connected to the timer peripheral for each output independently.

Figure 13-2. PWMB0 and PWMB1 Example Using Timer B Up Mode and Up/Down Mode

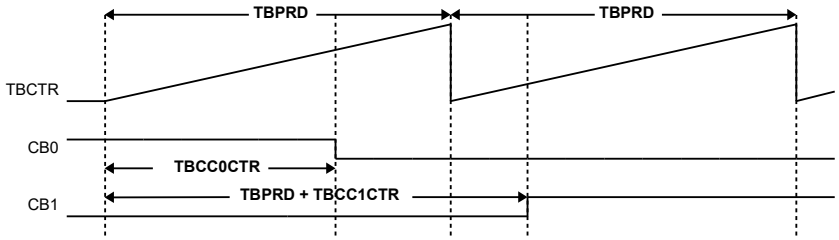


13.2.10.2. Capture Mode

The Capture mode is enabled with setting **TBCCxCTRL.CCMODE** to 1. The trip condition for capture mode can be configured using **TBCCxCTRL.CCEDGE**, high-to-low signal edge transition, low-to-high signal edge transition or both.

When trip condition is detected the actual **TBCTR** value is copied into **TBCCxCTR**.

Figure 13-3. CB0 and CB1 Capture Example



13.2.11. Timer and PWM/Capture Interrupt

The timer may generate interrupt based on the base timer wrap, or when a capture and compare event occurs.

In the base timer both up and up/down timer modes allow an interrupt to be generated when the count reaches 0. Each time the count reaches zero, the **TBCTL.INT** interrupt flag is set. If the interrupt is enabled using the **TBCTL.INTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TBCTL.INT** interrupt flag bit.

In the capture and compare PWM units, each time a compare threshold is reached or each time a capture event is detected the **TBCCxCTRL.CCINT** bit will be set for that particular timer unit. If the interrupt is enabled via the **TBCCxCTRL.CCINTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TBCCxCTRL.CCINT** interrupt flag bit.

The timer IRQ signal will be asserted if any of the timer interrupt flags **TBCTL.INT** or **TBCCxCTRL.CCINT** are set. The Timer IRQ signal will be de-asserted when all of the timer interrupt flags are cleared.

13.2.12. Dead-Time Generator

The dead-time generator can be configured to introduce dead-time for a complementary PWM output. The Timer B block supports up to 1 dead time generator.

13.2.12.1. Dead Time Input Clock Selection

The clock source for the DTGB0 can be selected using **TBCTL.DTGCLK**.

Clear **TBCTL.DTGCLK** to 0 to use clock source selected by **TBCTL.CLK** directly to use higher resolution for dead time insertion.

Set **TBCTL.DTGCLK** to 1 to use divided clock source selected by **TBCTL.CLK** and **TBCTL.CLKDIV** divider to use the same dead time resolution as Timer B.

13.2.12.2. Dead Time Range

The resolution for leading edge and trailing edge dead time is 12bits. Leading and trailing edge can be set independently using **DTGB0LED** and **DTGB0TED**.

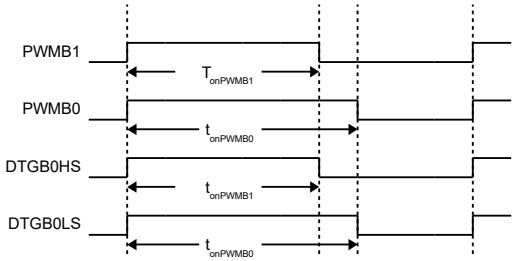
13.2.12.3. Bypass Mode

Set **DTGB0CTL.BYPASS** to 0 to enable dead time insertion.

Set **DTGB0CTL.BYPASS** to 1 to enable bypass mode, no dead time is inserted, PWMB1 is routed to DTGB0HS and PWMB0 is routed to DTGB0LS.

The DTGB0HS and DTGB0LS signals can be inverted in bypass mode.

Figure 13-4. DTGB0 Bypass Example



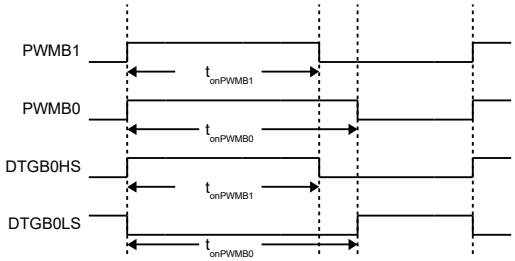
13.2.12.4. Inverting PWM Signal

The DTG output signals DTGB0HS and DTGB0LS can be inverted independently.

Set **DTGB0CTL.INVHS** to invert DTGB0HS signal.

Set **DTGB0CTL.INVLS** to invert DTGB0LS signal.

Figure 13-5. DTGB0 Bypass and Inverting LS Example



13.2.12.5. Dead Time Insertion

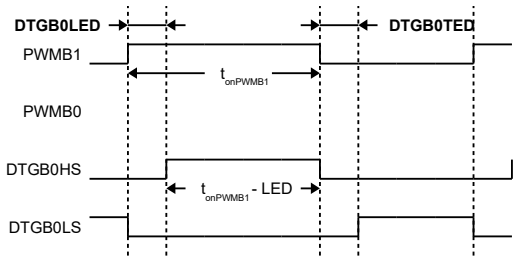
Set **DTGB0CTL.BYPASS** to 0 to enable dead time insertion. In dead time insertion mode only PWMB1 signal is used to generate DTGB0HS and DTGB0LS. PWMB0 signal is ignored and can be used for other purposes.

Set **DTGB0LED** for desired leading-edge and **DTGB0TED** for desired trailing edge in clock-cycles defined by **TBCTL.DTGCLK** clock source

NOTE:

In dead time insertion mode the DTGB0LS signal is automatically inverted compared to PWMB1 signal. Set **DTGB0CTL.INVLS** to 0, if this is desired behavior.

Figure 13-6. DTGB0 LED and TED Example



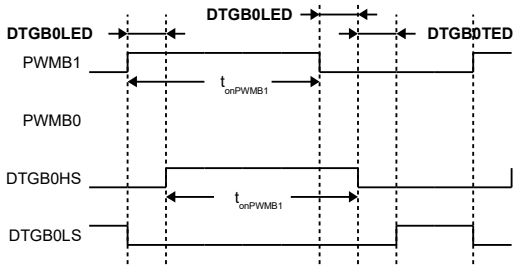
13.2.12.6. Dead Time Insertion with On Time Preservation

Set **DTGB0CTL.OTP** to 1 to enable on time preservation. In this mode the DTGB0HS is same as PWMB1 on time.

NOTE:

In dead time insertion mode the DTGB0LS signal is automatically inverted compared to PWMB1 signal. Set **DTGB0CTL.INVLS** to 0, if this is desired behavior.

Figure 13-7. DTGB0 LED and TED with On Time Preservation Example



13.2.13. PWM Output and Capture Input Pin Selection

Each of the DTGB0HS, DTGB0LS outputs, and CBx inputs can be routed to different I/Os, allowing great flexibility in pin assignments.

In capture mode only one I/O should be enabled as input to the capture. If more than one pin input is enabled, the capture might not work properly.

Note:

Not all pins are available pending package option, consult data sheet for available pins and signals.

Table 13-2. Timer B Signal to Pin Mapping

PWM	CAPTURE	DEADTIME	PINS
PWMB0	CB0	DTGB0LS	PA3, PA6, PD2
PWMB1	CB1	DTGB0HS	PD3, PD6
PWMB2	CB2	N/A	N/A
PWMB3	CB3	N/A	N/A

14. TIMER C

14.1. Register

14.1.1. Register Map

Table 14-1. Timer C Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Timer C			
0x400F 0000	**TCCTL**	Timer C control	0x0000 0000
0x400F 0004	**TCPRD**	Timer C period	0x0000 0000
0x400F 0008	**TCCTR**	Timer C counter	0x0000 0000
Timer C PWMC Capture and Compare			
0x400F 0040	**TCCCTRL0**	Timer C capture and compare 0 control	0x0000 0000
0x400F 0044	**TCCTR0**	Timer C counter 0	0x0000 0000
0x400F 0048	**TCCCTRL1**	Timer C capture and compare 1 control	0x0000 0000
0x400F 004C	**TCCTR1**	Timer C counter 1	0x0000 0000
Timer C Dead Time Generator			
0x400F 00A0	**DTGC0CTL**	Timer C dead time generator 0 control	0x0000 0080
0x400F 00A4	**DTGC0LED**	Timer C dead time generator 0 leading edge delay	0x0000 0000
0x400F 00A8	**DTGC0TED**	Timer C dead time generator 0 trailing edge delay	0x0000 0000

14.1.2. TCCTL

Register 14-1. TCCTL (Timer C Control, 0x400F 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	RO	0x0	Reserved
13	**DTGCLK**	RW	0x0	DTGCx clock select 1b: DTGCx use clock after **TCCTL.CLKDIV** 0b: DTGCx use clock selected by **TCCTL.CLK**
12	**SYNC**	RW	0x0	Timer C synchronization 1b: Synchronize Timer C, enable SYNC_IN 0b: Do not synchronize Timer C, disabled SYNC_IN
11:10	**MODE**	RW	0x0	Timer C Mode 11b: reserved 10b: up / down 01b: up 00b: disabled
9	**CLK**	RW	0x0	Timer C clock input source 1b: ACLK 0b: HCLK
8:6	**CLKDIV**	RW	0x0	Timer C input clock divider 111b: /128 110b: /64 101b: /32 100b: /16 011b: /8 010b: /4 001b: /2 000b: /1

BIT	NAME	ACCESS	RESET	DESCRIPTION
5	**INTEN**	RW	0x0	Timer C interrupt enable 1b: enable Timer C interrupt 0b: disable Timer C interrupt
4	**INT**	RW1C	0x0	Timer C interrupt 1b: interrupt, clear by write 1b 0b: no interrupt
3	**SS**	RW	0x0	Timer C single shot 1b: single shot mode 0b: continuous timer mode
2	**CLR**	RW	0x0	Timer C clear 1b: Clear Timer C, hold Timer C in reset and set SYNC_OUT 0b: Do not clear Timer and clear SYNC_OUT
1	**Reserved**	RO	0x0	Reserved
0	**PRDL**	RW	0x0	Timer C **TCPRD** update 1b: Latch new **TCPRD** value when Timer C counting down, **TCCTR** value = 0x1 and **TCCTL.MODE** = 10b 0b: Latch new **TCPRD** value when Timer C counting up and **TCCTR** value = **TCPRD** – 0x1.

14.1.3. TCPRD

Register 14-2. TCPRD (Timer C Period, 0x400F 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0	Reserved
15:0	**PERIOD**	RW	0x0	Timer C period value

14.1.4. TCCTR

Register 14-3. TCCTR (Timer C Counter, 0x400F 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CTR**	RW	0x0	Current Timer C counter value

14.1.5. TCCC0CTRL

Register 14-4. TCCC0CTL (Timer C PWMC0 Capture and Compare Control, 0x400F 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMC0 input 0b: Compare mode PWMC0 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected

BIT	NAME	ACCESS	RESET	DESCRIPTION
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMC0 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

14.1.6. TCCC0CTR

Register 14-5. TCCC0CTR (Timer C PWMC0 Capture and Compare Counter, 0x400F 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMC0 compare mode or counter value for PWMC0 capture mode

14.1.7. TCCC1CTRL

Register 14-6. TCCC1CTL (Timer C PWMC1 Capture and Compare Control, 0x400F 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMC1 input 0b: Compare mode PWMC1 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMC1 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

14.1.8. TCCC1CTR

Register 14-7. TCCC1CTR (Timer C PWMC1 Capture and Compare Counter, 0x400F 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMC1 compare mode or counter value for PWMC1 capture mode

14.1.9. DTGC0CTL

Register 14-8. DTGC0CTL (Timer C Dead Time Generator 0 Control, 0x400F 00A0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGC0 bypass active, DTGC0LS = PWMC0, DTGC0HS = PWMC1 0b: DTGC0 bypass inactive, dead time inserted to DTGC0LS and DTGC0HS, DTGC0LS = $\overline{\text{PWMC1}}$, DTGC0HS = PWMC1
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGC0HS high time is same as PWMC1 high time is shifted by **DTGC0LED** 0b: DTGC0HS high time is reduced by **DTGC0LED**
5	**INVHS**	RW	0x0	Invert DTGC0HS output signal 1b: invert DTGC0HS 0b: do not invert DTGC0HS
4	**INVLS**	RW	0x0	Invert DTGC0LS output signal 1b: invert DTGC0LS 0b: do not invert DTGC0LS
3:0	**Reserved**	RO	0x0	Reserved

14.1.10. DTGC0LED

Register 14-9. DTGC0LED (Timer C Dead Time Generator 0 Leading Edge Delay, 0x400F 00A4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGC0 leading edge dead time in clock cycles defined by **TCCTL.DTGCLK**

14.1.11. DTGC0TED

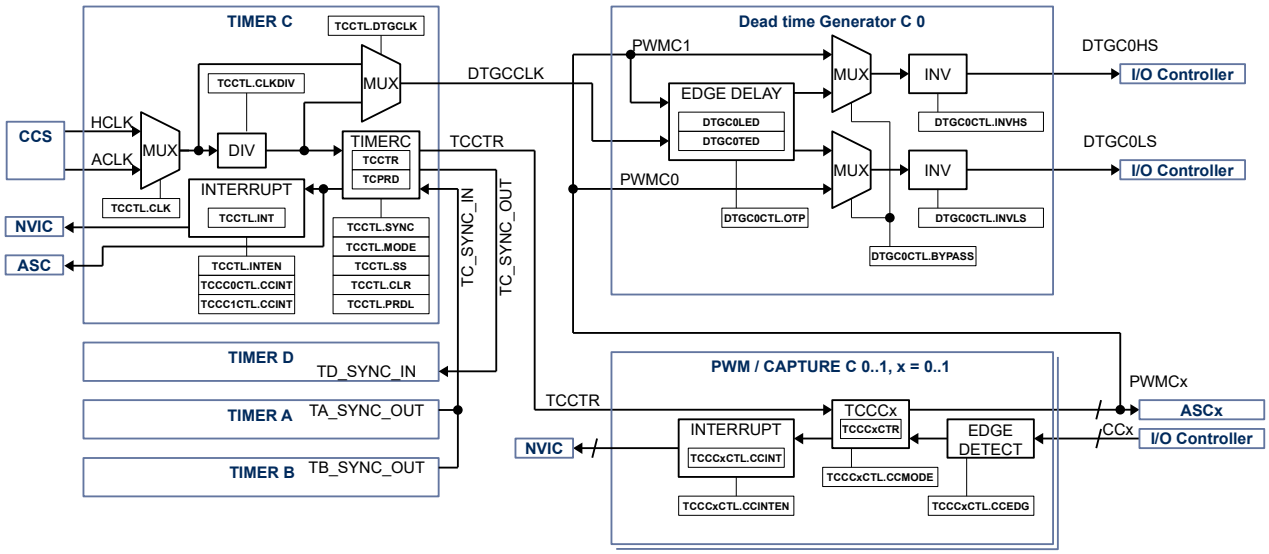
Register 14-10. DTGC0TED (Timer C Dead Time Generator 0 Trailing Edge Delay, 0x400F 00A8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGC0 trailing edge dead time in clock cycles defined by **TCCTL.DTGCLK**

14.2. Details of Operation

14.2.1. Block Diagram

Figure 14-1. Timer C



14.2.2. Configuration

Following blocks need to be configured for correct use of the Timer C:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller
- Gate Driver
- Auto sequencing controller (ASC)
- Timer A
- Timer B
- Timer D
-

14.2.3. Timer C Block

The timer C block consist of a 16-bit timer with up mode or up/down mode with 2 PWM/capture units and 1 dead-time generator unit.

14.2.4. Timer

Once enabled the timer counts up to the Timer C period value **TCPRD**. The **TCPRD** register can be written to while the timer is running, the new **TCPRD** value will be latched when the counter reaches old **TCPRD** value in up mode. In up/down mode there is the option to latch the new **TCPRD** value when counter counts back to zero. **TCCTL.PRDL** configures when the timer will be updated with the new **TCPRD** value in up/down mode.

The current timer value is accessible with the timer C counter value register **TCCTR**.

14.2.5. Register update

The **TCPRD**, **TCCCxCTR** register can be written to while the timer is running, the new **TCPRD, TCCCxCTR** value will be latched when the counter reaches old **TCPRD** value in up mode. In up/down mode there is the option to latch the new **TCPRD, TCCCxCTR** values when counter counts back to zero. **TCCTL.PRDL** configures when the timer will be updated with the new **TCPRD**, **TCCCxCTR** value in up/down mode.

14.2.6. Timer Modes

The timer supports 3 modes of operation: disabled, up and up/down using **TCCTL.MODE**.

By default, the timer mode is disabled. When the timer is disabled, the timer counter does not increment or decrement. If the timer is disabled when previously in up or up/down mode, then the timer counter stops where it is. If the timer is re-enabled by putting it back into up or up/down modes, then the counter continues from the point at which it was disabled. To reset the current counter value **TCCTR** to zero use **TCCTL.CLR**.

In up mode, the timer starts counting from 0 up to the value of **TCPRD**. When the timer counter reaches the value of **TCPRD**, then the timer counter is reset to a value of 0.This mode is typically used for timed events or edge-aligned PWM output.

In up/down mode, the timer starts counting from 0 up to the value of **TCPRD**, and then back down to a value of 0. This timer mode is typically used for center-aligned PWM output. It can also be used for timed events, and will allow a longer timer range due to the fact it counts up and down

14.2.7. Single Shot Mode

The timer can be configured to run either once or continuously.

When the timer is configured in single shot mode using **TCCTL.SS** the timer will only count to **TCPRD** value and stops in up mode. In up/down mode the timer will count to **TCPRD** and back to zero only once.

To start the timer in single-shot mode, **TCCTL.SS** must be set. The timer will start when **TCCTL.CLR** is set. To re-start a single-shot timer, **TCCTL.CLR** must be reset, and then set again.

14.2.8. Input clock and Pre-scaler

The timer can be configured to use HCLK or ACLK using **TCCTL.CLK**. The input clock for the can be divided further down from /1 to /128 using the **TCCTL.CLKDIV**.

14.2.9. Timer synchronization

The Timer A, B, C, D in the system have the ability to have synchronization between them. Each timer has a synchronization in signal (SYNC_IN) and the synchronization out signal (SYNC_OUT).

Timer C can be synchronized with Timer D as master. Timer B can be synchronized with Timer A, and B as slave.

The timer asserts the SYNC_OUT pulse when the **TCCTL.CLR** bit is set and de-asserts the SYNC_OUT pulse when the **TCCTL.CLR** bit is cleared.

If timer D that need to be synchronized as slave with master timer C need to set the **TxCTL.SYNC** bit. If this is bit is not set, then the sync_in signal is ignored and the timer operates independently.

When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is asserted, the timer clears the timer counter. The timer counter is also cleared anytime the **TxCTL.CLR** bit is set to a 1. When the **TxCTL.SYNC** bit is set and the

SYNC_IN signal is de-asserted and the timer mode is either up or up/down, then the timer will start counting. The timer will not start counting when the mode is set to up or up/down unless the SYNC_IN signal is de-asserted when **TxCTL.SYNC** is set.

NOTE:

In order for this feature to work correctly, all timers that are synchronized must be set to the same mode (up or up/down), with the same timer pre-scaler, timer clock input and timer period.

To enable synchronized timers, the following steps should be followed:

1. The slave timer C, D is configured with the selected timer input clock, timer pre-scaler, timer period and set the **TxCTL.SYNC** bit. The timer should still be set to disabled at this point.
2. The master timer A, or B is configured with the same timer input clock, timer pre-scaler, timer period and sets the **TxCTL.CLR** bit. This should clear all timer counters of the master and slave timers.
3. The slave timers set the timer mode to the desired state (either up or up/down).
4. The master timer sets the timer mode to either up or up/down and clears the **TxCTL.CLR** bit. This should start the master and all slave timers simultaneously based on the selected timer clock input.
5. Once configured as above, all timers can be disabled by the master setting **TxCTL.CLR** signal, to assert the SYNC_OUT signal. The timers can be re-enabled by clearing the **TxCTL.CLR** bit, which de-asserts the SYNC_OUT signal.

14.2.10. PWM/Compare Units

Timer C supports up to 2 PWM/Capture units PWMC0 to PWMC1. Each PWM/Compare unit can be configured independently in PWM mode or capture mode using **TCCCxCTL.CCMODE**.

14.2.10.1. PWM Mode

The PWM mode is enabled with setting **TCCCxCTRL.CCMODE** to 0.

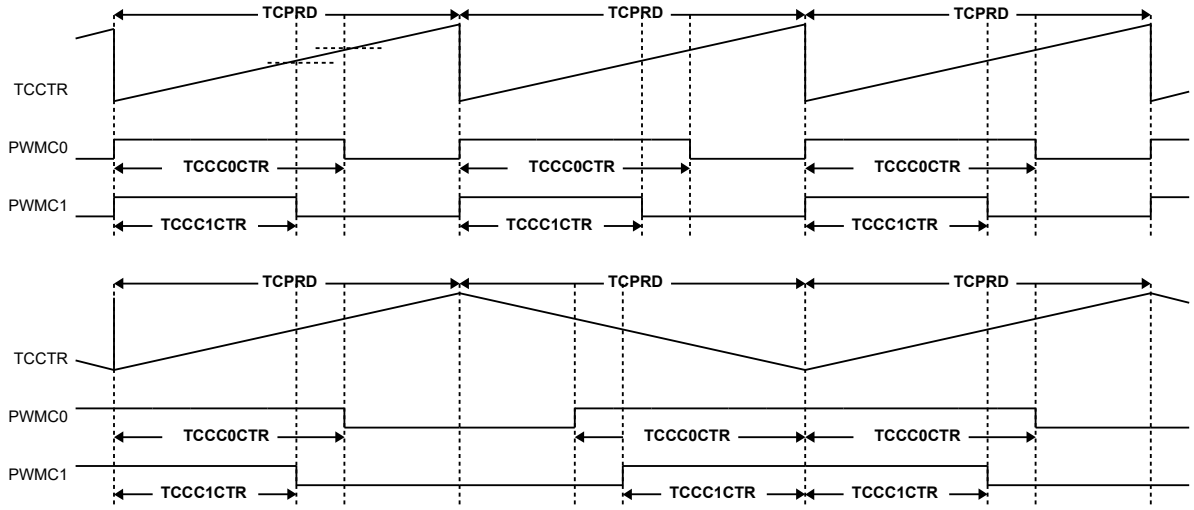
The timer configuration allows either edge-aligned (timer in up mode) or center-aligned (timer in up/down mode) modes of PWM operation.

In both edge-aligned and center-aligned modes of operation, the timer block outputs a PWM waveform that starts out high at a **TCCTR** value of 0 and then transitions to low when **TCCTR** counts up to **TCCCxCTR** compare value.

To configure a duty cycle of 0%, the **TCCCxCTR** should be set to 0; to configure a duty cycle of 100%, the **TCCCxCTR** value should be set to a value greater than or equal to **TCPRD**.

The polarity of the timer PWM outputs are not configurable. Adjustments to the polarity of the PWM outputs may be adjusted in the Dead-Time Generator (DTG) unit connected to the timer peripheral for each output independently.

Figure 14-2. PWMC0 and PWMC1 Example Using Timer C Up Mode and Up/Down Mode

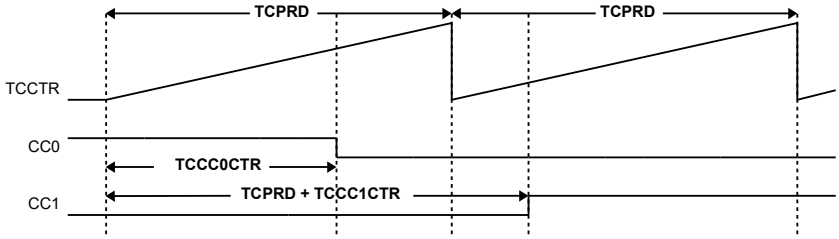


14.2.10.2. Capture Mode

The Capture mode is enabled with setting **TCCCxCTRL.CCMODE** to 1. The trip condition for capture mode can be configured using **TCCCxCTRL.CCEDGE**, high-to-low signal edge transition, low-to-high signal edge transition or both.

When trip condition is detected the actual **TCCTR** value is copied into **TCCCxCTR**.

Figure 14-3. CC0 and CC1 Capture Example



14.2.11. Timer and PWM/Capture Interrupt

The timer may generate interrupt based on the base timer wrap, or when a capture and compare event occurs.

In the base timer both up and up/down timer modes allow an interrupt to be generated when the count reaches 0. Each time the count reaches zero, the **TCCTL.INT** interrupt flag is set. If the interrupt is enabled using the **TCCTL.INTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TCCTL.INT** interrupt flag bit.

In the capture and compare PWM units, each time a compare threshold is reached or each time a capture event is detected the **TCCCxCTRL.CCINT** bit will be set for that particular timer unit. If the interrupt is enabled via the **TCCCxCTRL.CCINTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TCCCxCTRL.CCINT** interrupt flag bit.

The timer IRQ signal will be asserted if any of the timer interrupt flags **TCCTL.INT** or **TCCCxCTRL.CCINT** are set. The Timer IRQ signal will be de-asserted when all of the timer interrupt flags are cleared.

14.2.12. Dead-Time Generator

The dead-time generator can be configured to introduce dead-time for a complementary PWM output. The Timer C block supports up to 1 dead time generator.

14.2.12.1. Dead Time Input Clock Selection

The clock source for the DTGC0 can be selected using **TCCTL.DTGCLK**.

Clear **TCCTL.DTGCLK** to 0 to use clock source selected by **TCCTL.CLK** directly to use higher resolution for dead time insertion.

Set **TCCTL.DTGCLK** to 1 to use divided clock source selected by **TCCTL.CLK** and **TCCTL.CLKDIV** divider to use the same dead time resolution as Timer C.

14.2.12.2. Dead Time Range

The resolution for leading edge and trailing edge dead time is 12bits. Leading and trailing edge can be set independently using **DTGC0LED** and **DTGC0TED**.

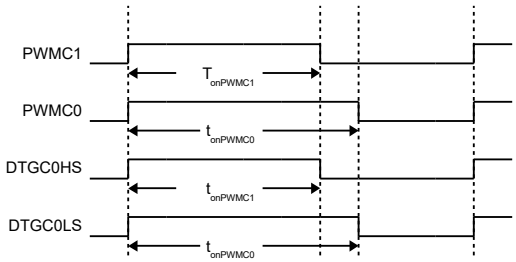
14.2.12.3. Bypass Mode

Set **DTGC0CTL.BYPASS** to 0 to enable dead time insertion.

Set **DTGC0CTL.BYPASS** to 1 to enable bypass mode, no dead time is inserted, PWMC1 is routed to DTGC0HS and PWMC0 is routed to DTGC0LS.

The DTGC0HS and DTGC0LS signals can be inverted in bypass mode.

Figure 14-4. DTGC0 Bypass Example



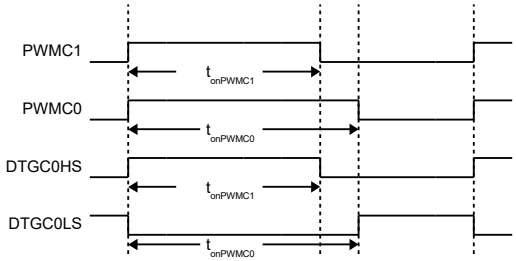
14.2.12.4. Inverting PWM Signal

The DTG output signals DTGC0HS and DTGC0LS can be inverted independently.

Set **DTGC0CTL.INVHS** to invert DTGC0HS signal.

Set **DTGC0CTL.INVLS** to invert DTGC0LS signal.

Figure 14-5. DTGC0 Bypass and Inverting LS Example



14.2.12.5. Dead Time Insertion

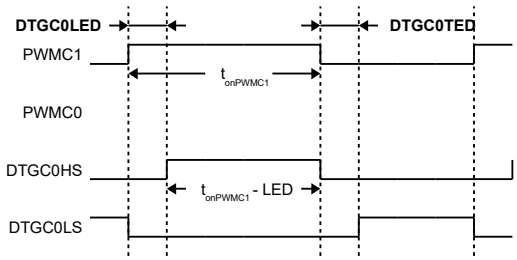
Set **DTGC0CTL.BYPASS** to 0 to enable dead time insertion. In dead time insertion mode only PWMC1 signal is used to generate DTGC0HS and DTGC0LS. PWMC0 signal is ignored and can be used for other purposes.

Set **DTGC0LED** for desired leading-edge and **DTGC0TED** for desired trailing edge in clock-cycles defined by **TCCTL.DTGCLK** clock source

NOTE:

In dead time insertion mode the DTGC0LS signal is automatically inverted compared to PWMC1 signal. Set **DTGC0CTL.INVLS** to 0, if this is desired behavior.

Figure 14-6. DTGC0 LED and TED Example



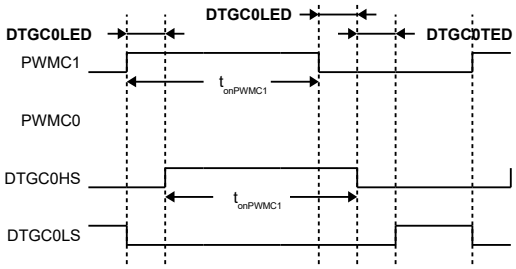
14.2.12.6. Dead Time Insertion With On Time Preservation

Set **DTGC0CTL.OTP** to 1 to enable on time preservation. In this mode the DTGC0HS is same as PWMC1 on time.

NOTE:

In dead time insertion mode the DTGC0LS signal is automatically inverted compared to PWMC1 signal. Set **DTGC0CTL.INVLS** to 0, if this is desired behavior.

Figure 14-7. DTGC0 LED and TED with On Time Preservation Example



14.2.13. PWM Output and Capture Input Pin Selection

Each of the DTGC0HS, DTGC0LS outputs, and CCx inputs can be routed to different I/Os, allowing great flexibility in pin assignments.

In capture mode only one I/O should be enabled as input to the capture. If more than one pin input is enabled, the capture might not work properly.

Note:

Not all pins are available pending package option, consult data sheet for available pins and signals.

Table 14-2. Timer C Signal to Pin Mapping

PWM	CAPTURE	DEADTIME	PINS
PWMC0	CC0	DTGC0LS	PA4
PWMC1	CC1	DTGC1LS	PA7, PD5

15. TIMER D

15.1. Register

15.1.1. Register Map

Table 15-1. Timer D Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
Timer D			
0x4010 0000	**TDCTL**	Timer D control	0x0000 0000
0x4010 0004	**TDPRD**	Timer D period	0x0000 0000
0x4010 0008	**TDCTR**	Timer D counter	0x0000 0000
Timer D PWMD Capture and Compare			
0x4010 0040	**TDCCTRL0**	Timer D capture and compare 0 control	0x0000 0000
0x4010 0044	**TDCTR0**	Timer D counter 0	0x0000 0000
0x4010 0048	**TDCCTRL1**	Timer D capture and compare 1 control	0x0000 0000
0x4010 004C	**TDCTR1**	Timer D counter 1	0x0000 0000
Timer D Dead Time Generator			
0x4010 00A0	**DTGD0CTL**	Timer D dead time generator 0 control	0x0000 0080
0x4010 00A4	**DTGD0LED**	Timer D dead time generator 0 leading edge delay	0x0000 0000
0x4010 00A8	**DTGD0TED**	Timer D dead time generator 0 trailing edge delay	0x0000 0000

15.1.2. TDCTL

Register 15-1. TDCTL (Timer D Control, 0x4010 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	RO	0x0	Reserved
13	**DTGCLK**	RW	0x0	DTGDx clock select 1b: DTGDx use clock after **TDCTL.CLKDIV** 0b: DTGDx use clock selected by **TDCTL.CLK**
12	**SYNC**	RW	0x0	Timer D synchronization 1b: Synchronize Timer D, enable SYNC_IN 0b: Do not synchronize Timer D, disabled SYNC_IN
11:10	**MODE**	RW	0x0	Timer D Mode 11b: reserved 10b: up / down 01b: up 00b: disabled
9	**CLK**	RW	0x0	Timer D clock input source 1b: ACLK 0b: HCLK
8:6	**CLKDIV**	RW	0x0	Timer D input clock divider 111b: / 128 110b: / 64 101b: /32 100b: /16 011b: /8 010b: /4 001b: /2 000b: /1

BIT	NAME	ACCESS	RESET	DESCRIPTION
5	**INTEN**	RW	0x0	Timer D interrupt enable 1b: enable Timer D interrupt 0b: disable Timer D interrupt
4	**INT**	RW1C	0x0	Timer D interrupt 1b: interrupt, clear by write 1b 0b: no interrupt
3	**SS**	RW	0x0	Timer D single shot 1b: single shot mode 0b: continuous timer mode
2	**CLR**	RW	0x0	Timer D clear 1b: Clear Timer D, hold Timer D in reset and set SYNC_OUT 0b: Do not clear Timer and clear SYNC_OUT
1	**Reserved**	RO	0x0	Reserved
0	**PRDL**	RW	0x0	Timer D TDPRD update 1b: Latch new TDPRD value when Timer D counting down, **TDCTR** value = 0x1 and **TDCTL.MODE** = 10b 0b: Latch new TDPRD value when Timer D counting up and **TDCTR** value = **TDPRD** – 0x1.

15.1.3. TDPRD

Register 15-2. TDPRD (Timer D Period, 0x4010 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0	Reserved
15:0	**PERIOD**	RW	0x0	Timer D period value

15.1.4. TDCTR

Register 15-3. TDCTR (Timer D Counter, 0x4010 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CTR**	RW	0x0	Current Timer D counter value

15.1.5. TDCC0CTL

Register 15-4. TDCC0CTRL (Timer D PWMD0 Capture and Compare Control, 0x4010 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMD0 input 0b: Compare mode PWMD0 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected

BIT	NAME	ACCESS	RESET	DESCRIPTION
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMD0 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

15.1.6. TDCC0CTR

Register 15-5. TDCC0CTR (Timer D PWMD0 Capture and Compare Counter, 0x4010 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMD0 compare mode or counter value for PWMD0 capture mode

15.1.7. TDCC1CTRL

Register 15-6. TDCC1CTL (Timer D PWMD1 Capture and Compare Control, 0x4010 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	RO	0x0	Reserved
4	**CCMODE**	RW	0x0	Capture and compare mode 1b: Capture mode PWMD1 input 0b: Compare mode PWMD1 output
3	**CCINTEN**	RW	0x0	Capture and compare interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**CCINT**	RW1C	0x0	Capture and compare interrupt 1b: interrupt, clear by write 1b 0b: no interrupt detected
1:0	**CCEDG**	RW	0x0	Capture mode edge detect PWMD1 11b: reserved 10b: high to low transition and low to high transition 01b: low to high transition only 00b: high to low transition only

15.1.8. TDCC1CTR

Register 15-7. TDCC1CTR (Timer D PWMD1 Capture and Compare Counter, 0x4010 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	RO	0x0	Reserved
15:0	**CCCTR**	RW	0x0	Counter value for PWMD1 compare mode or counter value for PWMD1 capture mode

15.1.9. DTGD0CTL

Register 15-8. DTGD0CTL (Timer D Dead Time Generator 0 Control, 0x4010 00A0)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RO	0x0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**BYPASS**	RW	0x1	Bypass dead time generator 1b: DTGD0 bypass active, DTGD0LS = PWMD0, DTGD0HS = PWMD1 0b: DTGD0 bypass inactive, dead time inserted to DTGD0LS and DTGD0HS, DTGD0LS = $\overline{\text{PWMD1}}$, DTGD0HS = PWMD1
6	**OTP**	RW	0x0	One Time Preservation 1b: DTGD0HS high time is same as PWMD1 high time and is shifted by **DTGD0LED** 0b: DTGD0HS high time is reduced by **DTGD0LED**
5	**INVHS**	RW	0x0	Invert DTGD0HS output signal 1b: invert DTGD0HS 0b: do not invert DTGD0HS
4	**INVLS**	RW	0x0	Invert DTGD0LS output signal 1b: invert DTGD0LS 0b: do not invert DTGD0LS
3:0	**Reserved**	RO	0x0	Reserved

15.1.10. DTGD0LED

Register 15-9. DTGD0LED (Timer D Dead Time Generator 0 Leading Edge Delay, 0x4010 00A4)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**LED**	RW	0x0	Counter value DTGD0 leading edge dead time in clock cycles defined by TDCTL.DTGCLK

15.1.11. DTGD0TED

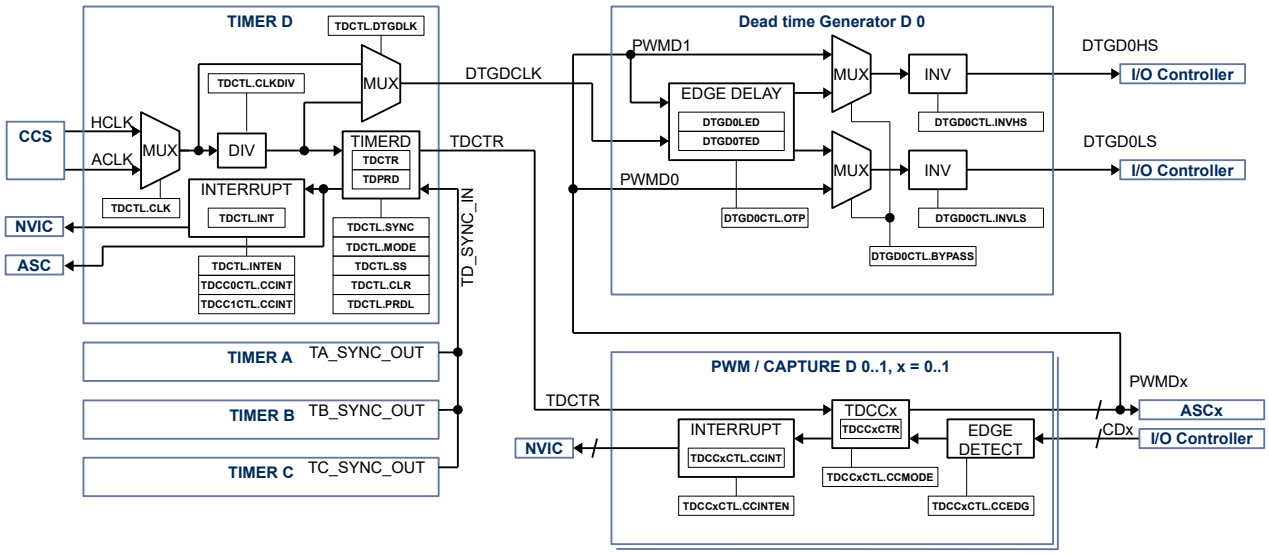
Register 15-10. DTGD0TED (Timer D Dead Time Generator 0 Trailing Edge Delay, 0x4010 00A8)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:12	**Reserved**	RO	0x0	Reserved
11:0	**TED**	RW	0x0	Counter value DTGD0 trailing edge dead time in clock cycles defined by TDCTL.DTGCLK

15.2. Details of Operation

15.2.1. Block Diagram

Figure 15-1. Timer D



15.2.2. Configuration

Following blocks need to be configured for correct use of the Timer D:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller
- Gate Driver
- Auto sequencing controller (ASC)
- Timer A
- Timer B
- Timer C

15.2.3. Timer D Block

The timer D block consist of a 16-bit timer with up mode or up/down mode with 2 PWM/capture units and 1 dead-time generator unit.

15.2.4. Timer

Once enabled the timer counts up to the Timer D period value **TDPRD**. The **TDPRD** register can be written to while the timer is running, the new **TDPRD** value will be latched when the counter reaches old **TDPRD** value in up mode. In up/down mode there is the option to latch the new **TDPRD** value when counter counts back to zero. **TDCTL.PRDL** configures when the timer will be updated with the new **TDPRD** value in up/down mode.

The current timer value is accessible with the timer D counter value register **TDCTR**.

15.2.5. Register Update

The **TDPRD**, **TDCCxCTR** register can be written to while the timer is running, the new **TDPRD, TDCCxCTR** value will be latched when the counter reaches old **TDPRD** value in up mode. In up/down mode there is the option to latch the new **TDPRD, TDCCxCTR** values when counter counts back to zero. **TDCTL.PRDL** configures when the timer will be updated with the new **TDPRD**, **TDCCxCTR** value in up/down mode.

15.2.6. Timer Modes

The timer supports 3 modes of operation: disabled, up and up/down using **TDCTL.MODE**.

By default, the timer mode is disabled. When the timer is disabled, the timer counter does not increment or decrement. If the timer is disabled when previously in up or up/down mode, then the timer counter stops where it is. If the timer is re-enabled by putting it back into up or up/down modes, then the counter continues from the point at which it was disabled. To reset the current counter value **TDCTR** to zero use **TDCTL.CLR**.

In up mode, the timer starts counting from 0 up to the value of **TDPRD**. When the timer counter reaches the value of **TDPRD**, then the timer counter is reset to a value of 0.This mode is typically used for timed events or edge-aligned PWM output.

In up/down mode, the timer starts counting from 0 up to the value of **TDPRD**, and then back down to a value of 0. This timer mode is typically used for center-aligned PWM output. It can also be used for timed events, and will allow a longer timer range due to the fact it counts up and down

15.2.7. Single Shot Mode

The timer can be configured to run either once or continuously.

When the timer is configured in single shot mode using **TDCTL.SS** the timer will only count to **TDPRD** value and stops in up mode. In up/down mode the timer will count to **TDPRD** and back to zero only once.

To start the timer in single-shot mode, **TDCTL.SS** must be set. The timer will start when **TDCTL.CLR** is set. To re-start a single-shot timer, **TDCTL.CLR** must be reset, and then set again.

15.2.8. Input Clock And Pre-Scaler

The timer can be configured to use HCLK or ACLK using **TDCTL.CLK**. The input clock for the can be divided further down from /1 to /128 using the **TDCTL.CLKDIV**.

15.2.9. Timer Synchronization

The Timer A, B, C, D in the system have the ability to have synchronization between them. Each timer has a synchronization in signal (SYNC_IN) and the synchronization out signal (SYNC_OUT).

Timer D can be synchronized with Timer A, B, or C as slave.

The timer asserts the SYNC_OUT pulse when the **TDCTL.CLR** bit is set and de-asserts the SYNC_OUT pulse when the **TDCTL.CLR** bit is cleared.

If timer D that need to be synchronized as slave with master timer C need to set the **TxCTL.SYNC** bit. If this is bit is not set, then the sync_in signal is ignored and the timer operates independently.

When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is asserted, the timer clears the timer counter. The timer counter is also cleared anytime the **TxCTL.CLR** bit is set to a 1. When the **TxCTL.SYNC** bit is set and the SYNC_IN signal is de-asserted and the timer mode is either up or up/down, then the timer will start counting. The timer will not start counting when the mode is set to up or up/down unless the SYNC_IN signal is de-asserted when **TxCTL.SYNC** is set.

NOTE:

In order for this feature to work correctly, all timers that are synchronized must be set to the same mode (up or up/down), with the same timer pre-scaler, timer clock input and timer period.

To enable synchronized timers, the following steps should be followed:

1. The slave timer B, C, D is configured with the selected timer input clock, timer pre-scaler, timer period

and set the **TxCTL.SYNC** bit. The timer should still be set to disabled at this point.

2. The master timer A, B or C is configured with the same timer input clock, timer pre-scaler, timer period and sets the **TxCTL.CLR** bit. This should clear all timer counters of the master and slave timers.
3. The slave timers set the timer mode to the desired state (either up or up/down).
4. The master timer sets the timer mode to either up or up/down and clears the **TxCTL.CLR** bit. This should start the master and all slave timers simultaneously based on the selected timer clock input.
5. Once configured as above, all timers can be disabled by the master setting **TxCTL.CLR** signal, to assert the SYNC_OUT signal. The timers can be re-enabled by clearing the **TxCTL.CLR** bit, which de-asserts the SYNC_OUT signal.

15.2.10. PWM/Compare Units

Timer D supports up to 2 PWM/Capture units PWMD0 to PWMD1. Each PWM/Compare unit can be configured independently in PWM mode or capture mode using **TDCCxCTL.CCMODE**.

15.2.10.1. PWM Mode

The PWM mode is enabled with setting **TDCCxCTRL.CCMODE** to 0.

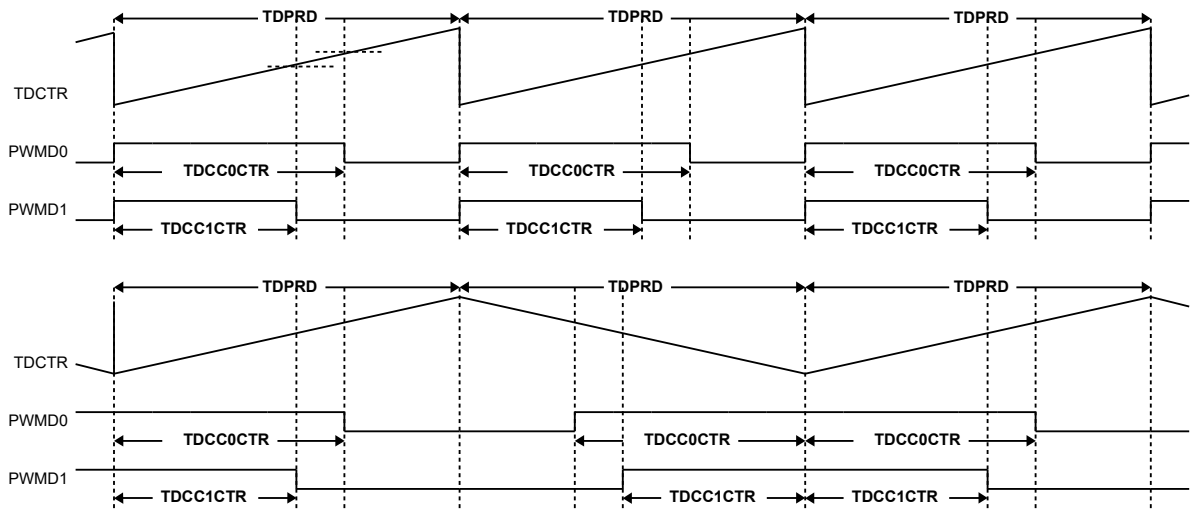
The timer configuration allows either edge-aligned (timer in up mode) or center-aligned (timer in up/down mode) modes of PWM operation.

In both edge-aligned and center-aligned modes of operation, the timer block outputs a PWM waveform that starts out high at a **TDCTR** value of 0 and then transitions to low when **TDCTR** counts up to **TDCCxCTR** compare value.

To configure a duty cycle of 0%, the **TDCCxCTR** should be set to 0; to configure a duty cycle of 100%, the **TDCCxCTR** value should be set to a value greater than or equal to **TDPRD**.

The polarity of the timer PWM outputs are not configurable. Adjustments to the polarity of the PWM outputs may be adjusted in the Dead-Time Generator (DTG) unit connected to the timer peripheral for each output independently.

Figure 15-2. PWMD0 and PWMD1 Example Using Timer D Up Mode and Up/Down Mode

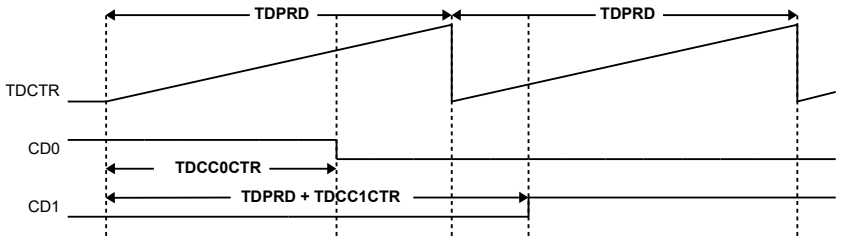


15.2.10.2. Capture Mode

The Capture mode is enabled with setting **TDCCxCTRL.CCMODE** to 1. The trip condition for capture mode can be configured using **TDCCxCTRL.CCEDGE**, high-to-low signal edge transition, low-to-high signal edge transition or both.

When trip condition is detected the actual **TDCTR** value is copied into **TDCCxCTR**.

Figure 15-3. CD0 and CD1 Capture Example



15.2.11. Timer and PWM/Capture Interrupt

The timer may generate interrupt based on the base timer wrap, or when a capture and compare event occurs.

In the base timer both up and up/down timer modes allow an interrupt to be generated when the count reaches 0. Each time the count reaches zero, the **TDCTL.INT** interrupt flag is set. If the interrupt is enabled using the **TDCTL.INTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TDCTL.INT** interrupt flag bit.

In the capture and compare PWM units, each time a compare threshold is reached or each time a capture event is detected the **TDCCxCTRL.CCINT** bit will be set for that particular timer unit. If the interrupt is enabled via the **TDCCxCTRL.CCINTEN**, then the Timer IRQ signal will be asserted to the CPU. The interrupt flag may be cleared by writing a 1 to the **TDCCxCTRL.CCINT** interrupt flag bit.

The timer IRQ signal will be asserted if any of the timer interrupt flags **TDCTL.INT** or **TDCCxCTRL.CCINT** are set. The Timer IRQ signal will be de-asserted when all of the timer interrupt flags are cleared.

15.2.12. Dead-Time Generator

The dead-time generator can be configured to introduce dead-time for a complementary PWM output. The Timer D block supports up to 1 dead time generator.

15.2.12.1. Dead Time Input Clock Selection

The clock source for the DTGD0 can be selected using **TDCTL.DTGCLK**.

Clear **TDCTL.DTGCLK** to 0 to use clock source selected by **TDCTL.CLK** directly to use higher resolution for dead time insertion.

Set **TDCTL.DTGCLK** to 1 to use divided clock source selected by **TDCTL.CLK** and **TDCTL.CLKDIV** divider to use the same dead time resolution as Timer D.

15.2.12.2. Dead Time Range

The resolution for leading edge and trailing edge dead time is 12bits. Leading and trailing edge can be set independently using **DTGD0LED** and **DTGD0TED**.

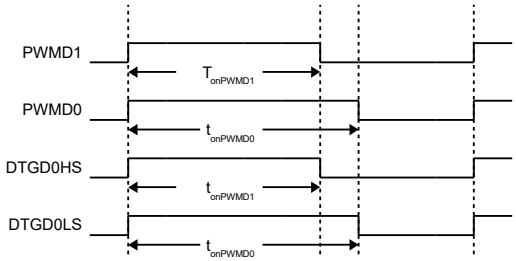
15.2.12.3. Bypass Mode

Set **DTGD0CTL.BYPASS** to 0 to enable dead time insertion.

Set **DTGD0CTL.BYPASS** to 1 to enable bypass mode, no dead time is inserted, PWMD1 is routed to DTGD0HS and PWMD0 is routed to DTGD0LS.

The DTGD0HS and DTGD0LS signals can be inverted in bypass mode.

Figure 15-4. DTGD0 Bypass Example



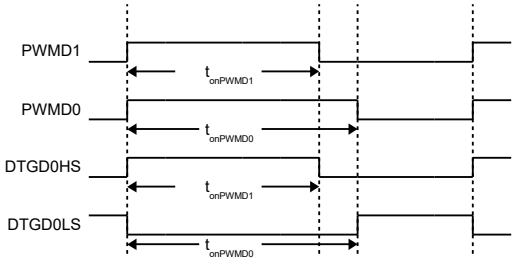
15.2.12.4. Inverting PWM Signal

The DTG output signals DTGD0HS and DTGD0LS can be inverted independently.

Set **DTGD0CTL.INVHS** to invert DTGD0HS signal.

Set **DTGD0CTL.INVLS** to invert DTGD0LS signal.

Figure 15-5. DTGD0 Bypass and Inverting LS Example



15.2.12.5. Dead Time Insertion

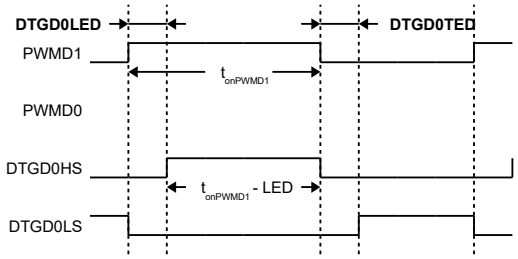
Set **DTGD0CTL.BYPASS** to 0 to enable dead time insertion. In dead time insertion mode only PWMD1 signal is used to generate DTGD0HS and DTGD0LS. PWMD0 signal is ignored and can be used for other purposes.

Set **DTGD0LED** for desired leading-edge and **DTGD0TED** for desired trailing edge in clock-cycles defined by **TDCTL.DTGCLK** clock source

NOTE:

In dead time insertion mode the DTGD0LS signal is automatically inverted compared to PWMD1 signal. Set **DTGD0CTL.INVLS** to 0, if this is desired behavior.

Figure 15-6. DTGD0 LED and TED Example



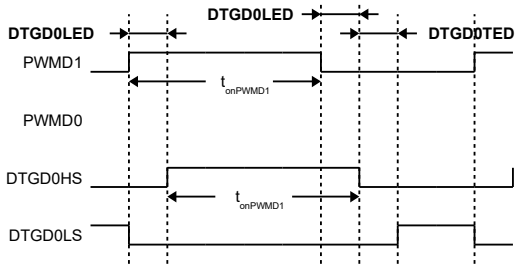
15.2.12.6. Dead Time Insertion with On Time Preservation

Set **DTGD0CTL.OTP** to 1 to enable on time preservation. In this mode the DTGD0HS is same as PWMD1 on time.

NOTE:

In dead time insertion mode the DTGD0LS signal is automatically inverted compared to PWMD1 signal. Set **DTGD0CTL.INVLS** to 0, if this is desired behavior.

Figure 15-7. DTGD0 LED and TED with On Time Preservation Example



15.2.13. PWM Output and Capture Input Pin Selection

Each of the DTGD0HS, DTGD0LS outputs, and CDx inputs can be routed to different I/Os, allowing great flexibility in pin assignments.

In capture mode only one I/O should be enabled as input to the capture. If more than one pin input is enabled, the capture might not work properly.

Note:

Not all pins are available pending package option, consult data sheet for available pins and signals.

Table 15-2. Timer D Signal to Pin Mapping

PWM	CAPTURE	DEADTIME	PINS
PWMD0	CD0	DTGD0LS	PA5, PD7
PWMD1	CD1	DTGD1LS	PD4

16. FLASH MEMORY CONTROLLER

16.1. Register

16.1.1. Register Map

Table 16-1. FLASH Memory Controller Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
FLASH Memory Controller			
0x4002 0000	**FLASHLOCK**	FLASH lock	0x0000 0000
0x4002 0004	**FLASHCTL**	FLASH control and status	0x0000 0000
0x4002 0008	**FLASHPAGE**	FLASH page selection	0x0000 0000
0x4002 000C	**Reserved**	Reserved	0x0000 0000
0x4002 0010	**Reserved**	Reserved	0x0000 0000
0x4002 0014	**FLASHPERASE**	FLASH page erase	0x0000 0000
0x4002 0018	**Reserved**	Reserved	0x0000 0000
0x4002 001C	**Reserved**	Reserved	0x0000 0000
0x4002 0020	**Reserved**	Reserved	0x0000 0000
0x4002 0024	**SWDACCESS**	SWD access control	0x0000 0000
0x4002 0028	**FLASHWSTATE**	FLASH wait state control	0x0000 0003
0x4002 002C	**FLASHBWRITE**	FLASH buffered write enable	0x0000 0000
0x4002 0030	**FLASHBWDATA**	FLASH buffered write data and address	0x0000 0000

16.1.2. FLASHLOCK

Register 16-1. FLASHLOCK (FLASH Lock, 0x4002 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:0	**LOCK**	RW	0x0	Unlock access to FLASH registers and FLASH memory 0xAAAA AAAA: allow write and erase to FLASH pages 0x1234 5678: allow write of FLASHWSTATE register 0xF983 62AB: allow write access to address 0x0010 0008 to disable SWD

16.1.3. FLASHCTL

Register 16-2. FLASHCTL (FLASH Control and Status, 0x4002 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	R	0x0	Reserved
4	**Reserved**	R	0x0	Reserved
3	**Reserved**	R	0x0	Reserved
2	**Reserved**	R	0x0	Reserved
1	**PERASE**	R	0x0	Page erase active 1b: page erase in progress 0b: page erase finished or no page erase in progress
0	**WRITE**	R	0x0	Buffered write active, only used in conjunction with FLASHBWRITE 1b: buffered write in progress 0b: buffered write finished or no buffered write in progress

16.1.4. FLASHPAGE

Register 16-3. FLASHPAGE (FLASH Page Selector, 0x4002 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	R	0x0	Reserved
4:0	**PAGE**	RW	0x0	FLASH page selector for page erase 0x1F: page 31, 0x0000 7C00 to 0x0000 7FFF 0x1E: page 30, 0x0000 7800 to 0x0000 7BFF 0x1D: page 29, 0x0000 7400 to 0x0000 73FF 0x1C: page 28, 0x0000 7000 to 0x0000 73FF 0x1B: page 27, 0x0000 6C00 to 0x0000 6FFF 0x1A: page 26, 0x0000 6800 to 0x0000 6BFF 0x19: page 25, 0x0000 6400 to 0x0000 67FF 0x18: page 24, 0x0000 6000 to 0x0000 63FF 0x17: page 23, 0x0000 5C00 to 0x0000 5FFF 0x16: page 22, 0x0000 5800 to 0x0000 5BFF 0x15: page 21, 0x0000 5400 to 0x0000 57FF 0x14: page 20, 0x0000 5000 to 0x0000 53FF 0x13: page 19, 0x0000 4C00 to 0x0000 4FFF 0x12: page 18, 0x0000 4800 to 0x0000 4BFF 0x11: page 17, 0x0000 4400 to 0x0000 47FF 0x10: page 16, 0x0000 4000 to 0x0000 43FF 0x0F: page 15, 0x0000 3C00 to 0x0000 3FFF 0x0E: page 14, 0x0000 3800 to 0x0000 3BFF 0x0D: page 13, 0x0000 3400 to 0x0000 37FF 0x0C: page 12, 0x0000 3000 to 0x0000 33FF 0x0B: page 11, 0x0000 2C00 to 0x0000 2FFF 0x0A: page 10, 0x0000 2800 to 0x0000 2BFF 0x09: page 9, 0x0000 2400 to 0x0000 27FF 0x08: page 8, 0x0000 2000 to 0x0000 23FF 0x07: page 7, 0x0000 1C00 to 0x0000 1FFF 0x06: page 6, 0x0000 1800 to 0x0000 1BFF 0x05: page 5, 0x0000 1400 to 0x0000 17FF 0x04: page 4, 0x0000 1000 to 0x0000 13FF 0x03: page 3, 0x0000 0C00 to 0x0000 0FFF 0x02: page 2, 0x0000 0800 to 0x0000 0BFF 0x01: page 1, 0x0000 0400 to 0x0000 07FF 0x00: page 0, 0x0000 0000 to 0x0000 03FF

16.1.5. FLASHPERASE

Register 16-4. FLASHPERASE (FLASH Page Erase, 0x4002 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:0	**PERASE**	R	0x0	Write of correct key value to this register starts FLASH page erase selected in FLASHPAGE.PAGE 0xA5A5 5A5A: start FLASH page erase selected by FLASHPAGE.PAGE

16.1.6. SWDACCESS

Register 16-5. SWDACCESS (SDW Access Status, 0x4002 0024)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:0	**DEBUG**	R	0x0	Status of SWD debug 0xFFFFFFFF: SWD access is enabled 0x69696969: SWD access is disabled

16.1.7. FLASHWSTATE

Register 16-6. FLASHWSTATE (FLASH Access Wait State, 0x4002 0028)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:2	**Reserved**	RW	0x0	Reserved, must be set to 0x0
1:0	**WSTATE**	RW	0x3	FLASH access wait state 11b: 3 wait states for 75MHz < HCLK < 100MHz 10b: 2 wait states for 50MHz < HCLK < 75MHz 01b: 1 wait state for 25MHz < HCLK < 50MHz 00b: 0 wait state for HCLK < 25MHz

16.1.8. FLASHBWRITE

Register 16-7. FLASHBWRITE (Buffered FLASH Write, 0x4002 002C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:0	**BWRITE**	RW	0x0	Write of correct key value to this register starts buffered write operation 0xCA72 6B18: start buffered write of FLASHBWDATA

16.1.9. FLASHBWDATA

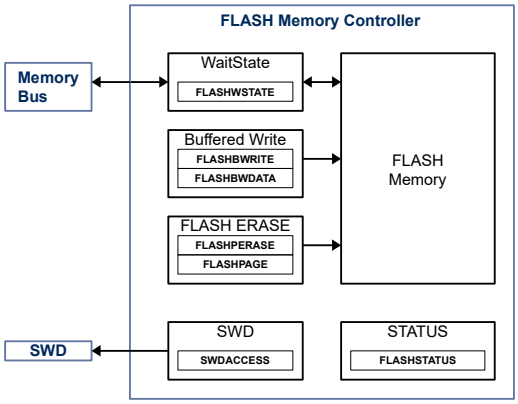
Register 16-8. FLASHBWDATA (Buffered FLASH Write Data, 0x4002 0030)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31	**Reserved**	RW	0x0	Reserved, must be set to 0x0
30	**Reserved**	RW	0x0	Reserved, must be set to 0x0
29:25	**PAGE**	RW	0x0	FLASH page selector to write to 0x1F: page 31, 0x0000 7C00 to 0x0000 7FFF 0x1E: page 30, 0x0000 7800 to 0x0000 7BFF 0x1D: page 29, 0x0000 7400 to 0x0000 73FF 0x1C: page 28, 0x0000 7000 to 0x0000 73FF 0x1B: page 27, 0x0000 6C00 to 0x0000 6FFF 0x1A: page 26, 0x0000 6800 to 0x0000 6BFF 0x19: page 25, 0x0000 6400 to 0x0000 67FF 0x18: page 24, 0x0000 6000 to 0x0000 63FF 0x17: page 23, 0x0000 5C00 to 0x0000 5FFF 0x16: page 22, 0x0000 5800 to 0x0000 5BFF 0x15: page 21, 0x0000 5400 to 0x0000 57FF 0x14: page 20, 0x0000 5000 to 0x0000 53FF 0x13: page 19, 0x0000 4C00 to 0x0000 4FFF 0x12: page 18, 0x0000 4800 to 0x0000 4BFF 0x11: page 17, 0x0000 4400 to 0x0000 47FF 0x10: page 16, 0x0000 4000 to 0x0000 43FF 0x0F: page 15, 0x0000 3C00 to 0x0000 3FFF 0x0E: page 14, 0x0000 3800 to 0x0000 3BFF 0x0D: page 13, 0x0000 3400 to 0x0000 37FF 0x0C: page 12, 0x0000 3000 to 0x0000 33FF 0x0B: page 11, 0x0000 2C00 to 0x0000 2FFF 0x0A: page 10, 0x0000 2800 to 0x0000 2BFF 0x09: page 9, 0x0000 2400 to 0x0000 27FF 0x08: page 8, 0x0000 2000 to 0x0000 23FF 0x07: page 7, 0x0000 1C00 to 0x0000 1FFF 0x06: page 6, 0x0000 1800 to 0x0000 1BFF 0x05: page 5, 0x0000 1400 to 0x0000 17FF 0x04: page 4, 0x0000 1000 to 0x0000 13FF 0x03: page 3, 0x0000 0C00 to 0x0000 0FFF 0x02: page 2, 0x0000 0800 to 0x0000 0BFF 0x01: page 1, 0x0000 0400 to 0x0000 07FF 0x00: page 0, 0x0000 0000 to 0x0000 03FF
24:16	**ADDRESS**	RW	0x0	Relative half word address within selected FLASHBWDATA.PAGE
15:0	**DATA**	RW	0x0	Data to write

16.2. Details of Operation

16.2.1. Block Diagram

Figure 16-1. FLASH Memory Controller



16.2.2. Configuration

Following blocks need to be configured for correct use of the FLASH:

- Clock Control System (CCS)

16.2.3. FLASH Memory

The Flash memory controller allows configuration of the FLASH memory. FLASH wait states, FLASH erase, buffered FLASH write, and SWD debug access can be configured. The FLASH memory has up to 32 pages of 1kByte each.

16.2.4. Writing to FLASH Controller Registers

The FLASH Controller registers are write protected to reduce chances of accidental erase or modification of FLASH memory. Each write to a FLASH controller register is a 2 step process. The first step is to write the correct key into **FLASHLOCK** followed by a FLASH controller register write.

Without correct key any writes to FLASH controller register will be ignored. Flash controller reads are always possible.

16.2.5. FLASH Wait State

After device reset, the **FLASHWSTATE** is set to 0x3. To allow optimal FLASH access time without delay, the FLASH wait state need to be set according to HCLK frequency used. See register table for correct setting.

To write to **FLASHWSTATE** register, **FLASHLOCK** need to be set to 0x1234 5678.

16.2.6. FLASH Page Erase

To erase a page of FLASH memory, set **FLASHLOCK** to 0xAAAA AAAA first, then set **FLASHBWDATA.PAGE** to the page to be erased and then set **FLASHPERASE** to 0xA5A5 5A5A. The FLASH page operation will start, **FLASHCTL.PERASE** is set to 1b and any access to FLASH memory address space is stalled until erase

operation is finished. **FLASHCTL.PERASE** is set to 0b when erase operation is finished.

It is not recommended to erase FLASH pages while executing from FLASH as any access to FLASH is stalled until the erase operation is finished. Either execute from SRAM or use SWD debug interface.

16.2.7. Write to FLASH

Only half-word address aligned half-word writes are supported. To write a half word to FLASH memory, make sure the memory location is erased by doing a read, it should return 0xFFFF. Set **FLASHLOCK** to 0xAAAA AAAA first, then write a half-word to the memory address directly.

It is not recommended to write to FLASH while executing from FLASH as any access to FLASH is stalled until the erase operation is finished. Either execute from SRAM or use SWD debug interface.

16.2.8. Buffered Write to FLASH

Only half-word address aligned half-word writes are supported. The FLASH memory controller also allows buffered write to FLASH, to allow CPU still react to interrupts or perform other tasks while waiting for FLASH write to finish when executing from SRAM.

To write a half word to FLASH memory, make sure the memory location is erased by doing a read, it should return 0xFFFF. Set **FLASHLOCK** to 0xAAAA AAAA first, then write to **FLASHBWDATA**, with **FLASHBWDATA.DATA** the half-word you want to write, and **FLASHBWDATA.PAGE** the page where the memory location resides and **FLASHBWDATA.ADDRESS** the relative address of the memory location.

The FLASH page operation will start, **FLASHCTL.WRITE** is set to 1b, AHB bus control will be given back to CPU to allow execution of other commands. Any access to FLASH memory while buffered write operation is active will stall. **FLASHCTL.WRITE** is set to 0b when buffered write operation is finished.

To calculate **FLASHBWDATA.PAGE** use:

$$PAGE = \frac{Memoryaddress}{pagesize} \tag{4}$$

Where:

PAGE: integer value for **FLASHBWDATA.PAGE**

Memoryaddress: Word memory address

pagesize: FLASH page size: 0x400

Then to calculate **FLASHBWDATA.ADDRESS** use:

$$ADDRESS = \frac{(Memoryaddress - PAGE * pagesize)}{2} \tag{5}$$

Where:

ADDRESS: integer value for **FLASHBWDATA.ADRESS**

PAGE: integer value for **FLASHBWDATA.PAGE**

Memoryaddress: Word memory address

pagesize: FLASH page size: 0x400

Example:

memoryaddress: 0x0000 0438

PAGE = 0x0000 0438 / 0x400 = 0x01

ADDRESS = (0x0000 0438 – 0x01*0x400)/0x2 = 0x38/0x2 = 0x1C

It is not recommended to write to FLASH while executing from FLASH as any access to FLASH is stalled until the erase operation is finished. Either execute from SRAM or use SWD debug interface.

16.2.9. SWD Debug Access Disable

The SWD debug access is enabled by default and can be disabled by a FUSE to prevent access of device memory.

Caution need to be taken. This action is not reversible.

To disable SWD set **FLASHLOCK** to 0xF983 62AB first, then write 0x6969 6969 to address 0x0010 0008 will disable the SWD debug access.

17. ADC AND AUTO SEQUENCER

17.1. Register

17.1.1. Register Map

Table 17-1. Register Map – EMUX

ADRESS	NAME	DESCRIPTON	RESET VALUE
EMUX			
0x4015 0000	EMUXCTL	ADC external MUX control register	0x0000 0000
0x4015 0004	EMUXDATA	ADC external MUX data register	0x0000 0000

Table 17-2. Register Map – ADC

ADRESS	NAME	DESCRIPTON	RESET VALUE
ADC			
0x4015 0008	ADCCTL	ADC control register	0x0000 0000
0x4015 000C	ADCR	ADC conversion result register	0x0000 0000
0x4015 0010	ADCINT	ADC Interrupt register	0x0000 0000

Table 17-3. Register Map – ADC Auto Sequencer 0

ADRESS	NAME	DESCRIPTON	RESET VALUE
ADC Auto Sequencer -0			
0x4015 0040	AS0CTL	Automated ADC sampling 0 control register	0x0000 0000
0x4015 0044	AS0S0	Automated Sampling 0 Sequence 0 register	0x0000 0000
0x4015 0048	AS0R0	Automated Sampling 0 Sample result 0 register	0x0000 0000
0x4015 004C	AS0S1	Automated Sampling 0 Sequence 1 register	0x0000 0000
0x4015 0050	AS0R1	Automated Sampling 0 Sample result 1 register	0x0000 0000
0x4015 0054	AS0S2	Automated Sampling 0 Sequence 2 register	0x0000 0000
0x4015 0058	AS0R2	Automated Sampling 0 Sample result 2 register	0x0000 0000
0x4015 005C	AS0S3	Automated Sampling 0 Sequence 3 register	0x0000 0000
0x4015 0060	AS0R3	Automated Sampling 0 Sample result 3 register	0x0000 0000
0x4015 0064	AS0S4	Automated Sampling 0 Sequence 4 register	0x0000 0000
0x4015 0068	AS0R4	Automated Sampling 0 Sample result 4 register	0x0000 0000
0x4015 006C	AS0S5	Automated Sampling 0 Sequence 5 register	0x0000 0000
0x4015 0070	AS0R5	Automated Sampling 0 Sample result 5 register	0x0000 0000
0x4015 0074	AS0S6	Automated Sampling 0 Sequence 6 register	0x0000 0000
0x4015 0078	AS0R6	Automated Sampling 0 Sample result 6 register	0x0000 0000
0x4015 007C	AS0S7	Automated Sampling 0 Sequence 7 register	0x0000 0000
0x4015 0080	AS0R7	Automated Sampling 0 Sample result 7 register	0x0000 0000

Table 17-4. Register Map – ADC Auto Sequencer 1

ADRESS	NAME	DESCRIPTON	RESET VALUE
ADC Auto Sequencer -1			
0x4015 0100	AS1CTL	Automated ADC sampling 1 control register	0x0000 0000
0x4015 0104	AS1S0	Automated Sampling 1 Sequence 0 register	0x0000 0000
0x4015 0108	AS1R0	Automated Sampling 1 Sample result 0 register	0x0000 0000
0x4015 010C	AS1S1	Automated Sampling 1 Sequence 1 register	0x0000 0000
0x4015 0110	AS1R1	Automated Sampling 1 Sample result 1 register	0x0000 0000
0x4015 0114	AS1S2	Automated Sampling 1 Sequence 2 register	0x0000 0000
0x4015 0118	AS1R2	Automated Sampling 1 Sample result 2 register	0x0000 0000
0x4015 011C	AS1S3	Automated Sampling 1 Sequence 3 register	0x0000 0000
0x4015 0120	AS1R3	Automated Sampling 1 Sample result 3 register	0x0000 0000
0x4015 0124	AS1S4	Automated Sampling 1 Sequence 4 register	0x0000 0000
0x4015 0128	AS1R4	Automated Sampling 1 Sample result 4 register	0x0000 0000
0x4015 012C	AS1S5	Automated Sampling 1 Sequence 5 register	0x0000 0000
0x4015 0130	AS1R5	Automated Sampling 1 Sample result 5 register	0x0000 0000
0x4015 0134	AS1S6	Automated Sampling 1 Sequence 6 register	0x0000 0000
0x4015 0138	AS1R6	Automated Sampling 1 Sample result 6 register	0x0000 0000
0x4015 013C	AS1S7	Automated Sampling 1 Sequence 7 register	0x0000 0000
0x4015 0140	AS1R7	Automated Sampling 1 Sample result 7 register	0x0000 0000

17.1.2. EMUXCTL

Register 17-1. EMUXCTL (ADC external MUX control register 0x4015 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:6	**Reserved**	R	0x0	Reserved
5	**EMUXC**	RW	0x0	ADC external MUX control 1b: EMUX is controlled by auto-sequencer unit 0b: EMUX manual control
4	**EMUXBUSY**	RW	0x0	ADC external MUX status 1b: ADC external MUX sending data 0b: ADC external MUX not busy or not used
3	**EMUXDONE**	RW	0x0	ADC external MUX data send done, auto-clear with read or when new data is sent over EMUX 1b: ADC external MUX data sent 0b: ADC external MUX busy
2:0	**EMUXCDIV**	RW	0x0	ADC external mux clock to FCLK divider 111b: FCLK/8 110b: FCLK/7 101b: FCLK.6 100b: FCLK/5 011b: FCLK/4 010b: FCLK/3 001b: FCLK/2 000b: FCLK/1

17.1.3. EMUXDATA

Register 17-2. EMUXDATA (EMUX data register 0x4015 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**DATA**	RW	0x0	EMUX data, writing this register will start transmission over EMUX

17.1.4. ADCCTL

Register 17-3. ADCCTL (ADC control register 0x4015 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	R	0x0	Reserved
15	**ADCEN**	RW	0x0	ADC module enable 1b: enable ADC module 0b: turn off ADC module
14	**ADCSTART**	R/W	0x0	Start ADC conversion. A write of 1b will start ADC conversion if ADCCTL.ADCBUSY = 0b and ADCCTL.ADCMODE = 00b. 1b: start ADC conversion 0b: stop ADC conversion, also stop repeated ADC conversions ADCCTL.ADCREPEAT = 1b.
13	**ADCREPEAT**	R/W	0x0	ADC repeat mode 1b: repeated conversion, also auto rearms auto sequencer 0b: single shot conversion
12:10	**ADCMODE**	R/W	0x0	ADC conversion mode 111b: automated sequencer 0 and 1 independently triggered 110b: automated sequencer 0 and 1 daisy chained trigger on AS0 101b: automated sequencer 1 only trigger condition 100b: automated sequencer 0 only trigger condition 011b: automated sequencer 0 and 1 daisy chained 010b: automated sequencer 1 only 001b: automated sequencer 0 only 000b: single channel
9:8	**Reserved**	R	0x0	Reserved
7	**ADCBUSY**	R	0x0	ADC busy 1b: ADC conversion or auto sequencer active 0b: ADC no operation
6:4	**ADCMUX**	R/W	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: AD1 000b: AD0
3	**DONE**	R	0x0	ADC Conversion Complete 0b: not complete 1b: complete
2:0	**ADCCDIV**	R/W	0x0	ADC input clock FCLK divider 111b: FCLK/8 110b: FCLK/7 101b: FCLK.6 100b: FCLK/5 011b: FCLK/4 010b: FCLK/3 001b: FCLK/2 000b: FCLK/1

17.1.5. ADCCR

Register 17-4. ADCCR (ADC conversion result register 0x4015 000C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.6. ADCINT

Register 17-5. ADCINT (ADC Interrupt register 0x4015 0010)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:13	**Reserved**	R	0x0	Reserved
19:18	**ASCINTSEQ**	RW	0x0	Last Auto sequencer to trigger ADCINT.ASCINT 11b: Auto sequencer 1 and 0 triggered interrupt 10b: Auto sequencer 1 triggered interrupt 01b: Auto sequencer 0 triggered interrupt 00b: no trigger
17:16	**ASCINTTR**	RW	0x0	Last Auto sequencer to run 11b: reserved 10b: auto sequencer 1 to run 01b: auto sequencer 0 to run 00b: no sequencer to r
15:13	**Reserved**	R	0x0	Reserved
12	**ASCINT_EN**	RW	0x0	Enable auto sequencer collision interrupt 1b: enable ASCINT 0b: disable ASCINT
11	**AS1INT_EN**	RW	0x0	Enable auto sequencer 1 conversions finished interrupt 1b: enable AS1INT 0b: disable AS1INT
10	**AS0INT_EN**	RW	0x0	Enable auto sequencer 0 conversions finished interrupt 1b: enable AS0INT 0b: disable AS0INT
9	**EMUXINT_EN**	RW	0x0	Enable EMUX transfer finished interrupt 1b: enable EMUXINT 0b: disable EMUXINT
8	**ADCINT_EN**	RW	0x0	Enable ADC conversion finished interrupt 1b: enable ADCINT 0b: disable ADCINT
7:5	**Reserved**	RW	0x0	Reserved, must be set to 0
4	**ASCINT**	RW	0x0	Auto sequencer collision interrupt 1b: interrupt, clear by writing 1b to it 0b: no interrupt NOTE: This bit is cleared by writing a 1b to it.
3	**AS1INT**	RW	0x0	Auto sequencer 1 conversions finished interrupt 1b: interrupt, clear by writing 1b to it 0b: no interrupt NOTE: This bit is cleared by writing a 1b to it.

BIT	NAME	ACCESS	RESET	DESCRIPTION
2	**AS0INT**	RW	0x0	Auto sequencer 0 conversions finished interrupt 1b: interrupt, clear by writing 1b to it 0b: no interrupt NOTE: This bit is cleared by writing a 1b to it.
1	**EMUXINT**	RW	0x0	EMUX data transfer finished interrupt 1b: interrupt, clear by writing 1b to it 0b: no interrupt NOTE: This bit is cleared by writing a 1b to it.
0	**ADCINT**	RW	0x0	ADC conversion finished interrupt 1b: interrupt, clear by writing 1b to it 0b: no interrupt NOTE: This bit is cleared by writing a 1b to it.

17.1.7. AS0CTL

Register 17-6. AS0CTL (Auto Sequencer 0 control register 0x4015 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:13	**Reserved**	R	0x0	Reserved
12	**AS0BUSY**	RW	0x0	Auto sequencer 0 busy 1b: auto sequencer 0 sampling active 0b: auto sequencer 0 not active
11	**AS0EN**	RW	0x0	Auto sequencer 0 enable 1b: auto sequencer 0 enabled 0b: auto sequencer 0 not enabled
10:8	**AS0D**	RW	0x0	Auto sequencer 0 sampling depth 111b: 8 samples 110b: 7 samples 101b: 6 samples 100b: 5 samples 011b: 4 samples 010b: 3 samples 001b: 2 samples 000b: 1 sample
7	**AS0TR**	RW	0x0	Auto sequencer 0 trigger source 1b: Timer, as defined by AS0CTL.AS0TRTMR 0b: PWM, as defined by AS0CTL.AS0TRPWM
6	**AS0TRE**	RW	0x0	Auto sequencer 0 trigger source AS0CTL.AS0TR edge 1b: high to low edge 0b: low to high edge
5:4	**AS0TRTMR**	RW	0x0	Auto sequencer 0 timer trigger source 11b: Timer D 10b: Timer C 01b: Timer B 00b: Timer A

BIT	NAME	ACCESS	RESET	DESCRIPTION
3:0	**AS0TRPWM**	RW	0x0	Auto sequencer 0 PWM trigger source 1111b: reserved 1110b: reserved 1101b: PWMD1 1100b: PWMD0 1011b: PWMC1 1010b: PWMC0 1001b: PWMB1 1000b: PWMB0 0111b: PWMA7 0110b: PWMA6 0101b: PWMA5 0100b: PWMA4 0011b: PWMA3 0010b: PWMA2 0001b: PWMA1 0000b: PWMA0

17.1.8. AS0S0

Register 17-7. AS0S0 (Auto sequencer 0-sample 0 control 0x4015 0044)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S0.EMUXD data after S/H of ADC 01b: send AS0S0.EMUXD data at beginning of this sample sequence 00b: do not send AS0S0.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.9. AS0R0

Register 17-8. AS0R0 (Auto sequencer 0-sample 0 result register 0x4015 0048)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.10. AS0S1

Register 17-9. AS0S1 (Auto sequencer 0-sample 1 control 0x4015 004C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S1.EMUXD data after S/H of ADC 01b: send AS0S1.EMUXD data at beginning of this sample sequence 00b: do not send AS0S1.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.11. AS0R1

Register 17-10. AS0R1 (Auto sequencer 0-sample 1 result register 0x4015 0050)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.12. AS0S2

Register 17-11. AS0S2 (Auto sequencer 0-sample 2 control 0x4015 0054)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock

BIT	NAME	ACCESS	RESET	DESCRIPTION
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S2.EMUXD data after S/H of ADC 01b: send AS0S2.EMUXD data at beginning of this sample sequence 00b: do not send AS0S2.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.13. AS0R2

Register 17-12. AS0R2 (Auto sequencer 0-sample 2 result register 0x4015 0058)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.14. AS0S3

Register 17-13. AS0S3 (Auto sequencer 0-sample 3 control 0x4015 005C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S3.EMUXD data after S/H of ADC 01b: send AS0S3.EMUXD data at beginning of this sample sequence 00b: do not send AS0S3.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.15. AS0R3

Register 17-14. AS0R3 (Auto sequencer 0-sample 3 result register 0x4015 0060)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.16. AS0S4

Register 17-15. AS0S4 (Auto sequencer 0-sample 4 control 0x4015 0064)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S4.EMUXD data after S/H of ADC 01b: send AS0S4.EMUXD data at beginning of this sample sequence 00b: do not send AS0S4.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.17. AS0R4

Register 17-16. AS0R4 (Auto sequencer 0-sample 4 result register 0x4015 0068)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.18. AS0S5

Register 17-17. AS0S5 (Auto sequencer 0-sample 5 control 0x4015 006C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock

BIT	NAME	ACCESS	RESET	DESCRIPTION
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S5.EMUXD data after S/H of ADC 01b: send AS0S5.EMUXD data at beginning of this sample sequence 00b: do not send AS0S5.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.19. AS0R5

Register 17-18. AS0R5 (Auto sequencer 0-sample 5 result register 0x4015 0070)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.20. AS0S6

Register 17-19. AS0S6 (Auto sequencer 0-sample 6 control 0x4015 0074)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S6.EMUXD data after S/H of ADC 01b: send AS0S6.EMUXD data at beginning of this sample sequence 00b: do not send AS0S6.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.21. AS0R6

Register 17-20. AS0R6 (Auto sequencer 0-sample 6 result register 0x4015 0078)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.22. AS0S7

Register 17-21. AS0S7 (Auto sequencer 0-sample 7 control 0x4015 007C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS0S7.EMUXD data after S/H of ADC 01b: send AS0S7.EMUXD data at beginning of this sample sequence 00b: do not send AS0S7.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.23. AS0R7

Register 17-22. AS0R7 (Auto sequencer 0-sample 7 result register 0x4015 0080)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.24. AS1CTL

Register 17-23. AS1CTL (Auto Sequencer 1 control register 0x4015 0100)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:13	**Reserved**	R	0x0	Reserved
12	**AS1BUSY**	RW	0x0	Auto sequencer 1 busy 1b: auto sequencer 1 sampling active 0b: auto sequencer 1 not active
11	**AS1EN**	RW	0x0	Auto sequencer 1 enable 1b: auto sequencer 1 enabled 0b: auto sequencer 1 not enabled
10:8	**AS1D**	RW	0x0	Auto sequencer 1 sampling depth 111b: 8 samples 110b: 7 samples 101b: 6 samples 100b: 5 samples 011b: 4 samples 010b: 3 samples 001b: 2 samples 000b: 1 sample

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**AS1TR**	RW	0x0	Auto sequencer 1 trigger source 1b: Timer, as defined by AS1CTL.AS1TRTMR 0b: PWM, as defined by AS1CTL.AS1TRPWM
6	**AS1TRE**	RW	0x0	Auto sequencer 1 trigger source AS1CTL.AS1TR edge 1b: high to low edge 0b: low to high edge
5:4	**AS1TRTMR**	RW	0x0	Auto sequencer 1 timer trigger source 11b: Timer D 10b: Timer C 01b: Timer B 00b: Timer A
3:0	**AS1TRPWM**	RW	0x0	Auto sequencer 1 PWM trigger source 1111b: reserved 1110b: reserved 1101b: PWMD1 1100b: PWMD0 1011b: PWMC1 1010b: PWMC0 1001b: PWMB1 1000b: PWMB0 0111b: PWMA7 0110b: PWMA6 0101b: PWMA5 0100b: PWMA4 0011b: PWMA3 0010b: PWMA2 0001b: PWMA1 0000b: PWMA0

17.1.25. AS1S0

Register 17-24. AS1S0 (Auto sequencer 1-sample 0 control 0x4015 0104)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S0.EMUXD data after S/H of ADC 01b: send AS1S0.EMUXD data at beginning of this sample sequence 00b: do not send AS1S0.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.26. AS1R0

Register 17-25. AS1R0 (Auto sequencer 1-sample 0 result register 0x4015 0108)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.27. AS1S1

Register 17-26. AS1S1 (Auto sequencer 1-sample 1 control 0x4015 010C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S1.EMUXD data after S/H of ADC 01b: send AS1S1.EMUXD data at beginning of this sample sequence 00b: do not send AS1S1.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.28. AS1R1

Register 17-27. AS1R1 (Auto sequencer 1-sample 1 result register 0x4015 0110)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.29. AS1S2

Register 17-28. AS1S2 (Auto sequencer 1-sample 2 control 0x4015 0114)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S2.EMUXD data after S/H of ADC 01b: send AS1S2.EMUXD data at beginning of this sample sequence 00b: do not send AS1S2.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.30. AS1R2

Register 17-29. AS1R2 (Auto sequencer 1-sample 2 result register 0x4015 0118)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.31. AS1S3

Register 17-30. AS1S3 (Auto sequencer 1-sample 3 control 0x4015 011C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S3.EMUXD data after S/H of ADC 01b: send AS1S3.EMUXD data at beginning of this sample sequence 00b: do not send AS1S3.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.32. AS1R3

Register 17-31. AS1R3 (Auto sequencer 1-sample 3 result register 0x4015 0120)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.33. AS1S4

Register 17-32. AS1S4 (Auto sequencer 1-sample 4 control 0x4015 0124)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S4.EMUXD data after S/H of ADC 01b: send AS1S4.EMUXD data at beginning of this sample sequence 00b: do not send AS1S4.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.34. AS1R4

Register 17-33. AS1R4 (Auto sequencer 1-sample 4 result register 0x4015 0128)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.35. AS1S5

Register 17-34. AS1S5 (Auto sequencer 1-sample 5 control 0x4015 012C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S5.EMUXD data after S/H of ADC 01b: send AS1S5.EMUXD data at beginning of this sample sequence 00b: do not send AS1S5.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.36. AS1R5

Register 17-35. AS1R5 (Auto sequencer 1-sample 5 result register 0x4015 0130)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.37. AS1S6

Register 17-36. AS1S6 (Auto sequencer 1-sample 6 control 0x4015 0134)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S6.EMUXD data after S/H of ADC 01b: send AS1S6.EMUXD data at beginning of this sample sequence 00b: do not send AS1S6.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.38. AS1R6

Register 17-37. AS1R6 (Auto sequencer 1-sample 6 result register 0x4015 0138)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.1.39. AS1S7

Register 17-38. AS1S7 (Auto sequencer 1-sample 7 control 0x4015 013C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**ADCMUX**	RW	0x0	ADC MUX input select 111b: VSSA 110b: reserved 101b: AD5 100b: AD4 011b: AD3 010b: AD2 001b: reserved 000b: EMUX
11:10	**DELAY**	RW	0x0	Delay between start of sample sequence and start of ADC conversion in ADC input clocks FLCK/ADCCTL.ADCCDIV 11b: 16 ADC input clock cycles 10b: 8 ADC input clock cycles 01b: 4 ADC input clock cycles 00b: 0 ADC input clock
9:8	**EMUXS**	RW	0x0	EMUX transmission start 11b: reserved 10b: send AS1S7.EMUXD data after S/H of ADC 01b: send AS1S7.EMUXD data at beginning of this sample sequence 00b: do not send AS1S7.EMUXD data
7:0	**EMUXD**	RW	0x0	EMUX data to transmit

17.1.40. AS1R7

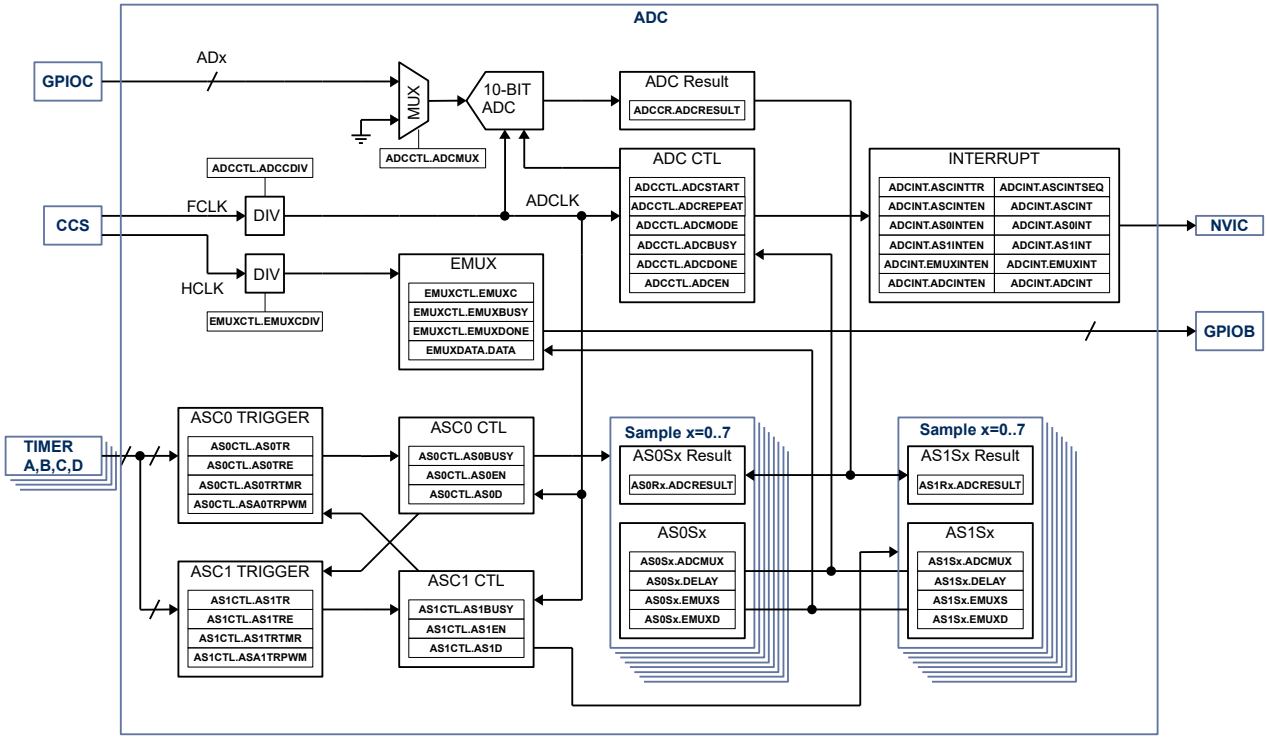
Register 17-39. AS1R7 (Auto sequencer 1-sample 7 result register 0x4015 0140)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	R	0x0	Reserved
9:0	**ADCRESULT**	R	0x0	ADC conversion result

17.2. Details of Operation

17.2.1. Block Diagram

Figure 17-1. ADC, EMUX, ASC0, ASC1



17.3. Details of Operation

17.3.1. Basic Configuration

Following blocks need to be configured for correct operation of the ADC

- CCS
- Timer A, B, C or D
- GPIOB
- GPIOC
- NVIC

17.3.2. ADC, Autosequencer and EMUX

The ADC is a 10-bit SAR ADC. It can be used standalone or together with up to 2 independent low latency auto sequencer state machines to take series of up to 8 samples each into dedicated sample result registers, triggered by either PWM or timer signals. Each sample setup can be programmed with dedicated ADC-MUX setting and settling time delay. A dedicated, programmable high speed low latency communication interface is available to set analog both MUX, sample and hold circuits in the analog peripherals.

17.3.3. Clock Setting

The ADC clock is derived from FCLK and can be set with **ADCCTL.ADCCDIV**. The ADC clock should not

exceed 16MHz for correct operation.

The EMUX clock is derived from HCLK and can be set with **EMUXCTL.EMUXCDIV**.

17.3.4. ADC

The ADC, ASC0, ASC1 and EMXUX block is enabled with **ADCCTL.ADCEN**. In manual conversion mode set **ADCCTL.ADCMODE** to 0b. Set the ADx channel with **ADCCTL.ADCMUX**. For single conversion, set **ADCCTL.REPEAT** to 0b, for repeated conversion set **ADCCTL.REPEAT** to 1b. To start a conversion set **ADCCTL.ADCSTART** to 1b. The ADC will start sampling the analog input channel for 3 clock cycles and holds the analog value in it's internal S/H for conversion. It is safe to switch ADC input channel 4 clocks after ADC start without affecting the ADC result.

One complete ADC conversion will take 16 ADC clock cycles and the ADC conversion result will be available in **ADCCR**. In repeated mode **ADCCR** will be overwritten every 16 ADC clock cycles.. The **ADCCTL.ADCBUSY** flag will 1b as long as conversions are active. **ADCCTL.ADCSTART** will auto clear in single conversion mode. To stop a conversion manually clear **ADCCTL.ADCSTART**.

Figure 17-2. ADC Conversion (Single Shot)

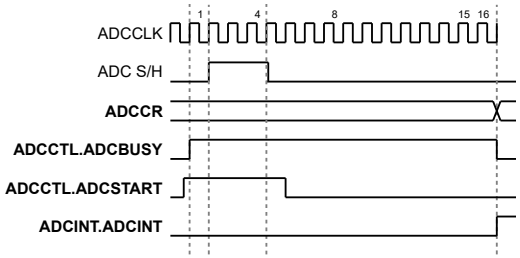
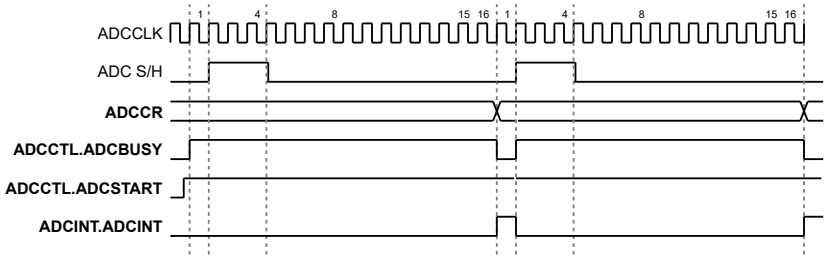


Figure 17-3. ADC Conversion (Repeat Mode)



17.3.5. EMUX

The EMUX is a low latency high speed serial interface with 8-bit data message to control the external ADC MUX and S/H in the analog front end. The EMUX interface is independent from the SOC BUS bridge.

The clock frequency of the EMUX can be adjusted with **EMUXCTL.EMUXCDIV** from HCLK/1 to HCLK/8.

To allow use of EMUX with auto sequencer ASC0 and ASC1, **EMUXCTL.EMUXC** must be set to 1b.

In manual mode with **EMUXCTL.EMUXC** = 0b, the EMUX will start sending the message MSB first as soon as a 8-bit message is written to **EMUXDATA**. While the message is transferred, **EMUXCTL.EMUXDONE** is cleared and is set to 1b when the message transfer is complete.

17.3.6. Auto Sequencer ASC0, ASC1

The ADC and EMUX can be controlled with 2 independent auto sequencer state machines ASC0 and ASC01 to offload the CPU from high speed, low latency sampling. Each sequencer can be programmed to take up to 8 consecutive ADC samples from different analog inputs.

17.3.6.1. Auto Sequencer Modes

The AC0, ASC1 support 8 different modes, configurable with **ADCCTL.ADCMODE**.

With **ADCCTL.ADCMODE** = 000b ASC0 and ASC1 are disabled and the ADC is used in manual mode.

With **ADCCTL.ADCMODE** = 001b only ASC0 is active and manually triggered with **AS0CTL.AS0EN**.

With **ADCCTL.ADCMODE** = 010b only ASC1 is enabled and manually triggered with **AS1CTL.AS1EN**.

With **ADCCTL.ADCMODE** = 011b ASC0 and ASC1 are enabled and daisy chained. When manually triggered with **AS0CTL.AS0EN**. ASC0 with convert all programmed samples, when finished ASC0 will automatically trigger ASC1.

With **ADCCTL.ADCMODE** = 100b only ASC0 is active and triggered with trigger source configured in **AS0CTL.AS0TR**.

With **ADCCTL.ADCMODE** = 101b only ASC1 is active and triggered with trigger source configured in **AS1CTL.AS1TR**.

With **ADCCTL.ADCMODE** = 110b ASC0 and ASC1 are enabled and daisy chained. When triggered with source defined in **AS0CTL.AS0TR**, ASC0 with convert all programmed samples, when finished ASC0 will automatically trigger ASC1.

With **ADCCTL.ADCMODE** = 111b ASC0 and ASC1 are enabled and run independently. ASC0 is triggered with **AS0CTL.AS0TR**, ASC1 is triggered with **AS1CTL.AS1TR**.

Figure 17-4. ASCx, ADCCTL.ADCMODE = 001b, 010b, 100b, 101b

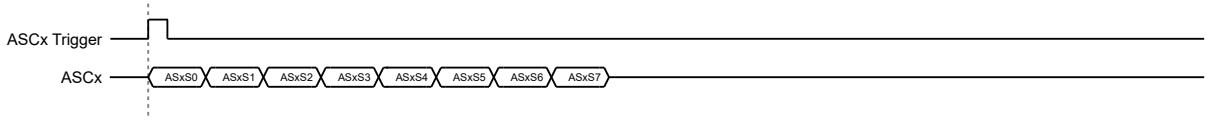


Figure 17-5. ASCx, ADCCTL.ADCMODE = 011b, 110b

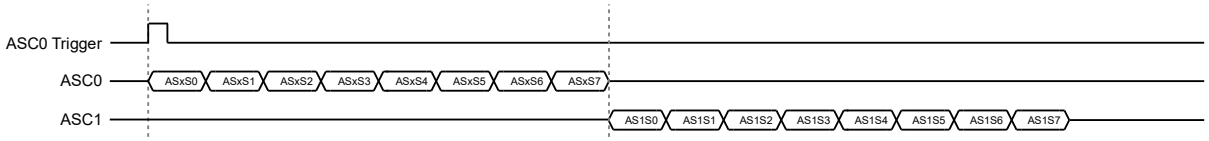
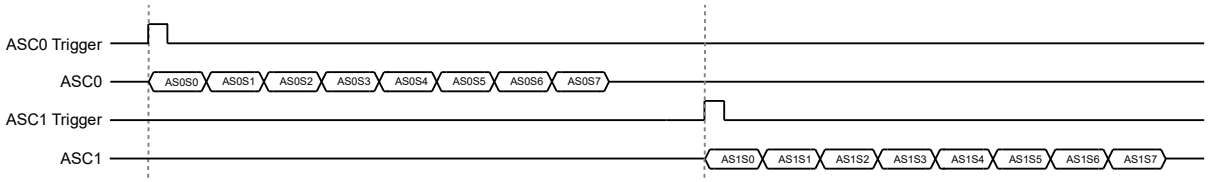


Figure 17-6. ASCx, ADCCTL.ADCMODE = 111b



17.3.6.2. Sequencer trigger

Each sequencer ASC0 and ASC1 can use 2 different trigger modes, manual mode or automated mode.

In automated mode use **ASxCTL.ASxTR** to set the trigger source either to timer A, B, C, or D or PWMAx, PWMBx, PWMCx or PWMDx. Use **AS0xCTL.ASxTRE** to set rising or falling edge trigger. Use **AS0xCTL.ASxTMR** to select timer source or **AS0xCTL.AsxTRPWM** to PWM source.

17.3.6.3. ASC Samples

Each sequencer can be programmed to take 1 to 8 samples up on triggering using **ASxCTL.AsxD**. For each sample, the ADC channel can be programmed with **ASxSy.ADCMUX** , a delay between MUX change and ADC start using **ASxSy.DELAY**, ,a EMUX message to be send with **ASxSy.EMUXD**, and a configuration with **ASxSy.EMUXS** to not send EMUXD, send right after ADCMUX change or send right after start of delay.

NOTE:

Make sure that the EMUX transmission is finished within delay time or ADC conversion time by choosing the correct EMUX clock divider setting.

Figure 17-7. ASxSy Sample with ASxSy.EMUXS = 00b and ASxSy.DELAY = 11b

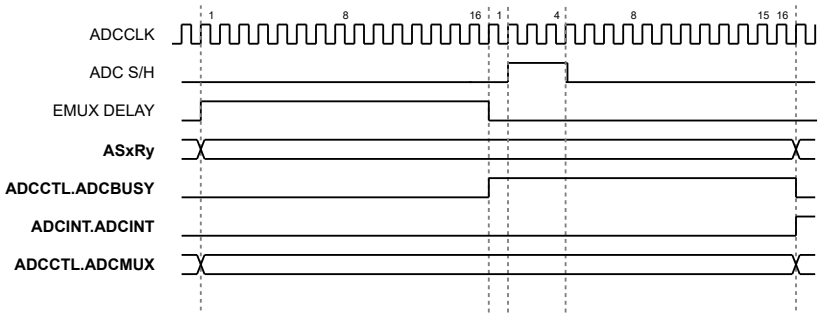


Figure 17-8. ASxSy Sample with ASxSy.EMUXS = 01b and ASxSy.DELAY = 11b

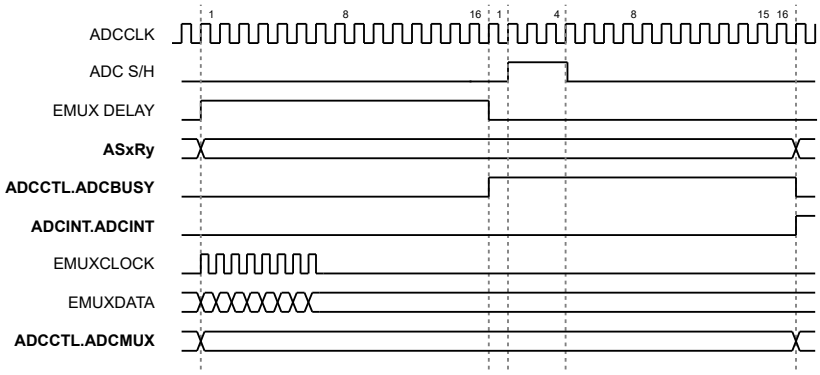
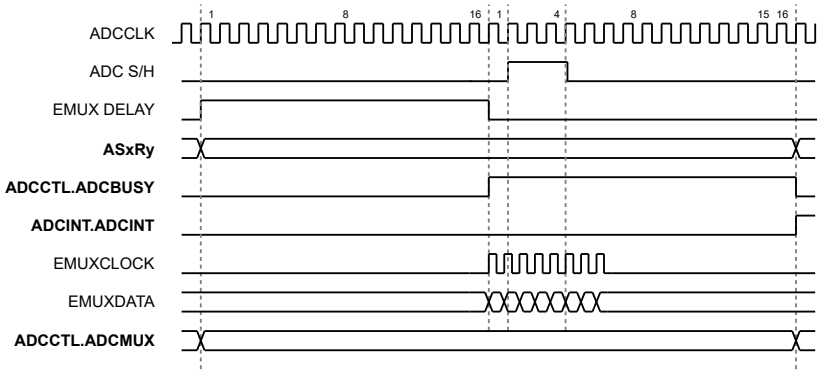


Figure 17-9. ASxSy Sample with ASxSy.EMUXS = 10b and ASxSy.DELAY = 11b



17.3.6.4. ASC0, ASC1 Priority and Collision

In **ADCCTL.ADCMODE** = 100b, 101b, 110b the ASC are triggered with external trigger timer or PWM. Care has to be taken to space the trigger wide enough to allow sequencer ASCx to finish all samples. In case the sequencer trigger event happens before ASC sequencer finishes all samples, the **ADCINT.ASCINT** collision interrupt will be set and the trigger will be ignored. When **ADCINT.ASCINT** is set, **ADCINT.ASCINTSEQ** shows the ASC0 or ASC1 trigger causing the collision interrupt and **ADCINT.ASCINTR** the actual running sequencer.

Figure 17-10. ASCx, 8 samples, No Collision

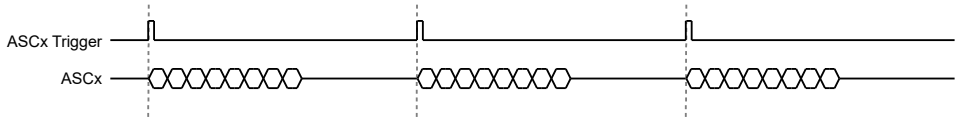
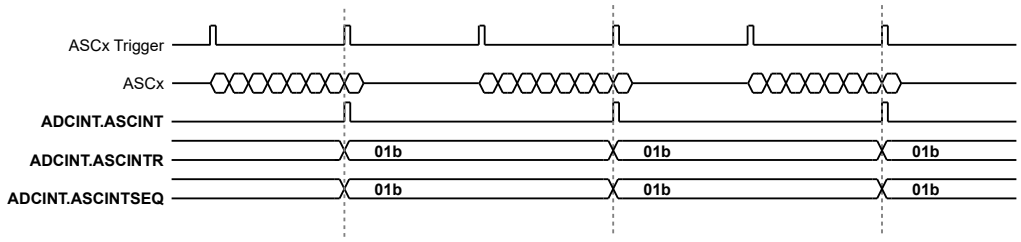


Figure 17-11. ASCx 8 samples, Collision

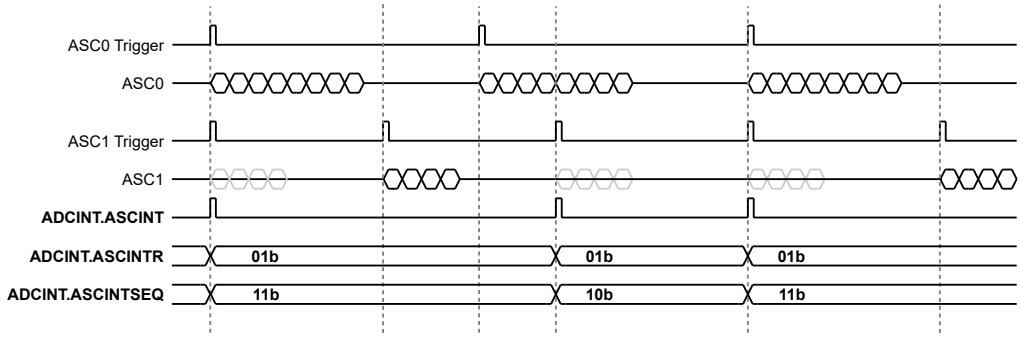


In **ADCCTL.ADCMODE** = 111b, the ASC0 and ASC1 sequencer are triggered independently but are accessing the same ADC.

In case of both ASC0 and ASC1 are triggered at the same time, ASC0 has always higher priority and will be executed while ASC1 is skipped and ignored. **ADCINT.ASCINT** will be set, **ADCINT.ASCINTSEQ** shows the ASC0 or ASC1 trigger causing the collision interrupt and **ADCINT.ASCINTR** the actual running sequencer.

In case of ASC0 or ASC1 sequencer running while the other is triggered, the second sequencer trigger is skipped and ignored, **ADCINT.ASCINT** will be set, **ADCINT.ASCINTSEQ** shows the ASC0 or ASC1 trigger causing the collision interrupt and **ADCINT.ASCINTR** the actual running sequencer.

Figure 17-12. ASC0 8 samples, ASC1 4 samples, Collision



18. I²C

18.1. Register

18.1.1. Register Map

Table 18-1. I²C Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
I²C			
0x401B 0000	**I2CCFG**	I²C configuration	0x0000 0000
0x401B 0004	**I2CSTATUS**	I²C interrupt and status	0x0000 0000
0x401B 0008	**I2CIE**	I²C interrupt enable	0x0000 0000
0x401B 0030	**I2CMCTRL**	I²C master access control	0x0000 0000
0x401B 0034	**I2CMRXDATA**	I²C master receive data	0x0000 0000
0x401B 0038	**I2CMTXDATA**	I²C master transmit data	0x0000 0000
0x401B 0040	**I2CBAUD**	I²C master baud rate	0x01EC 01EC
0x401B 0070	**I2CSRXDATA**	I²C slave receive data	0x0000 0000
0x401B 0074	**I2CSTXDATA**	I²C slave transmit data	0x0000 0000
0x401B 0078	**I2CADDR**	I²C slave address	0x0000 0000

18.1.2. I2CCFG

Register 18-1. I2CCFG (I²C Configuration, 0x401B 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:6	**Reserved**	R	0x0	Reserved
5	**DISPULSEFILT**	RW	0x1	Disable pulse filter 1b: Do not disabled 0b: Enable pulse filter
4	**ADDRMODE**	RW	0x0	Address Mode 1b: 10-bit addressing 0b: 7-bit addressing
3	**Reserved**	RW	0x0	Reserved, must be set to 0b
2	**MAEN**	RW	0x0	Master 1b: I2C Master enable 0b: I2C Master enable
1	**Reserved**	RW	0x0	Reserved, must be set to 0b
0	**SLEN**	RW	0x0	Slave Enable 1b: I2C Slave enable 0b: I2C Slave disable

18.1.3. I2CSTATUS

Register 18-2. I2CSTATUS (I²C Interrupt Status, 0x401B 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:25	**Reserved**	R	0x0	Reserved
24	**SLXFERDONEINT**	R	0x0	Slave Transfer 1b = Slave transfer complete, clears on read 0b = Slave transfer not done

BIT	NAME	ACCESS	RESET	DESCRIPTION
23:19	**Reserved**	R	0x0	Reserved
18	**SLRXFINT**	R	0x0	Slave receive data register SLRXDATA full 1b: SLRXDATA received data from I2C bus, clears on read 0b: SLRXDATA did not receive data since last read of I2CINT
17	**SLTXEINT**	R	0x0	Slave transmit data register SLTXDATA empty 1b: SLTXDATA transmitted to I2C bus, clears on read 0b: SLTXDATA not transmitted since last read of I2CINT
16	**SLADDRMINT**	R	0x0	Slave Address match 1b: Slave address match detected, clears on read 0b: no match
15:12	**Reserved**	R	0x0	Reserved
11	**MADACKINT**	R	0x0	Master data acknowledge 1b: Master data NACK'd, clears on read 0b: Master data ACK'd
10	**MAARBLINT**	R	0x0	Master lost arbitration 1b: Master lost arbitration, clear on read 0b: no error
9	**MAADDRACKINT**	R	0x0	Master address acknowledge 1b: Master address NACK'd, clears on read 0b: Master address ACK'd
8	**MAXFERDONEINT**	RW	0x0	Master transfer complete 1b: Master transfer complete, clears on read 0b: not done
7:3	**Reserved**	R	0x0	Reserved
2	**MARXF**	R	0x0	Master receive data register MARXDATA full 1b: MARXDATA received data from I2C bus, clears on read 0b: MARXDATA did not receive data since last read of I2CINT
1	**MACTLE**	RW	0x0	MACCTL access register accessed 1b: I2CMACTL processed by I2C engine, clears on read 0b: I2CMACTL not accessed by I2C engine since last read of I2CINT
0	**MATXE**	R	0x0	Master transmit data register MATXDATA empty 1b: MATXDATA transmitted to I2C bus, clears on read 0b: MATXDATA not transmitted since last read of I2CINT

18.1.4. I2CIE

Register 18-3. I2CIE (I²C Interrupt Enable, 0x401B 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:25	**Reserved**	R	0x0	Reserved
24	**SLXFERDONEINTEN**	RW	0x0	SLXFERDONE Interrupt enable 1b: interrupt enable 0b: interrupt disabled
23:19	**Reserved**	R	0x0	Reserved
18	**SLRXF**	R	0x0	SLRXF Interrupt enable 1b: interrupt enable 0b: interrupt disabled
17	**SLTXE**	R	0x0	SLTXE Interrupt enable 1b: interrupt enable 0b: interrupt disabled
16	**SLADDRM**	R	0x0	SLADDRM Interrupt enable 1b: interrupt enable 0b: interrupt disabled
15:9	**Reserved**	R	0x0	Reserved
8	**MAXFERDONE**	R	0x0	MAXFERDONE Interrupt enable 1b: interrupt enable 0b: interrupt disabled
7:3	**Reserved**	R	0x0	Reserved
2	**MARXF**	R	0x0	MARXF Interrupt enable 1b: interrupt enable 0b: interrupt disabled
1	**MACTLE**	R	0x0	MACTLE Interrupt enable 1b: interrupt enable 0b: interrupt disabled
0	**MATXE**	R	0x0	MATXE Interrupt enable 1b: interrupt enable 0b: interrupt disabled

18.1.5. I2CMCTRL

Register 18-4. I2CMCTRL (I²C Master Access Control, 0x401B 0030)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	R	0x0	Reserved
13	**I2CMACTLF**	R	0x0	I2CMACTL full 1b: I2CMACTL full, write not allowed, read to clear 0b: I2CMACTL processed, write allowed
12	**Reserved**	R	0x0	Reserved
11	**XFERTYPE**	RW	0x0	Master transfer type 1b: I²C Master Read 0b: I²C Master Write
10	**RSTART**	RW	0x0	Repeated start 1b: No STOP at end of transfer Repeated START 0b: STOP at end of transfer
9:7	**I2CADDRU**	RW	0x0	Upper I²C address bit 9:7
6:0	**I2CADDRL**	RW	0x0	Lower I²C address bit 6:0

18.1.6. I2CMRXDATA

Register 18-5. I2CMRXDATA (I²C Master Receive Data, 0x401B 0034)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
8	**I2CMARXDATAF**	R	0x0	I2CMARXDATA full 1b: I2CMARXDATA register full, clear by read 0b: I2CMARXDATA register empty
7:0	**MARXDATA**	RW	0x0	Master Data Byte received

18.1.7. I2CMTXDATA

Register 18-6. I2CMTXDATA (I²C Master Transmit Data, 0x401B 0038)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
9	**LBYTE**	RW	0x0	Last Byte of Transfer 1b: Last byte of READ or WRITE indicator, initiate STOP after data transfer
8	**I2CMATXDATAF**	R	0x0	I2CMATXDATA full 1b: I2CMATXDATA register full, data not transmitted 0b: I2CMATXDATA register empty
7:0	**MATXDATA**	RW	0x0	Master Data Byte to transmit

18.1.8. I2CBAUD

Register 18-7. I2CBAUD (I²C Baud Rate, 0x401B 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:27	**Reserved**	R	0x0	Reserved
26:16	**SCLH**	RW	0x1EC	Number of HCLK cycles for I2CCL high time
15:11	**Reserved**	R	0x0	Reserved
10:0	**SCLL**	RW	0x1EC	Number of HCLK cycles for I2CCL low time

18.1.9. I2CSLRXDATA

Register 18-8. I2CSLRXDATA (I²C Slave Receive Data, 0x401B 0070)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
8	**I2CSLRXDATAF**	R	0x0	I2CSLRXDATA full 1b: I2CSLRXDATA register full, data not transmitted 0b: I2CSLRXDATA register empty
7:0	**SLRXDATA**	RW	0x0	Slave Data Byte received

18.1.10. I2CSLTXDATA

Register 18-9. I2CSLTXDATA (I^2C Slave Transmit Data, 0x401B 0074)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
9	**I2CSLTXDATAF**	R	0x0	I2CSLTXDATA full 1b: I2CSLTXDATA register full, data not transmitted 0b: I2CSLTXDATA register empty
8	**NACK**	RW	0x0	Slave ACK or NACK 1b:Issue NACK on I^2C Write 0b: Issue ACK on I^2C Write
7:0	**SLTXDATA**	RW	0x0	Slave Data Byte to transmit

18.1.11. I2CADDR

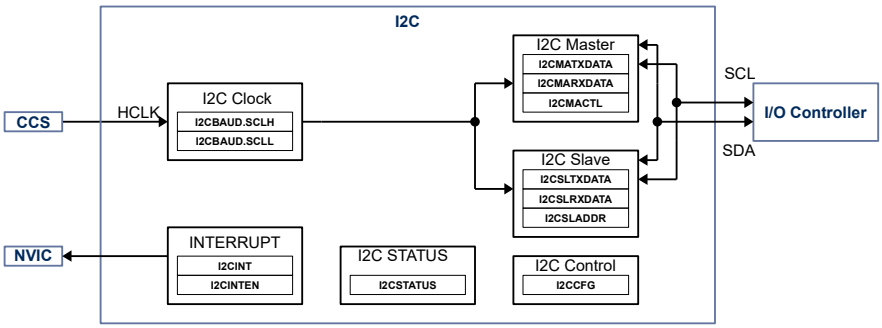
Register 18-10. I2CADDR (I^2C Slave Address, 0x401B 0078)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:10	**Reserved**	RW	0x0	Reserved
9:7	**SLADDRH**	R	0x0	Higher Slave address bit 9:7
6:0	**SLADDRL**	RW	0x0	Lower Slave address bit 6:0

18.2. Details of Operation

18.2.1. Block Diagram

Figure 18-1. I2C



18.2.2. Configuration

Following blocks need to be configured for correct use of the I2C:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller

18.2.3. I2C

The I2C Controller has one master and one slave connected to the same I/O that can be configured to be master only, slave only or concurrent master/slave. The I2C controller supports Normal mode (100kHz), Fast mode (400kHz), and Fast Mode+ (1MHz) operation as well as either 7-bit or 10-bit addressing.

The master supports both single master and multi-master, multi-master sync and multi-master arbitration. The slave supports clock stretching as well.

18.2.4. I2C Clock setting

The I2C SCLK frequency is derived from HCLK, **I2CBAUD.SCLH** sets the SCLK high pulse and **I2CBAUD.SCLL** sets the SCLK low pulse in HCLK cycles and need to be set correctly for different I2C mode.

The minimum HCLK for correct function of the I2C block is: 2.8MHz for normal mode, 3.2MHz for fast mode and 6.14MHz for fast+ mode.

The table below shows pre-calculated **I2CBAUD** settings for normal, fast and fast+ mode with 50MHz HCLK.

Table 18-11. I2CBAUD settings for different HCLK

I2C Mode	SCLK Frequency	HLCK	I2CBAUD.SCLH	I2CBAUD.SCLL
Normal	100kHz	50MHz	0xFA	0xFA
Normal	100kHz	4MHz	0x14	0x14
Normal	100kHz	2.8MHz	0x0E	0x0E
Fast	400kHz	50MHz	0x3E	0x3E
Fast	400kHz	4MHz	0x3E	0x3E
Fast	400kHz	3.2MHz	0x04	0x04
Fast+	1000kHz	50MHz	0x18	0x18
Fast+	1000kHz	6.14MHz	0x03	0x03

18.2.5. I2C Addressing

The I2C address is for I2C master is set in **I2CMCTRL.I2CADDRL** and **I2CMCTRL.I2CADDRH**. The slave address is set **I2CADDR.SLADDRL** and **I2CADDR.SLADDRH**.

18.2.6. I2C Master Read Transactions

The diagram below shows an example of an I2C master read, including which interrupts occur for firmware processing of this transaction.

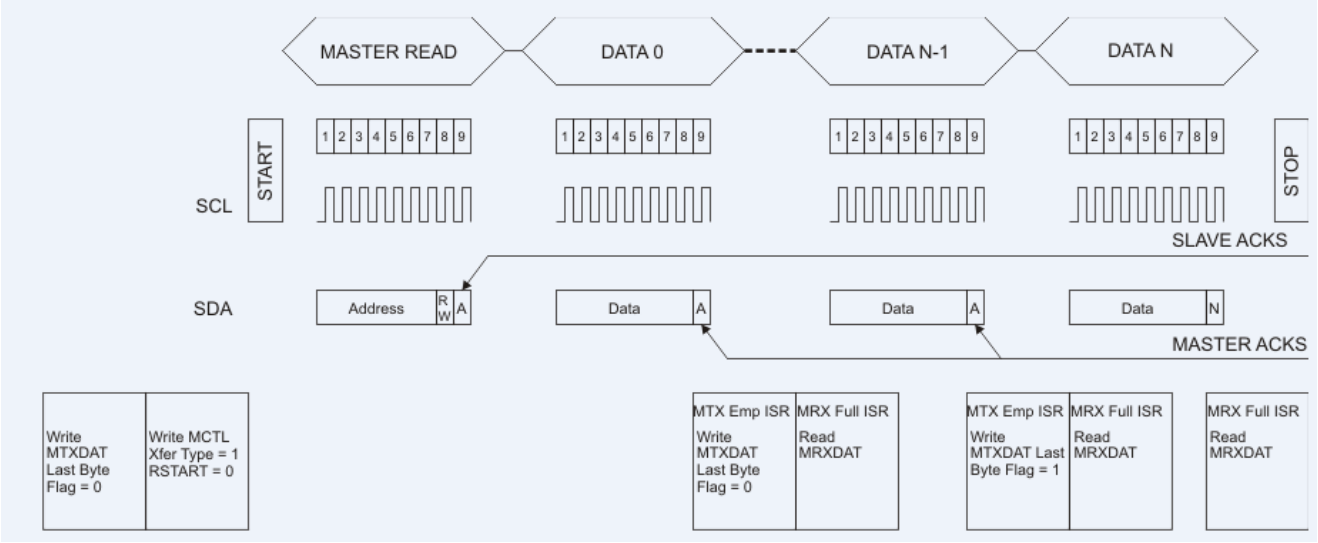


Figure 18-2. I2C Master Read Transaction

A Master Read is initiated when you write to the **I2CMTXDATA** and **I2CMCTRL** register. They need to be written in this order: **I2CMTXDATA** first, then **I2CMCTRL**.

- On the last byte of the transaction, write **MTXDATA** bit 0 to a zero. This tells the system to wait for an ACK from the slave.
- Write **I2CMCTRL** and set **XFERTYPE** to 1 (I2C Master Read), **RSTART** to the desired value (0: No STOP, 1: STOP) and the slave address in **I2CADDRU** and **I2CADDRL**.
- Once **I2CMCTRL** is written, the I2C transfer will begin.

The Master will send the first byte with the slave address and the Read command. The slave will ACK. Immediately after this first ACK, the master will request the first data byte.

When the first data byte is transferred into the Master, the Master will ACK and generate two interrupts: one for Master Transmit Data Register Empty and then one for Master Receive Data Register Full.

- Upon **I2CSTATUS.MTXE** interrupt (master transmit empty), the firmware must write a 1 to the **I2CMTXDATA.LBYTE** flag if there are more than one data byte pending to be received, or a 0 to the **I2CMTXDATA.LBYTE** if the next byte to be received is the last.
- Upon the **I2CSTATUS.MRXF** interrupt (master receive full), the firmware must read the **I2CMRXDATA** register.

Next, repeat until the N-1 data byte is received. On this byte, the firmware must write a 1 to t **I2CMTXDATA.LBYTE.** On the last byte received, the **I2CMTXDATA** must not be written. The **I2CMRXDATA** register still needs to be read.

The waveforms will be similar to the figure below.

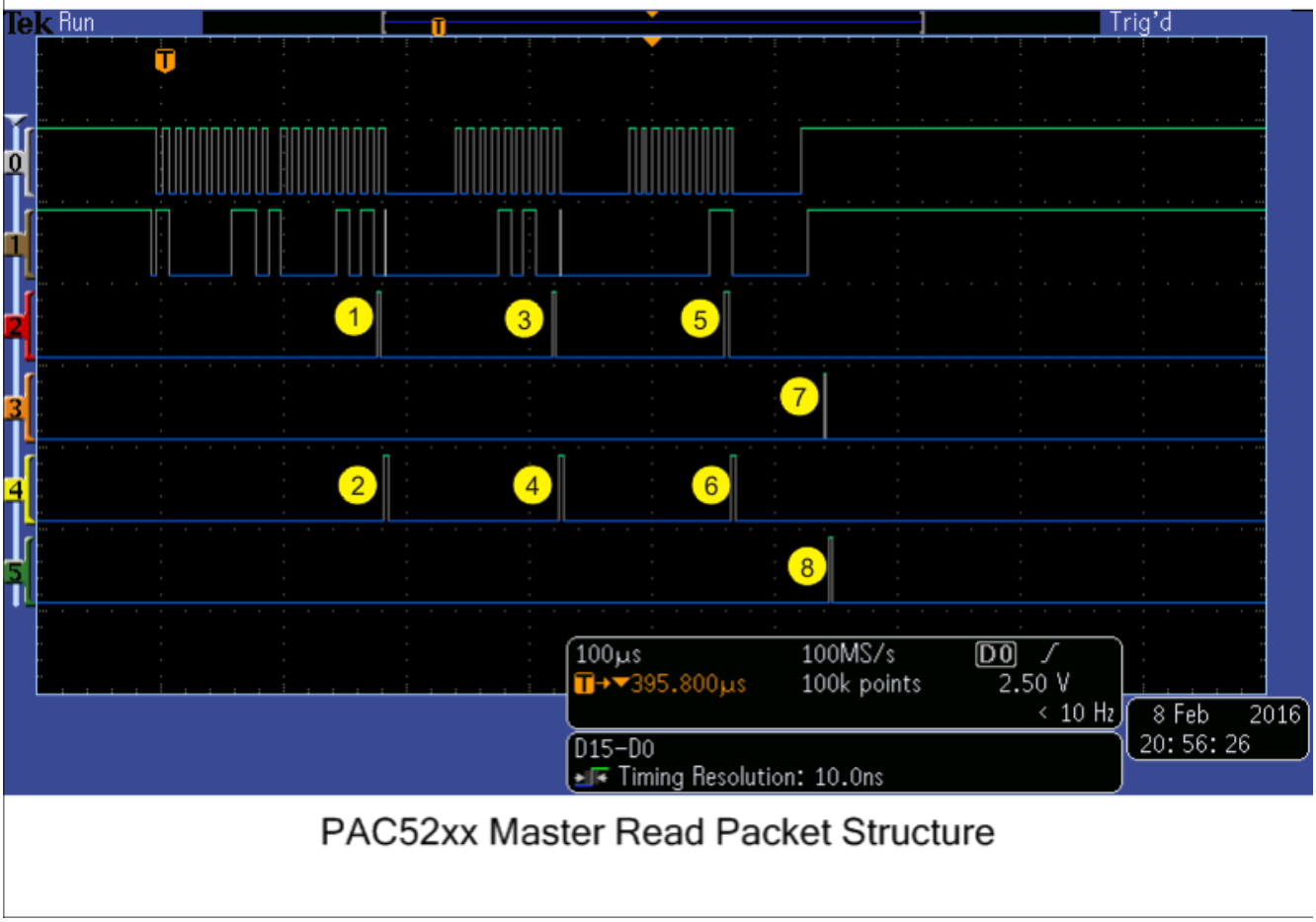


Figure 18-3. I2C Master Read Waveforms

1. First Data Byte **I2CSTATUS.MATXE** interrupt, **I2CMTXDATA.LBYTE** = 0

2. First Data Byte **I2CSTATUS.MRXF** interrupt, read the **I2CMTXDATA** register
3. Second Data Byte **I2CSTATUS.MATXE** interrupt, **I2CMTXDATA.LBYTE** = 1 (as byte #3 will be NACK'd)
4. Second Data Byte **I2CSTATUS.MRXF** interrupt, read the **I2CMTXDATA** register
5. Third Data Byte **I2CSTATUS.MATXE** interrupt, do not write to the **I2CMTXDATA** register
6. Third Data Byte **I2CSTATUS.MRXF** interrupt, read the **I2CMTXDATA** register
7. **I2CMCTRL** Access Register Accessed – can be used for multi-packet communication management.
8. Master Transfer Complete – A STOP has been issued

19. UART

19.1. Register

19.1.1. Register Map

Table 19-1. UART Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
UART			
0x401D 0000	**UARTRXTX**	UART receive/transmit FIFO (available only if **UARTLCR.DLAB** = 0b)	0x0000 0000
	UARTDL_L	UART divisor latch low (available only if **UARTLCR.DLAB** = 1b)	
0x401D 0004	**UARTIER**	UART interrupt enable (available only if **UARTLCR.DLAB** = 0b)	0x0000 0000
	UARTDL_H	UART divisor latch high (available only if **UARTLCB.DLAB** = 1b)	
0x401D 0008	**UARTIIR**	UART interrupt identification (only for register read)	0x0000 0001
	UARTFCTL	UART FIFO control (only for register write)	
0x401D 000C	**UARTLCR**	UART line control	0x0000 0000
0x401D 0010	**UARTMCR**	UART modem control	0x0000 0000
0x401D 0014	**UARTLSR**	UART line status	0x0000 0060
0x401D 0018	**UARTMSR**	UART modem status	0x0000 0000
0x401D 001C	**UARTSP**	UART Scratch Pad	0x0000 0000
0x401D 0020	**UARTFCTL2**	UART FIFO control	0x0000 0000
0x401D 0024	**UARTIER2**	UART interrupt enable	0x0000 0000
0x401D 0028	**UARTDL_L2**	UART divisor latch low byte	0x0000 0000
0x401D 002C	**UARTDL_H2**	UART divisor latch high byte	0x0000 0000
0x401D 0038	**UARTFD_F**	UART fractional divisor value	0x0000 0000
0x401D 003C	**Reserved**	Reserved	0x0000 0000
0x401D 0040	**UARTSTAT**	UART FIFO status	0x0000 0005

19.1.2. UARTRXTX

Register 19-1. UARTRXTX (UART Receive/Transmit FIFO, 0x401D 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Receive and Transmit FIFO buffer on READ: RX FIFO on WRITE: TX FIFO

The **UARTRXTX** register is available when **UARTLCR.DLAB** = 0b.

During a read of **UARTRXTX.VAL**, the head of the FIFO is read. During a write of **UARTRXTX.VAL**, the tail of the FIFO is written with the new data.

19.1.3. UARTDL_L

Register 19-2. UARTDL_L (UART Divisor Latch (low byte), 0x401D 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Divisor value, low byte.

The **UARTDL_L** register is available when **UARTLCR.DLAB** = 1b.

This register allows the user to read or write the low byte of the divisor latch.

19.1.4. UARTIER

Register 19-3. UARTIER (UART Interrupt Enable, 0x401D 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:4	**Reserved**	R	0x0	Reserved
3	**MSI**	RW	0x0	Modem Status interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**RLSI**	RW	0x0	Receive interrupt enable 1b: enable interrupt 0b: disable interrupt
1	**THREI**	RW	0x0	TX register empty interrupt enable 1b: enable interrupt 0b: disable interrupt
0	**RDAI**	RW	0x0	RX register data available interrupt enable 1b: enable interrupt 0b: disable interrupt

The **UARTIER** register is available when **UARTLCR.DLAB** = 0b.

This register allows the user to set the interrupt enable status of the different status conditions of the UART (modem status, receive status, TX register empty and RX register empty).

19.1.5. UARTDL_H

Register 19-4. UARTDL_H (UART Divisor Latch (high byte), 0x401D 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Divisor value, high byte.

The **UARTDL_H** register is available when **UARTLCR.DLAB** = 1b. This register allows the user to read or write the high byte of the divisor latch.

19.1.6. UARTIIR

Register 19-5. UARTIIR (UART Interrupt Identification, 0x401D 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7	**RXFEN**	R	0x0	RX FIFO enable flag 1b: enabled 0b: disabled
6	**TXFEN**	R	0x0	TX FIFO enable flag 1b: enabled 0b: disabled
5:4	**Reserved**	R	0x0	Reserved
3:1	**IID**	R	0x0	UART Interrupt type 111b: reserved 110b: Timeout 101b: reserved 100b: reserved 011b: RX Line Status 010b: RX Data Available 001b: TX Hold register empty 000b: Modem Status
0	**PI**	R	0x1	UART Interrupt 1b: UART interrupt 0b: no UART interrupt

The **UARTIIR** register is available only when the user performs a register read. All fields in this register are read-only.

19.1.7. UARTFCTL

Register 19-6. UARTFCTL (UART FIFO Control, 0x401D 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:6	**RT**	W	0x1	RX FIFO Threshold 11b: 14 Bytes in FIFO 10b: 8 Bytes in FIFO 01b: 4 Bytes in FIFO 00b: 1 Byte in FIFO
5:3	**Reserved**	R	0x0	Reserved
2	**TR**	W	0x0	TX FIFO reset 1b: clear TX FIFO, bit auto clears 0b: no action
1	**RR**	W	0x0	RX FIFO reset 1b: clear RX FIFO, bit auto clears 0b: no action
0	**EN**	W	0x0	FIFO enable 1b: enable RX, TX FIFO 0b: disable RX, TX FIFO

The **UARTFCTL** register is available only when the user performs a register write. All fields in this register are write-only.

19.1.8. UARTLCR

Register 19-7. UARTLCR (UART Line Control, 0x401D 000C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7	**DLAB**	RW	0x0	Divisor Latch Access 1b: Allow access to the divistor latch 0b: Allow access to the FIFOs and IER
6	**SB**	RW	0x0	Break Control 1b: force TX to 0b 0b: normal operation
5	**SP**	RW	0x0	Stick Parity 1b: enable 0b: disable
4	**EPS**	RW	0x0	Parity type 1b: generate EVEN parity 0b: generate ODD parity
3	**PEN**	RW	0x0	Parity Bit 1b: enable Parity 0b: disable Parity
2	**STB**	RW	0x0	Stop Bits 1b: 2 STOP bits (1.5 STOP bits for BPC=00) 0b: 1 STOP bit
1:0	**BPC**	RW	0x0	Bits per Character 11b: 8 bits 10b: 7 bits 01b: 6 bits 00b: 5 bits

19.1.9. UARTMCR

Register 19-8. UARTMCR (UART Modem Control, 0x401D 0010)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:5	**Reserved**	R	0x0	Reserved
4	**LP**	RW	0x0	Loopback 1b: loopback enabled 0b: loopback not enabled
2:0	**Reserved**	RW	0x0	Reserved

19.1.10. UARTLSR

Register 19-9. UARTLSR (UART Line Status, 0x401D 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7	**RFE**	R	0x0	RX FIFO Error 1b: at least 1 parity, framing or break error active in FIFO 0b: no error in RX FIFO
6	**TE**	R	0x1	TX Empty 1b: TX shift register and TX FIFO are empty 0b: not empty
5	**THR**	R	0x1	TX FIFO Empty 1b:TX FIFO are empty 0b: not empty
4	**BI**	R	0x0	RX Break 1b: entry on top of RX FIFO has break error, bit clears on read 0b: error cleared
3	**FE**	R	0x0	RX Framing Error 1b: entry on top of RX FIFO has framing error, bit clears on read 0b: error cleared
2	**PE**	R	0x0	RX Parity Error 1b: entry on top of RX FIFO has parity error, bit clears on read 0b: error cleared
1	**OE**	R	0x0	RX Overrun error 1b: RX FIFO full and last entry overwritten, bit clears on read 0b: error cleared
0	**DR**	R	0x0	RX Data ready 1b: at least 1 entry in RX FIFO 0b: RX FIFO empty

19.1.11. UARTSP

Register 19-10. UARTSP (UART Scratch Pad, 0x401D 001C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	8b scratch pad

19.1.12. UARTFCTL2

Register 19-11. UARTFCTL2 (FIFO Control, 0x401D 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:6	**RT**	RW	0x1	RX FIFO Threshold 11b: 14 Bytes in FIFO 10b: 8 Bytes in FIFO 01b: 4 Bytes in FIFO 00b: 1 Byte in FIFO
5:3	**Reserved**	R	0x0	Reserved
2	**TR**	RW	0x0	TX FIFO reset 1b: clear TX FIFO, bit auto clears 0b: no action
1	**RR**	RW	0x0	RX FIFO reset 1b: clear RX FIFO, bit auto clears 0b: no action
0	**EN**	RW	0x0	FIFO enable 1b: enable RX, TX FIFO 0b: disable RX, TX FIFO

19.1.13. UARTIER2

Register 19-12. UARTIER2 (UART Interrupt Enable, 0x401D 0024)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:4	**Reserved**	R	0x0	Reserved
3	**MSI**	RW	0x0	Modem Status interrupt enable 1b: enable interrupt 0b: disable interrupt
2	**RLSI**	RW	0x0	Receive interrupt enable 1b: enable interrupt 0b: disable interrupt
1	**THREI**	RW	0x0	TX register empty interrupt enable 1b: enable interrupt 0b: disable interrupt
0	**RDAI**	RW	0x0	RX register data available interrupt enable 1b: enable interrupt 0b: disable interrupt

19.1.14. UARTDL_L2

Register 19-13. UARTDL_L2 (UART Divisor Latch Low Byte, 0x401D 0028)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Divisor value, low byte (does not need **DLAP** = 0 in order to work).

19.1.15. UARTDL_H2

Register 19-14. UARTDL_H2 (UART Divisor Latch High Byte, 0x401D 002C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Divisor value, high byte (does not need **DLAP** = 0 in order to work).

19.1.16. UARTFD_F

Register 19-15. UARTFD_F (UART Fractional Divisor Value, 0x401D 0038)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	R	0x0	Reserved
7:0	**VAL**	RW	0x0	Fractional divisor value

19.1.17. UARTSTAT

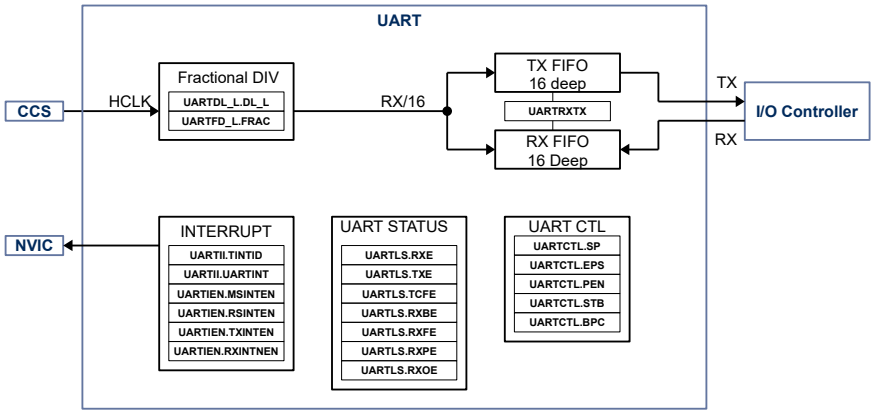
Register 19-16. UARTSTAT (UART FIFO Status, 0x401D 0040)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:4	**Reserved**	R	0x0	Reserved
3	**RXFULL**	R	0x0	RX FIFO full 1b: RX FIFO full 0b: RX FIFO not full
2	**RXEMPTY**	R	0x1	RX FIFO empty 1b: RX FIFO empty 0b: RX FIFO not empty
1	**TXFULL**	R	0x0	TX FIFO full 1b: TX FIFO full 0b: TX FIFO not full
0	**TXEMPTY**	R	0x1	TX FIFO empty 1b: TX FIFO empty 0b: TX FIFO not empty

19.2. Details of Operation

19.2.1. Block Diagram

Figure 19-1. UART



19.2.2. Configuration

Following blocks need to be configured for correct use of the UART:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- I/O Controller

19.2.3. UART

The UART supports up to 3.125 Mbps communication speed, has programmable clock selection with fractional divider, loop back mode for testing, 16 Byte transmit and 16 Byte receive FIFO with programmable receive interrupt threshold.

19.2.4. UART Clock Rate Setting

The UART block has a fractional divider to set up the baud rate. The UART clock is fed by the HCLK. The UART clock must be set to 16x the desired RX TX baud rate setting for correct functioning.

To calculate settings for **UARTDL_H, UARTDL_L, UARTFD_F**, first calculate the desired divider setting using following formula:

$$UARTDivisor = \frac{HCLK}{BAUDRATE * 16} \quad (6)$$

Where:

UARTDivisor: calculated divisor

HCLK: HCLK frequency in Hz

BAUDRATE: desired Baud rate

The integer portion of UART divisor is used to set **UARTDL_H, UARTDL_L**.

To calculate the value of **UARTFD_F**, use formula below and round to the nearest integer.

$$UARTFD = UARTDivisor_{frac} * 256 \quad (7)$$

Where:

UARTFD: calculated UARTFD value

UARTDIVISOR_FRAC: UARTDivisor fractional value

To calculate Baud rate error use following:

$$BAUDRATEERROR = BAUDRATE - (HCLK / (UARTDivisor + UARTDivisor_{frac} / 256) / 16 \quad (8)$$

Where:

BAUDRATEERROR: absolute baud rate error

BAUDRATE: Baudrate

HCLK: HCLK frequency in Hz

UARTDivisor: UART divisor integer value

UARTDivisor_frac: UART divisor fractional value

To calculate relative Baud rate error use following:

$$Relative\ BAUDRATEERROR = BAUDRATEERROR / BAUDRATE * 100 \quad (9)$$

Where:

Relative BAUDRATE ERROR: relative BAUD rate error in %.

BAUDRATEERROR: absolute Baudrateerror

BAUDRATE: desired baudrate setting

The table below shows pre-calculated divisor settings for common Baud rates with 50MHz HCLK.

Register 19-17. UART Divisor Settings for 50 MHz HCLK

Baud Rate	Desired Divisor	Int	frac	UARTDL_H	UARTDL_L	UARTFD_F	Absolute BAUD rate error	BAUD rate error %
300	10416.667	10416	171	0x28	0xB0	0xAB	0.000010	0.0000%
600	5208.333	5208	85	0x14	0x58	0x55	-0.000038	0.0000%
900	3472.222	3472	57	0x0D	0x90	0x39	-0.000058	0.0000%
1200	2604.167	2604	43	0x0A	0x2C	0x2B	0.000154	0.0000%
2400	1302.083	1302	21	0x05	0x16	0x15	-0.000614	0.0000%
4800	651.042	651	11	0x02	0x8B	0x0B	0.002458	0.0001%
9600	325.521	325	133	0x01	0x45	0x85	0.004915	0.0001%
19200	162.760	162	195	0x00	0xA2	0xC3	-0.049152	-0.0003%
38400	81.380	81	97	0x00	0x51	0x61	-0.1	-0.0003%
57600	54.253	54	65	0x00	0x41	0x41	-0.501355	-0.0009%
115200	27.127	27	33	0x00	0x1B	0x21	1.120655	0.0010%

19.2.5. Data settings

The **UARTLCR** register defines the character settings like number of data bits, parity and number of stop bits

19.2.6. FIFO Settings

The FIFO can be configured with the **UARTFCTL** register. FIFO enable, depth and TX/RX FIFO reset can configured. The FIFO status can be monitored in the **UARTLSR** register. A write to **UARTRXTX** writes to the TX FIFO, while a read from **UARTRXTX** reads from RX FIFO.

19.2.7. Error Checking on Received Data

The character specific error does not show up in **UARTLSR** until the character is at the top of the RX FIFO.

Each captured character is checked for following errors

- Break Error **UARTLSR.RXBE** – there was a logic '0' detected on the RX input for more than one character transmission period (BAUD RATE *(1 startbit + **UARTLCR.BPC** data bits + 1 parity bit + **UARTLCR.STB** stopbits)
- Framing Error **UARTLSR.RXFE** – there was a logic '0' detected where there should be a STOP bit
- Parity Error **UARTLSR.PE** – received parity and calculated parity do not match.

20. SOC BUS BRIDGE

20.1. Register

20.1.1. Register Map

Table 20-1. SOC Bus Bridge Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
SOC Bus Bridge			
0x4020 0000	**SOCBCTL**	SOC Bus Bridge control	0x0000 0000
0x4020 0004	**SOCBCFG**	SOC Bus Bridge configuration	0x0000 0200
0x4020 0008	**SOCBCLKDIV**	SOC Bus Bridge clock divider	0x0000 0008
0x4000 000C	**Reserved**	Reserved	0x0000 0000
0x4000 0010	**Reserved**	Reserved	0x0000 0000
0x4020 0014	**SOCBSTAT**	SOC Bus Bridge status	0x0000 0000
0x4020 0018	**SOCBCSSTR**	SOC Bus Bridge Chip Select Steering Register	0x0000 0000
0x4020 001C	**SOCBD**	SOC Bus Bridge data	0x0000 0000
0x4020 0020	**SOCBINT_EN**	SOC Bus Bridge interrupt enable	0x0000 0034

20.1.2. SOCBCTL

Register 20-1. SOCBCTL (SOC Bus Bridge Control, 0x4020 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	R	0x0	Reserved
8	**Reserved**	RW	0x0	Reserved, set to 0x0
7	**Reserved**	RW	0x0	Reserved, set to 0x1
6	**Reserved**	RW	0x0	Reserved, set to 0x1
5	**MTRARM**	W	0x0	MTRANS re-arm Writing a 1b to this bit re-arms the **SOCBCTL.MTRANS** operation by de-asserting the CSx chip select and returning the master mode state machine to IDLE.
4:2	**Reserved**	RW	0x0	Reserved, set to 0x0
1	**SIE**	RW	0x0	SOC Bus Bridge interrupt enable. 1b = Enable the interrupt 0b = Disable the interrupt
0	**SSEN**	RW	0x0	SOC Bus Bridge enable: 1b = Enable this module. 0b = Disable this module.

20.1.3. SOCBCFG

Register 20-2. SOCBCFG (SOC Bus Bridge Configuration, 0x4020 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	R	0x0	Reserved
13	**Reserved**	RW	0x0	Reserved, set to 0x0
12	**Reserved**	RW	0x0	Reserved, be set to 0x0

BIT	NAME	ACCESS	RESET	DESCRIPTION
11	**Reserved**	RW	0x0	Reserved, be set to 0x1
10	**Reserved**	RW	0x0	Reserved, be set to 0x1
9:6	**Reserved**	RW	0x0	Reserved, be set to 0x0
5	**Reserved**	RW	0x0	Reserved, be set to 0x0
4	**Reserved**	RW	0x0	Reserved, be set to 0x0
3	**Reserved**	RW	0x0	Reserved, mbe set to 0x0
2	**MRST**	RW	0x0	Module reset. 1b: Force soft reset of module. The internal state machines are reset; Status register is cleared; However, the soft reset doesn't affect control register values. 0b: do not hold the module in reset.
1:0	**WL**	RW	0x0	Word Length select. 11b: Word length: 32-bits 10b: Word length: 24-bits 01b: Word length: 16-bits 00b: Word length: 8-bits

20.1.4. SOCBCLKDIV

Register 20-3. SOCBCLK (SOC Bus Bridge Clock Divider, 0x4020 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	R	0x0	Reserved
15:0	**CLKDIV**	RW	0x8	Clock divisor for SCLK: SCLK = HCLK / [(CLKDIV+1)*2] the minimum divider is /2

20.1.5. SOCBSTAT

Register 20-4. SOCBSTAT (SOC Bus Bridge Status, 0x4020 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**CURSTATE**	RW	0x0	Raw status of the SOC bus bridge master state machine's "current_state" register. 111b: CSBEGIN 110b: MTRANS 101b: CKWAIT 100b: CSWAIT 011b: CSHOLD 010b: TRANSFER 001b: CSSETUP 000b: IDLE
11	**Reserved**	R	0x0	Reserved
10	**RXFULL**	R	0x0	Raw indicator that the Rx incoming holding register contains a valid data word. 1b: Rx incoming holding register contains a valid data word. 0b: Rx incoming holding register contains no valid data word.
9	**TXFULL**	R	0x0	Raw indicator that the Tx outgoing holding register is still in use, and not ready to accept another data word. 1b: Tx outgoing holding register is still in use not ready to accept another data word. 0b: Tx outgoing register is ready to accept another data word

BIT	NAME	ACCESS	RESET	DESCRIPTION
8	**WRUFL**	RW	0x0	Write Buffer Underflow: set on the start of a second outgoing transfer if data hasn't been written to the **SOCBD** after the previous transfer 1b: Write Underflow detected, clear by writing 1b to it 0b: No Write Underflow since this bit was cleared Note: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable **SOCBINT_EN.WRUFL_EN.**
7:6	**Reserved**	RW	0x0	Reserved, must be set to 0x0
5	**CYC_DONE**	RW	0x0	Cycle Done: this bit will set when the current transfer of 8 bits is complete. It indicates that 8 bits were sent on the transmit port and 8 bits were sampled on the receive port. 1b: Cycle done detected, clear by writing 1b to it 0b: No Cycle Done detected since this bit was cleared Note: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable **SOCBINT_EN.CYC_DONE_EN**.
4:3	**Reserved**	RW	0x0	Reserved, must be set to 0x0
2	**RDOFL**	RW	0x0	Read Buffer Overflow: set on the completion of a second incoming transfer if data hasn't been read from the **SOCBD** from the previous transfer 1b: Read Overflow detected, clear by writing 1b to it 0b: No Read Overflow since this bit was cleared Note: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable **SOCBINT_EN.RDOFL_EN**.
1	**Reserved**	R	0x0	Reserved
0	**SOCB_INT**	R	0x0	SOC bus bridge Interrupt Logical OR of each raw status bit WRUFL, RDOFL, and CYC_DONE, qualified with its corresponding SOCBINT_EN enable. 1b: interrupt 0b: no interrupt Note that if the corresponding SOCBINT_EN of those status bits is reset to '0', those status bits themselves will still assert upon meeting the condition, but will not contribute to the assertion of SOCB_INT. The status bits are true "raw" status bits, and the corresponding SOCBINT_EN simply allows them to cause an interrupt.

20.1.6. SOCBCSSTR

Register 20-5. SOCBCSSTR (SOC Bus Bridge Chip Select Steering, 0x4020 0018)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**Reserved**	RW	0x0	Reserved, must be set to 0x0
23:20	**CSSETUP**	RW	0x0	Chip Select Setup
19:16	**CSHOLD**	RW	0x0	Chip Select Hold
15:12	**CSWAIT**	RW	0x0	Chip Select Wait
11:8	**CKWAIT**	RW	0x0	SOC Bus Bridge Clock Wait
7:0	**Reserved**	RW	0x0	Reserved, must be set to 0x0

20.1.7. SOCBD

Register 20-6. SOCBD (SOC Bus Bridge Data, 0x4020 001C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:8	**Reserved**	RW	0x0	Reserved, must be set to 0x0
7:0	**DATA**	RW	0x0	SOC bus bridge data On READ: retrieve received data word from the incoming holding buffer. On WRITE: write a address or data word to the outgoing holding buffer.

20.1.8. SOCBINT_EN

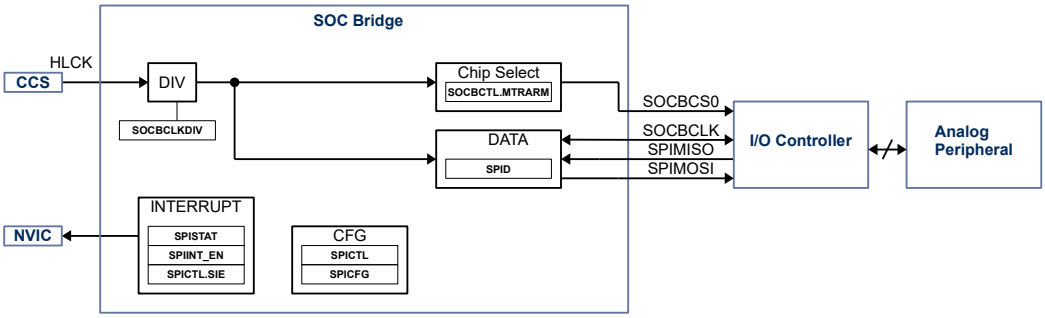
Register 20-7. SOCBINT_EN (SOC Bus Bridge Interrupt Enable, 0x4020 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
8	**WRUFL_EN**	RW	0x0	Write buffer underflow WRUFL interrupt enable 1b: enable **SOCBSTAT.WRUFL** interrupt 0b: disable **SOCBSTAT.WRUFL** interrupt
7:6	**Reserved**	RW	0x0	Reserved, set to 0x0
5	**CYC_DONE**	RW	0x1	Cycle done interrupt enable 1b: enable **SOCBSTAT.CYC_DONE** interrupt 0b: disable **SOCBSTAT.CYC_DONE** interrupt
4:3	**Reserved**	RW	0x1	Reserved, set to 0x0
2	**RDOFL_EN**	RW	0x1	Read buffer overflow RDOFL interrupt enable 1b: enable **SOCBSTAT.RDOFL** interrupt 0b: disable S**OCBSTAT.RDOFL** interrupt
1:0	**Reserved**	RW	0x0	Reserved

20.2. Details of Operation

20.2.1. Block Diagram

Figure 20-1. SOC Bridge



20.2.2. Configuration

Following blocks need to be configured for correct use of the SPI:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller

20.2.3. SOC Bridge

The SOC bridge is used to set and read registers in the analog section of the device.

20.2.4. SOC Bridge Clock Rate Setting

The SOC bridge Module SOCCLK is derived from HCLK to drive the SPI logic, generate setup, hold and wait timings for CSx signals, and SOCCLK.

The SPICLK clock is derived from HCLK using a clock divider configurable with **SOCBCLKDIV**. The lowest clock allowable divider in SPI slave mode is HCLK/8. In SPI Master mode the lowest allowable clock divider is HCLK/2.

To calculate SOCCLK use

$$SOCCLK = \frac{HLCK}{(SOCCLKDIV + 1) * 2} \quad (10)$$

Where:

SOCCLK: SOCCLK in Hz

HCLK: HLCK in Hz

SOCCLKDIV: **SOCBCLKDIV** setting

20.2.5. Enable and Setup of SOC Bridge

20.2.6. SOC Interrupt

The SOC bridge engine interrupt is enabled with **SOCBCTL.SIE**. Then any sub interrupts are enabled in **SINT_EN**.

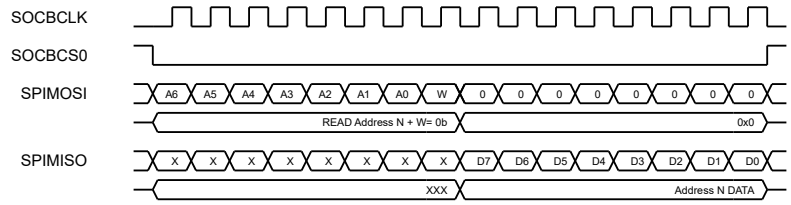
20.2.7. SOC Bridge Protocol

The SOC bridge protocol is a 2 byte protocol, the first byte is the address packet including a 7-bit address [7:1] and a write bit [0], the second packet is an 8bit data packet.

20.2.8. Reading from SOC Bridge

To read to the SOC bridge, start with writing the address packet to **SOCBD** first. Wait for **SOCBSTAT.CYC_DONE** = 1b. Clear **SOCBSTAT.CYC_DONE** by set to 1b. Then write a second dummy address packet to **SOCBD** to clock out the data packet. Wait for **SOCBSTAT.CYC_DONE** = 1b. Clear **SOCBSTAT.CYC_DONE** by set to 1b. Set **SOCBCTL.MTRARM** to 1b to deassert SOCBCS0. Then read from **SOCBD** to get the data packet content

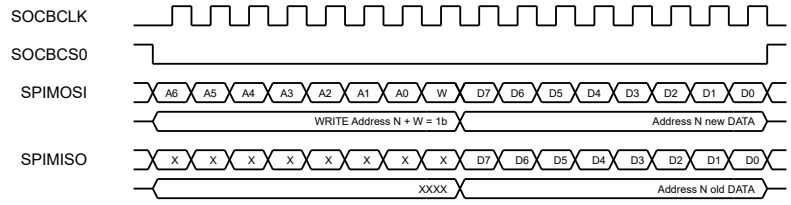
Figure 20-2. Single Read from SOC Bridge



20.2.9. Writing to SOC Bridge

To write to the SOC bridge, start with writing the address packet to **SOCBD** first. Wait for **SOCBSTAT.CYC_DONE** = 1b. Clear **SOCBSTAT.CYC_DONE** by set to 1b. Then write the data packet to **SOCBD** . Wait for **SOCBSTAT.CYC_DONE** = 1b. Clear **SOCBSTAT.CYC_DONE** by set to 1b. Set **SOCBCTL.MTRARM** to 1b to deassert SOCBCS0. Optional you read from **SOCBD** to get the data packet content with old data content of the register address.

Figure 20-3. Single Write to SOC Bridge



21. SPI

21.1. Register

21.1.1. Register Map

Table 21-1. SPI Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
SPI			
0x4021 0000	**SPICTL**	SPI control	0x0000 0000
0x4021 0004	**SPICFG**	SPI configuration	0x0000 0200
0x4021 0008	**SPICLKDIV**	SPI clock divider	0x0000 0008
0x4021 000C	**Reserved**	Reserved	0x0000 0000
0x4021 0010	**Reserved**	Reserved	0x0000 0000
0x4021 0014	**SPISTAT**	SPI status	0x0000 0000
0x4021 0018	**SPICSSTR**	SPI chip select steering	0x0000 0000
0x4021 001C	**SPID**	SPI data	0x0000 0000
0x4021 0020	**SPIINT_EN**	SPI interrupt enable	0x0000 0000

21.1.2. SPICTL

Register 21-1. SPICTL (SPI Control, 0x4021 0000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	R	0x0	Reserved
8	**RTRANS**	RW	0x0	Auto-retransmit on UCLK error. 1b: On a UCLK error, the transmit holding register does NOT get reset, so the word that was transmitting when the UCLK error occurred remains queued. 0b: On a UCLK error, the transmit holding register is reset, and the word that was transmitting when the UCLK error occurred is lost.
7	**MMST_N**	RW	0x0	Multi-master mode (MASTER ONLY) 1b: Single-Master mode, always drive a value onto CSx, SPICLK and MOSI. 0b: Multi-master mode, always tri-state the CSx, SPICLK and MOSI lines when a transfer is complete.
6	**MTRANS**	RW	0x0	Multiple Transfer Mode (MASTER ONLY) 1b: Generate multiple transfers of [SPICFG.WL] bits within a single CSx assertion. 0b: Generate single transfers (assert CSx, transfer data word of [SPICFG.WL] bits, de-assert CSx).
5	**MTRARM**	W	0x0	MTRANS re-arm (MASTER ONLY): Writing a 1b to this bit re-arms the SPICTL.MTRANS operation by de-asserting the CSx chip select and returning the master mode state machine to IDLE.
4	**Reserved**	RW	0x0	Reserved, must be set to 0x0
3	**SE**	RW	0x0	Slave Enable 1b: SPI is a SLAVE. 0b: SPI is a MASTER.

BIT	NAME	ACCESS	RESET	DESCRIPTION
2	**LPBK**	RW	0x0	Internal loop back Mode 1b: Tie the serial out source to the serial in line (internal signaling, does not traverse the chip IO bi-di buffers). 0b = Normal operation.
1	**SIE**	RW	0x0	SPI Interrupt enable. 1b = Enable the interrupt 0b = Disable the interrupt
0	**SSEN**	RW	0x0	SPI enable: 1b = Enable this module. 0b = Disable this module.

21.1.3. SPICFG

Register 21-2. SPICFG (SPI Configuration, 0x4021 0004)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:14	**Reserved**	R	0x0	Reserved
13	**Reserved**	RW	0x0	Reserved, must be set to 0x0
12	**Reserved**	RW	0x0	Reserved, must be set to 0x0
11	**MTURBO**	RW	0x0	Master “turbo” operation mode: 1b: Enable master turbo mode, using HCLK-based bit count, allowing operation down to max 2:1 HCLK:SPICLK ratio 0b: Disable master turbo mode, legacy operation down to max 8:1 HCLK:SPICLK ratio.
10	**TXDBUF**	RW	0x0	Transmit Double-Buffer mode: 1b: enable double-buffer “ping-pong” on shift register transmit output path (keep up with back-to-back words at faster HCLK:SPICLK ratios) 0b: disable double-buffer, legacy operation with a single shift register buffer and single queuing buffer
9	**TXDATPH**	RW	0x0	Early Transmit Data Phase. 1b: Enable: MISO (slave mode) or MOSI(master mode) transitions occur ½ an SPICLK period sooner than the normal protocol (i.e. transition on capture edge instead of launch edge) 0b: Disable: normal transmit data phase, transitions are on the launch edge of SPICLK
8	**Reserved**	RW	0x0	Reserved, must be set to 0x0
7	**RCVCPH**	RW	0x0	SLAVE MODE Clock Phase 1b: Second clock transition of a new transfer is used to sample data 0b: First clock transition of a new transfer is used to sample data
6	**RCVCP**	RW	0x0	SLAVE MODE Clock Polarity 1b: SPICLK is HI in its inactive state 0b: SPICLK is LO in its inactive state
5	**CPH**	RW	0x0	MASTER MODE Clock Phase 1b: Second clock transition of a new transfer is used to sample data 0b: First clock transition of a new transfer is used to sample data
4	**CP**	RW	0x0	MASTER MODE Clock Polarity 1b: SPICLK is HI in its inactive state 0b: SPICLK is LO in its inactive state
3	**LB1ST**	RW	0x0	Least Bit First 1b: LSB is the first serial bit of a transfer 0b: MSB is the first serial bit of a transfer

BIT	NAME	ACCESS	RESET	DESCRIPTION
2	**MRST**	RW	0x0	Module reset. 1b: Force soft reset of module. The internal state machines are reset; Status register is cleared; However, the soft reset doesn't affect control register values. 0b: do not hold the module in reset.
1:0	**WL**	RW	0x0	Word Length select 11b: Word Length = 32-bits 10b: Word Length = 24-bits 01b: Word Length = 16-bits 00b: Word Length = 8-bits

21.1.4. SPICLKDIV

Register 21-3. SPICLKDIV (SPI Clock Divider, 0x4021 0008)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:16	**Reserved**	R	0x0	Reserved
15:0	**CLKDIV**	RW	0x8	Clock divisor for SCLK: SCLK = HCLK / [(CLKDIV+1)*2] In Master/Slave mode with **SPICFG.MTURBO** = 0b, minimum divider is /8. In Master mode with **SPICFG.MTURBO** =1b, minimum divider is /2

21.1.5. SPISTAT

Register 21-4. SPISTAT (SPI Status, 0x4021 0014)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:15	**Reserved**	R	0x0	Reserved
14:12	**CURSTATE**	RW	0x0	Raw status of the SOC bus bridge master state machine's "current_state" register. 111b: CSBEGIN 110b: MTRANS 101b: CKWAIT 100b: CSWAIT 011b: CSHOLD 010b: TRANSFER 001b: CSSETUP 000b: IDLE
11	**Reserved**	R	0x0	Reserved
10	**RXFULL**	R	0x0	Raw indicator that the Rx incoming holding register contains a valid data word. 1b: Rx incoming holding register contains a valid data word. 0b: Rx incoming holding register contains no valid data word.
9	**TXFULL**	R	0x0	Raw indicator that the Tx outgoing holding register is still in use, and not ready to accept another data word. 1b: Tx outgoing holding register is still in use not ready to accept another data word. 0b: Tx outgoing register is ready to accept another data word
8	**WRUFL**	RW	0x0	Write Buffer Underflow Set on the start of a second outgoing transfer if data hasn't been written to the SD register after the previous transfer 1b: Write Underflow detected, clear by writing 1b to it 0b: No Write Underflow since this bit was cleared Note: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.WRUFL_EN.
7	**Reserved**	RW	0x0	Reserved, must be set to 0x0

BIT	NAME	ACCESS	RESET	DESCRIPTION
6	**TE**	RW	0x0	Chip Select Trailing Edge Detect 1b: a chip select de-assertion was detected, clear by writing 1b to it 0b: no chip select de-assertion detected since this bit was cleared NOTE: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.TE_EN.
5	**CYC_DONE**	RW	0x0	Cycle Done: this bit will set when the current transfer of 8 bits is complete. It indicates that 8 bits were sent on the transmit port and 8 bits were sampled on the receive port. 1b: Cycle done detected, clear by writing 1b to it 0b: No Cycle Done detected since this bit was cleared NOTE: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.CYC_DONE_EN.
4	**UCLK**	RW	0x0	Underclock Condition Set if the current transfer received less than word-length “WL” clocks on the SPICLK line prior to CSx de-assertion. 1b: Underclock condition detected, clear by writing 1b to it 0b: No underclock condition detected since this bit was cleared. NOTE: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.UCLK_EN.
3	**LE**	RW	0x0	Chip Select Leading Edge Detect: 1b: a chip select assertion was detected, clear by writing 1 to it 0b: no chip select assertion detected since this bit was cleared NOTE: This bit is cleared by writing a 1b to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.LE_EN.
2	**RDOFL**	RW	0x0	Read Buffer Overflow: set on the completion of a second incoming transfer if data hasn’t been read from the SD register from the previous transfer 1b: Read Overflow detected, clear by writing 1b to it 0b: No Read Overflow since this bit was cleared NOTE: This bit is cleared by writing a ‘1’ to it. This bit is a sticky status bit, and will set upon meeting the condition regardless of the state of its corresponding interrupt enable SPIINT_EN.RDOFL_EN.
1	**Reserved**	R	0x0	Reserved
0	**SPI_INT**	R	0x0	SPI Interrupt Logical OR of each raw status bit WRUFL, RDOFL, and CYC_DONE, qualified with its corresponding INT_EN enable. 1b: interrupt 0b: no interrupt NOTE: that if the corresponding INT_EN of those status bits is reset to ‘0’, those status bits themselves will still assert upon meeting the condition, but will not contribute to the assertion of SPI_INT. The status bits are true “raw” status bits, and the corresponding SPIINT_EN simply allows them to cause an interrupt.

21.1.6. SPICSSTR

Register 21-5. SPICSSTR (SPI Chip Select Steering, 0x4021 0018)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:24	**Reserved**	RW	0x0	Reserved
23:20	**CKWAIT**	RW	0x0	SPI Clock Wait (MASTER mode only): Only applies if SPICTL.MTRANS=1b (multiple transfers with one chip select assertion). This value determines the minimum number of SPICLK periods to wait between back-to-back transfers. During this wait time, SPICLK does not toggle but CSx remains active.
19:16	**CSWAIT**	RW	0x0	Chip Select Wait (MASTER mode only): This value determines the minimum number of SPICLK periods to wait between the de-assertion of CSx and the re-assertion of Csx.
15:12	**CSHOLD**	RW	0x0	Chip Select Hold (MASTER mode only): This value is the minimum number of SPICLK periods to wait from the last SPICLK transition to de-assertion of Csx.
11:8	**CSSETUP**	RW	0x0	Chip Select Setup (MASTER mode only): This value is the minimum number of SPICLK periods to wait from the assertion of CSx to the first SPICLK transition.
7:3	**Reserved**	R	0x0	Reserved
2	**CSL**	RW	0x0	Chip Select active level select (MASTER or SLAVE mode): 1b: active HI outgoing (master) or incoming (slave) chip select 0b: active LO outgoing (master) or incoming (slave) chip select
1:0	**CSNUM**	RW	0x0	Chip Select Number: Outgoing (MASTER mode) - select which one of the possible four chip selects are asserted (the other three are driven de-asserted). Incoming (SLAVE mode) – select which one of the possible four chip selects are actively used. 11b: reserved 10b: SPICS2 01b: SPICS1 00b: SPICS0

21.1.7. SPID

Register 21-6. SPID (SPI Data, 0x4021 001C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:0	**DATA**	RW	0x0	SOC bus bridge data On READ: retrieve received data word from the incoming holding buffer. On WRITE: write a data word to the outgoing holding buffer.

21.1.8. SPIINT_EN

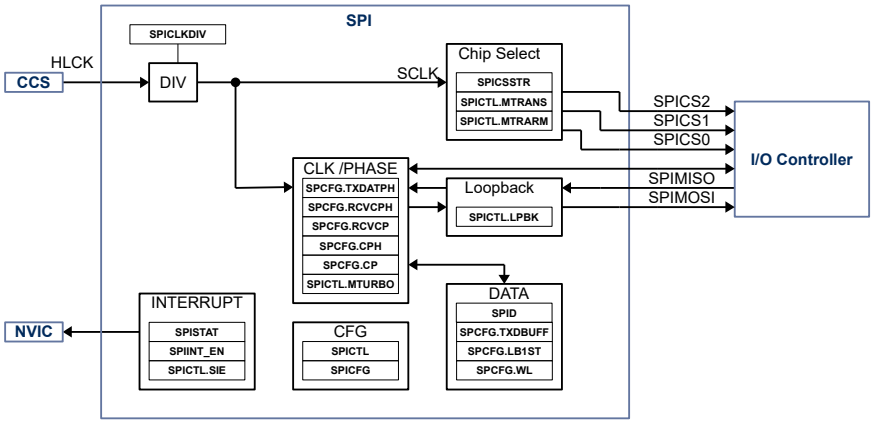
Register 21-7. SPIINT_EN (SPI Interrupt Enable, 0x4021 0020)

BIT	NAME	ACCESS	RESET	DESCRIPTION
31:9	**Reserved**	RW	0x0	Reserved
8	**WRUFL_EN**	RW	0x0	Write buffer underflow WRUFL interrupt enable 1b: enable SPISTAT.WRUFL interrupt 0b: disable SPISTAT.WRUFL interrupt
7	**Reserved**	RW	0x0	Reserved
6	**TE_EN**	RW	0x0	Trailing Edge detect TE interrupt enable 1b: enable SPISTAT.TE interrupt 0b: disable SPISTAT.TE interrupt
5	**CYC_DONE_EN**	RW	0x0	Cycle done CYC_DONE interrupt enable 1b: enable SPISTAT.CYC_DONE interrupt 0b: disable SPISTAT.CYC_DONE interrupt
4	**UCLK_EN**	RW	0x1	Underclock condition detect UCLK interrupt enable 1b: enable SPISTAT.UCLK interrupt 0b: disable SPISTAT.UCLK interrupt
3	**LE_EN**	RW	0x0	Leading Edge detect LE interrupt enable 1b: enable SPISTAT.LE interrupt 0b: disable SPISTAT.LE interrupt
2	**RDOFL_EN**	RW	0x1	Read buffer overflow RDOFL interrupt enable 1b: enable SPISTAT.RDOFL interrupt 0b: disable SPISTAT.RDOFL interrupt
1:0	**Reserved**	RW	0x0	Reserved

21.2. Details of Operation

21.2.1. Block Diagram

Figure 21-1. SPI



21.2.2. Configuration

Following blocks need to be configured for correct use of the SPI:

- Clock Control System (CCS)
- Nested Vectored Interrupt Controller (NVIC)
- IO Controller

21.2.3. SPI

The SPI engine has selectable data byte ordering LSB or MSB first, 4 different data / clock modes, can send / receive packets 8, 16, 32, 64 boundaries, selectable CS polarity, soft reset, and auto-retransmit.

In master mode it supports up to 3 different slaves using chip select. The master mode also allows sending multiple packets per CS.

21.2.4. SPI Clock Rate Setting

The SPI Module SPICLK is derived from HCLK to drive the SPI logic, generate setup, hold and wait timings for CSx signals, and SPICLK.

The SPICLK clock is derived from HCLK using a clock divider configurable with **SPICLKDIV**. The lowest clock allowable divider in SPI slave mode is HCLK/8. In SPI Master mode the lowest allowable clock divider is HCLK/2 if **SPICFG.MTURBO** is 1b, HCLK/8 if **SPICFG.MTURBO** is 0b. **SPICFG.MTURBO** works only in Master mode.

To calculate SCLK use

$$SCLK = \frac{HLCK}{(SPICLKDIV + 1) * 2} \tag{11}$$

Where:

SCLK: SCLK in Hz

HCLK: HLCK in Hz

SPICLKDIV: **SPICLKDIV** setting

21.2.5. Master Slave Mode

The master mode is selected with **SPICTL.SE**= 0b. In master mode a write to **SPID** will initiate SPI data transfer.

When **SPICFG.TXDBUF** is set to 1b, the **SPID** is double buffering is enabled, allowing queuing of the next data word while the current one is transferred.

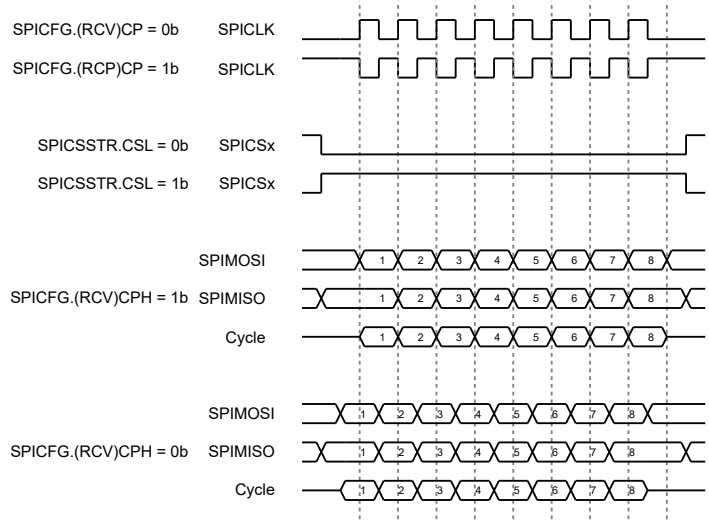
21.2.6. Clock Phase, Polarity

The clock and phase can be programmed independently for master and slave.

The master mode clock polarity is configured with **SPICFG.CP** and the phase is configured with **SPCFG.CPH**.

The slave mode clock polarity is configured with **SPICFG.RCVCP** and the phase is configured with **SPICFG.RCVCPH**.

Figure 21-2. SPI clock polarity and phase



21.2.7. SPI Early Data Transmit

For cases when the SPI is running at high speed the transmitted data can be transmitted 1/2 a SPICLK early to compensate for delays on the receiver side. When **SPICFG.TXDATPH** is 1b, SPIMOSI is transmitted 1/2 SPICLK early in master mode. In slave mode SPIMISO is transmitted 1/2 SPICLK early.

Figure 21-3. SPIMOSI early transmit in master mode

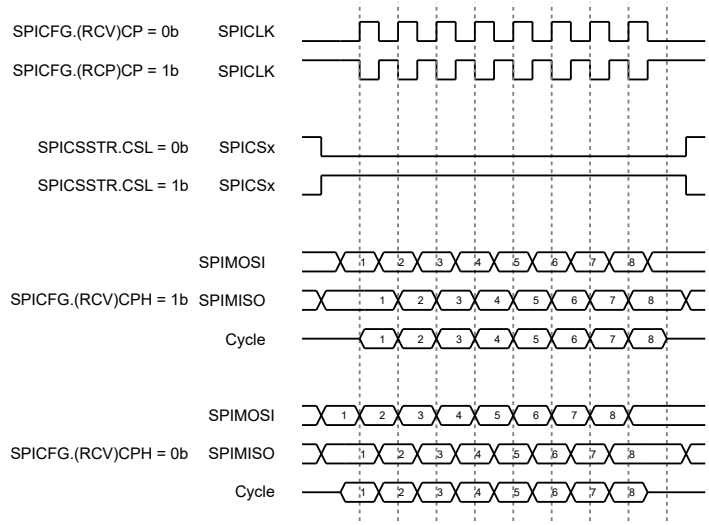
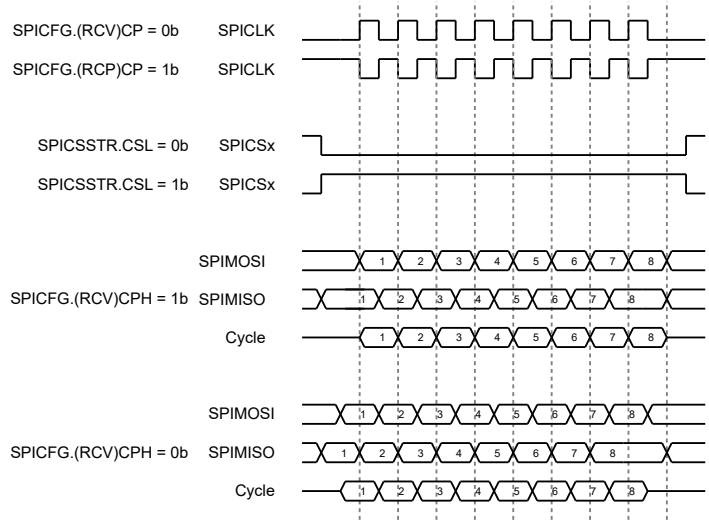


Figure 21-4. SPIMISO early transmit in slave mode



21.2.8. Data Format

The **SPICFG.WL** sets the word length from 8bit to 32bit in 8bit steps.

The **SPICFG.LB1ST** configures the data format to transmit, LSB ior MSB first.

21.2.9. Chip Select Settings

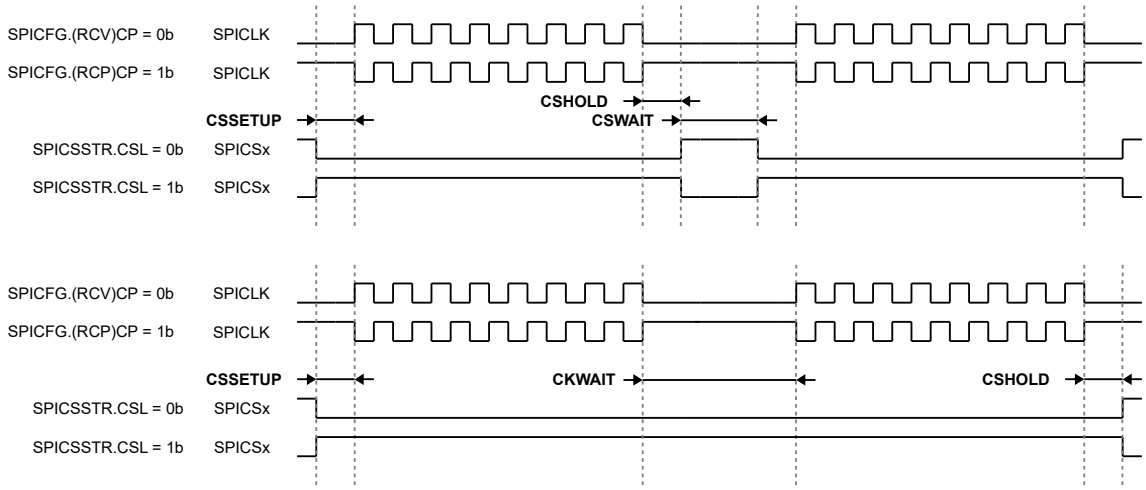
The SPI engine supports up to 3 chip select signals SPICS0, SPICS1 and SPICS2, selectable with **SPICSSTR.CSNUM**.

The CS polarity can be set with **SPICSSTR.CSL**, 0b for active low, 1b for active high.

In SPI engine can automatically assert and deassert SPICSx with each data transaction with **SPICTL.MTRANS** = 0b. To allow multiple data words within a SPICSx assertion set **SPICTL.MTRANS** to 1b. The first data word will assert SPICSx. SPICSx will remain low until it is deasserted with writing **SPICTL.MTARM** to 1b.

Timings for CS behavior can be programmed with granularity of SPICLK. The period between assertion of SPICSx and SPICLK transition can be set with **SPICSSTR.CSSETUP**. The period between last SPICLK transition and deassertion of SPICSxcan be set with **SPICSSTR.CSHOLD**. The period between deassertion and assertion of SPICSx can be set with **SPICSSTR.CSWAIT**. The period between SPICLK transitions between multi word packages can be set with **SPICSSTR.CKWAIT**.

Figure 21-5. SPICSx



21.2.10. Auto Retransmit Data Word

Upon detection of an undercount **SPISTAT.UCLK** error, the default behavior of the SPI module is to reset the SERDES bit counters and the transmit side holding buffers, assuming that software must re-load the word that did not complete transmission because of the UCLK error.

Note that the receive side holding buffers are *not* reset, and contain the data word received from the last *good* transfer.

An optional mode is provided by setting the RTRANS bit in the **SPICTL.RTRANS**. When set, the reset of the transmit side holding buffers because of UCLK error is disabled, and the word that did not complete transmission remains queued for transmit.

21.2.11. Loop Back Mode

The SPI engine has a loop back mode for test purposes, enabled with **SPICTL.LPBK**. The loop back mode is only available in master mode. In loop back mode, SPIMOSI and SPIMISO are connected together internally and SPICLK, SPIMOSI, SPCSx can be still observed on pins.

21.2.12. SPI Interrupt

The SPI engine interrupt is enabled with **SPICTL.SIE**. Then any sub interrupts are enabled in **SPIINT_EN**.

21.2.13. SPI Enable

The SPI engine is enabled with **SPICTL.SSEN**. To soft reset the state machine and clear all status bits use **SPICFG.MRST**.

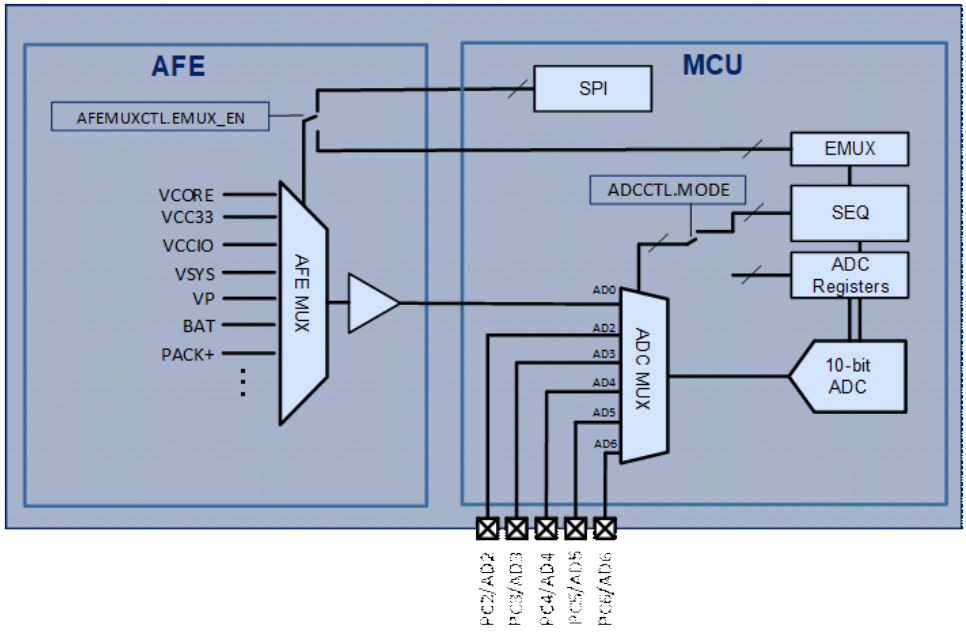
22. ADC MUXES

The device allows the user to internally monitor the various internal and external analog channels through a series of MUXes on the MCU and AFE.

22.1. System Block Diagram

The various MUXes that are used for signal sampling are shown in the diagram below.

Figure 22-1. ADC MUX inputs



There are two main MUXes in the device:

- ADC MUX
- AFE MUX

22.2. ADC MUX

The ADC MUX is an 8-channel MUX local to the ADC on the MCU that is directly controlled either by registers in the MCU, or automatically by the ADC sequencer.

To configure the ADC for manual mode, set **ADCCTL.MODE** to 000b. When the ADC is in manual mode, **ADCCTL** may be used to enable and configure the ADC, including selecting the MUX channel that is used for sampling.

To configure the ADC for sequencer mode, set one of the sequencer modes by setting **ADCCTL.MODE** to 001b to 111b. In one of the sequencer modes, the operation of the ADC and ADC MUX are done automatically in hardware according the sequencer and ADC configuration.

There are 5 external pins and one internal ADC channel that may be configured for ADC analog input that are shown in the table below.

Table 22-1 ADC MUX channels

ADC Channel	MCU I/O PIN	Description
AD0	PC0	Connected to AFE MUX
AD2	PC2	Package pin
AD3	PC3	Package pin
AD4	PC4	Package pin
AD5	PC5	Package pin
AD6	PC6	Package pin

The AD0 channel is always used for analog input from the AFE and is connected to the AFE MUX on MCU internal pin PC0. ADC channels AD<6:2> are directly connected to package pins on the device as shown in the table above.

To use any of these channels as analog inputs to the ADC the IO controller configuration for these pins must be configured as analog input. See the section on the IO controller for more information on IO configuration.

22.3. AFE MUX

The AFE MUX resides in the AFE and is used to select analog signals found in the CAFE.

The MUX select for the AFE MUX may also be controlled directly through the SOC registers or through the EMUX from the MCU's ADC sequencer.

When the ADC is configured for manual mode, the EMUX enable function in the AFE should be disabled. To select the AFE MUX channel using the SOC registers, set **SOC.SHCFG1.EMUXEN** to 0b (disabled). The MUX channel may be selected from **SOC.SHCFG2.MUXA**.

When the ADC is configured for ADC sequencer mode, the EMUX enable function in the AFE should be enabled. To select the AFE MUX channel using the EMUX set **SOC.SHCFG1.EMUXEN** to 1b (enabled). The AFE MUX channel may be selected from the EMUX data sent from the ADC sequencer.

The channels available on the AFE MUX are shown in the table below.

Table 22-2 AFE MUX channels

AFE MUX Channel	Value	Description
VCORE	0	VCORE
VCORE	1	VCORE / 2.5
VDDA	2	VDDA / 2.5
VCCIO	3	VCCIO / 2.5
VSYS	4	VSYS / 2.5
ISENSE	5	Current Sense Diff Amp Output
VPTAT	6	Internal temperature sensor (VPTAT).
VP	7	VP / 10
VREF	8	VREF / 2
FUSE	9	FUSE / 10
CHG	10	CHG / 50
DSG	11	DSG / 50
BAT	12	BAT / 50
AIO0A	13	Analog Input/Output 0 Amp Output
LOADDET	14	Load Detection Voltage
SCPDAC	15	SCP DAC Voltage
OCCDAC	16	OCC DAC Voltage
OCDDAC	17	OCD DAC Voltage
BATOVDAC	18	BAT Over Voltage DAC
VIN	19	VIN / 50
PACK+	20	PACK+ / 50
VCP	21	VCP / 50

For more information on the configuration and use of the EMUX, see the section below.

23. EMUX

The EMUX is a dedicated high-speed, low-latency serial interface to control the AFE MUX using the ADC sequencing engine.

To enable the EMUX, set the **SOC.AFEMUXCTL.EMUX_EN** = 1b. This will enable the ADC sequencer to command the control of the AFE MUX using the EMUX data sent by the sequencer.

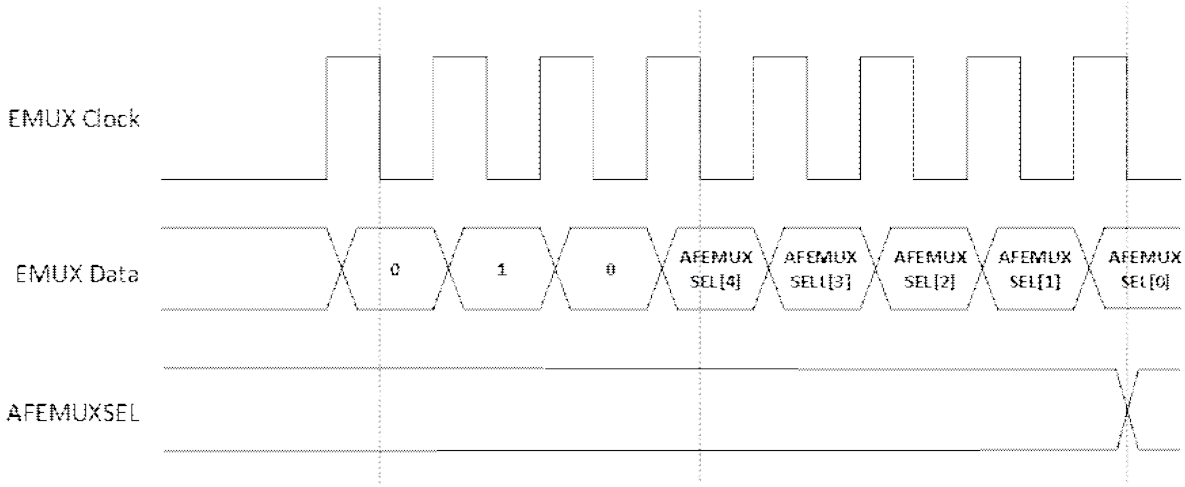
The format of the EMUX command used to control the AFE MUX is shown below. The EMUX data is transmitted MSb first.

Register 23-1. EMUX Packet Structure

BIT	NAME	DESCRIPTION
7	**BIT7**	Bit 7 should be set to 0b
6	**BIT6**	Bit 6 should be set to 1b
5	**BIT5**	Bit 5 should be set to 0b
4:0	**AFEMUXSEL**	AFE MUX Channel Selector: 0 = VCORE 1 = VCORE / 2.5 2 = VDDA / 2.5 3 = VCCIO / 2.5 4 = VSYS / 2.5 5 = ISENSE 6 = VPTAT 7 = VP / 10 8 = VREF / 2 9 = FUSE / 10 10 = CHG / 50 11 = DSG / 50 12 = BAT / 50 13 = AIO0A 14 = LOADDET 15 = SCPDAC 16 = OCCDAC 17 = OCDDAC 18 = BATOVDAC 19 = VIN / 50 20 = PACK+ / 50 21 = VCP / 50

The EMUX data is written on this bus MSb first from the ADC sequencer. See the timing diagram below.

Figure 23-1. EMUX Timing Diagram



Bits [7:5] should contain 010b so that the EMUX controller will recognize the EMUX data as valid.

At the 8th EMUX clock falling edge, the AFE will read the AFEMUXSEL**[4:0]** data. At this time the AFE will set the AFE MUX to the proper channel, according to this data.

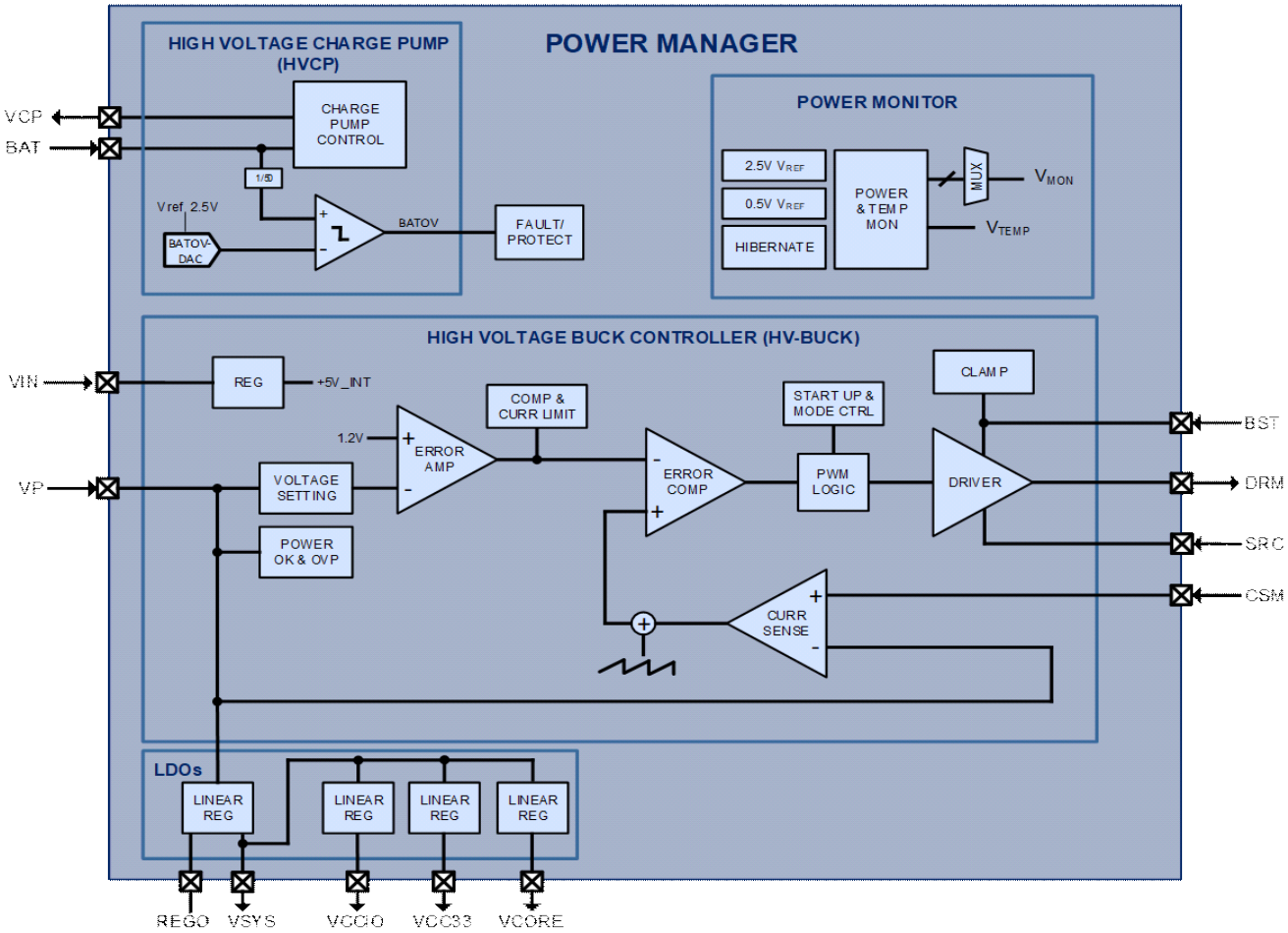
24. CONFIGURABLE POWER MANAGER

24.1. Features

- High Voltage BUCK
- High Voltage Charge Pump for CHG/DSG high-side gate driver supply
- 4 additional Linear regulators with power and hibernate management
- High-accuracy voltage reference for ADC and comparators
- Power and temperature monitor, warning, fault detection
- Extremely low hibernate mode I_Q of 3µA at 80V

24.2. System Block Diagram

Figure 24-1. Configurable Power Manager Block Diagram



24.3. Functional Description

The Configurable Power Manager is optimized to efficiently provide “all-in-one” power management required by the PAC22140 and associated application circuitry. It incorporates a high-voltage power supply controller that is used to convert power from either the battery stack or charger input to generate a main supply output VP.

It incorporates a High-Voltage Charge Pump (HVCP) to efficiently convert power from the battery stack to generate a gate driver VCP.

There are also linear regulators to provide VSYS, VCC33 and VCORE supplies for 5V system, 3.3V mixed signal and micro-controller core circuitry. The power manager also handles system functions including internal reference generation, timers, hibernate mode management, and power and temperature monitoring.

24.4. Hibernate Mode

Hibernate mode on the device allows a very low I_Q mode when not in operation to minimize energy consumption when the motor is not running, and the battery pack need not provide power.

To enter hibernate mode, configure the **SOC.HIBCTL** register settings, and the set **SOC.HIBENTER.HIB** to 1b. This bit will be automatically cleared when exiting hibernate.

To wake-up from hibernate mode the user may either use the Hibernate Wake-up Timer, the Push-button function or PACK+ detection. Before entering hibernate mode, one of these methods should be configured.

24.4.1. Push Button

Before entering hibernate mode by setting **SOC.HIBENTER.HIB** to 1b, the system could be configured to exit hibernate mode with a properly polarized push button assertion, by setting the **SOC.HIBENTER.PBWAKEEN** bit to 1d.

The Push Button wake up method is selected by configuring the **SOC.HIBENTER.WAKESRC** to 0d.

Further Push Button functionality is detailed in section 30.3.5

24.4.2. Pack+ Wake Up

Before entering hibernate mode by setting **SOC.HIBENTER.HIB** to 1b, the system could be configured to exit hibernation mode by detecting the moment in which the battery pack is connected to a charger, or a load, by setting the **SOC.HIBENTER.HIB** to 1b.

The Push Button wake up method is selected by configuring the **SOC.HIBENTER.WAKESRC** to 1d.

24.4.3. Wake Up Timer

Before entering hibernate mode by setting **SOC.HIBENTER.HIB** to 1b, the system could be configured to periodically exit hibernate mode by configuring the **SOC.HIBENTER.WUTIMER** from 0d to 7d. Refer to Section 34.1.6 for more information on available wake up timing periods. A configuration setting equal to 0d will in effect disable the Wake Up Timer. Settings from 1d to 7d will enable the Wake Up Timer with periods ranging from as low as 125 ms to as large as 8 seconds.

The Wake Up Timer method is selected by configuring the **SOC.HIBENTER.WAKESRC** to 2d.

24.5. Power Manager Faults

The power manager monitors all of the power supplies and LDOs for faults during operation.

During a power management fault condition such as under-voltage or over-current each of the power supplies will set a fault bit and certain power supplies can be disabled as a result of the fault. The device may be reset and power rails restarted. To determine the fault condition, fault registers can be read. The corresponding fault bit(s) should be written to a 1b to clear the fault flag.

24.6. Temperature Warnings and Faults

The PAC22140 has an integrated temperature sensor that is used for temperature warnings and faults and can also be sampled by the MCU ADC through the AFE MUX using the VPTAT MUX channel.

This value has a compensation coefficient available in INFO FLASH that can be used to obtain an accurate temperature. The parameter VT300K will be stored in INFO FLASH and will indicate the compensation factor.

The die temperature in degrees Kelvin can then be calculated by the following formula:

TKELVIN = 300 * (VPTAT + 0.075) / (VT300K + 0.075)

The PAC22140 contains two warning thresholds and one fault threshold:

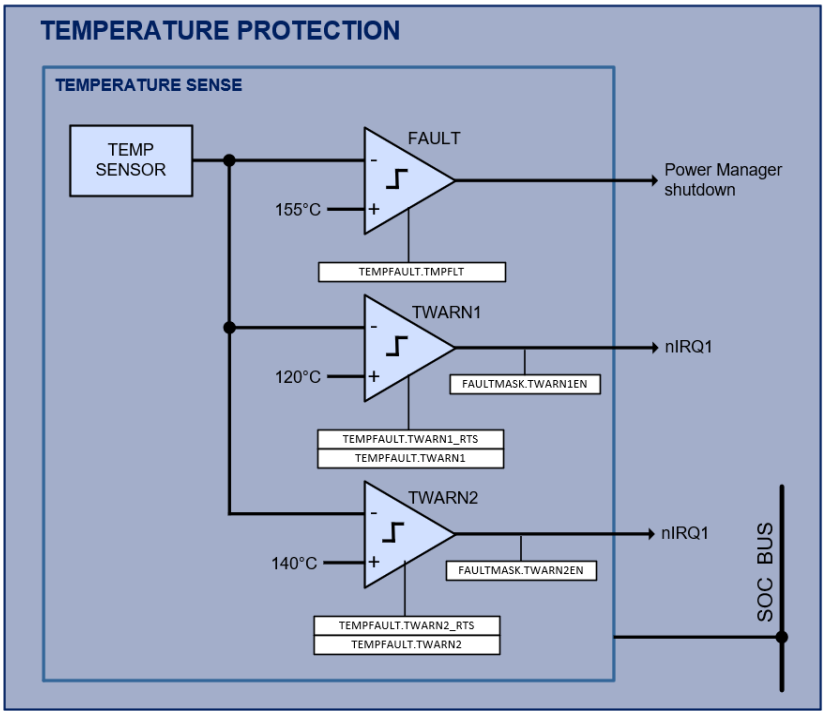
- Warn 1: 120°C
- Warn 2: 140°C
- Fault: 155°C

If the internal temperature (VPTAT) rises above the Warn 1 threshold, the device will indicate this through a latched register bit. The user may enable a mask-able interrupt to the MCU to announce this condition.

If the internal temperature (VPTAT) rises above the Warn 2 threshold, the device will indicate this through a separate latched register bit. The user may enable an interrupt to the MCU to announce this condition. This will be the same interrupt as the Warn 1 threshold.

If the internal temperature (VPTAT) rises above the Fault threshold, the Charge pump, DC/DC and gate drivers will be disabled. The device will indicate this through a latched register bit. There is not interrupt for this condition. When the device falls below the hysteresis value, then the DC/DC will be re-enabled.

The temperature hysteresis level for all three thresholds is 10°C.



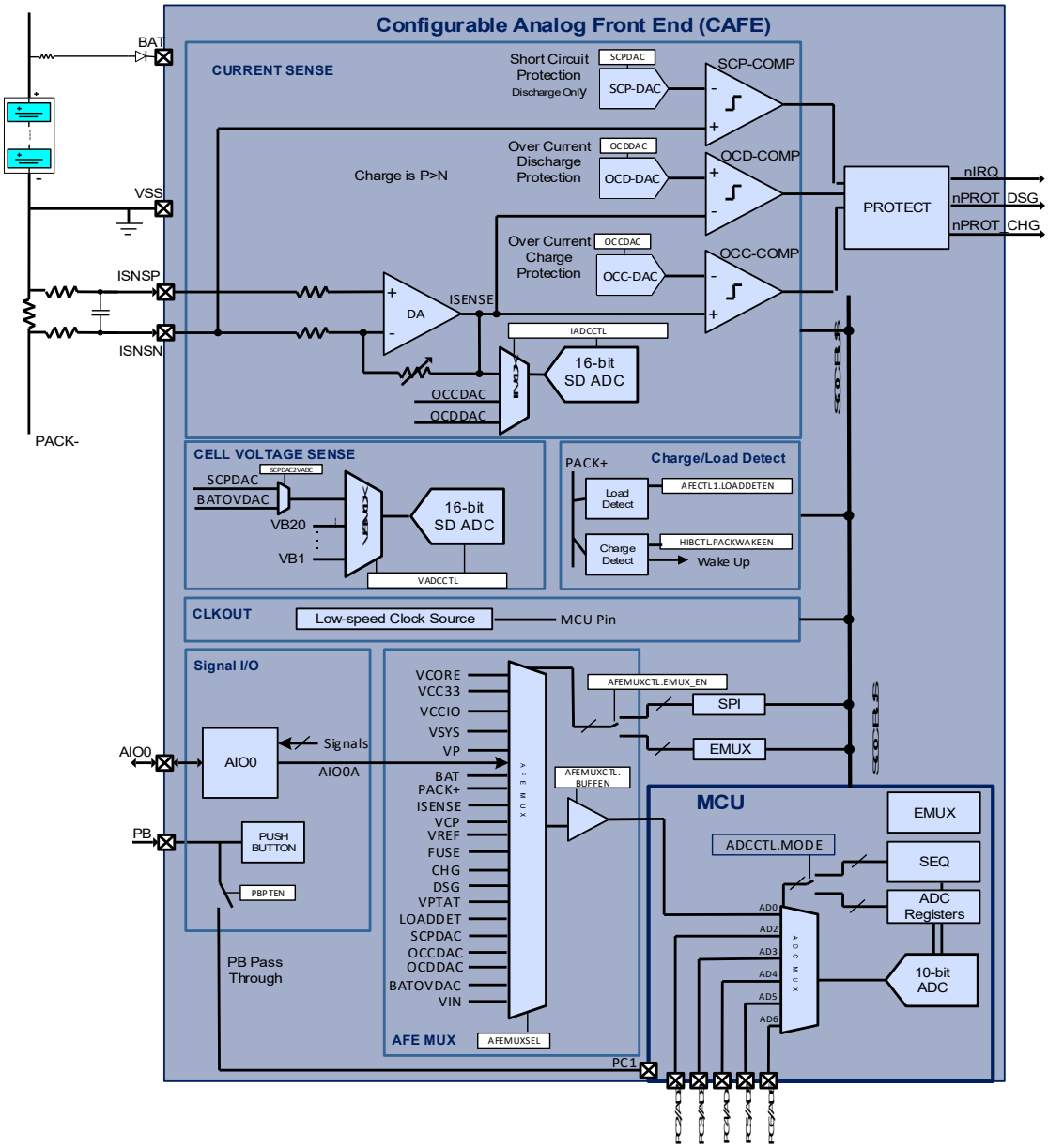
25. CONFIGURABLE ANALOG FRONT END

25.1. Features

- 16-bit Sigma-Delta ADC for current sense
- Current Sense Differential Programmable-Gain Amplifier
- Analog input/output gain amplifier with ADC Input or signal output selection
- Three comparators with DACs for programmable references for configurable OC warning or fault handling
- 16-bit Sigma-Delta ADC for Voltage Sampling of up to 20 Cell Voltages
- 10-bit Successive Approximation Register ADC in MCU for Temperature sense and internal power supplies
- Internal Temperature Warnings and Fault Protec
- Push-Button (PB) input for exiting hibernate mode
- PACK+ Charge detection and Load detection
- 2kHz independent clock source

25.2. Block Diagram

Figure 25-1. Configurable Analog Front End



25.3. Functional Description

The PAC22140 contains a Configurable Analog Front-End (CAFETM). The CAFE can be used to sense battery pack current using an integrated Differential Amplifier and 16-bit Sigma-Delta ADC. The CAFE can configure three independent DACs to set three comparator thresholds for over-current detection for the battery pack.

The CAFE can also sense cell voltages via a 16-bit Sigma Delta ADC and MUX which contains inputs for each of the cell balance channels. Internal power supply rails, temperature, and other signals can be selected using the AFE Mux and sampled using the MCU 10-bit SAR ADC.

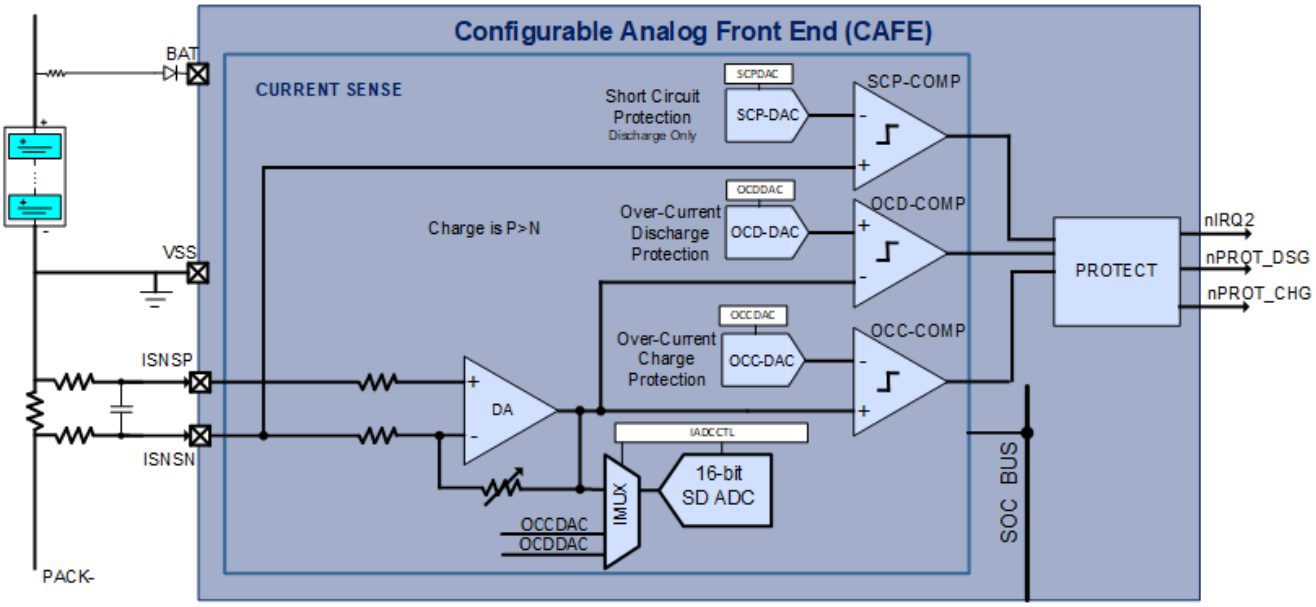
25.3.1. Register Protection

Select registers require the protection address SOC.PROT_KEY be written before the select registers can be written. Registers that require this will contain a note about writing PROT_KEY. To enable writing of the protected registers, write SOC.PROT_KEY.KEY with 0xA5, and then write the protected register.

25.3.2. Current Sensing

The PAC22140 contains circuitry for battery pack current sense. The positive terminal, I sense positive (ISNSP) and negative terminal, I sense negative(ISNSN) are connected to each side of an external sense resistor. The ISNSP pin is connected to the positive terminal of a differential amplifier (DA) and the ISNSN pin is connected to the negative terminal of the amplifier. The differential amplifier has a programmable gain up to x128. This amplifier has a common-mode range of -0.3V to 0.5V.

The output of the differential amplifier is connected to the 16-bit Sigma-Delta ADC for current sensing and also to over current protection comparators.



25.3.2.1. IADC 16-bit Sigma-Delta ADC

The IADC is a 16-bit Sigma-Delta ADC and has an input range of -500mV to +500mV. The IADC is controlled using SOC.IADCCTL register. Write the SOC.IADCCTL register to set the Diff Amp gain, select the IMUX setting, and start the IADC conversion. To know if the conversion is complete, the SOC.IADCCTL.IADCBUSY bit can be polled. The IADCBUSY bit signal is also routed to the MCU PA3 GPIO, which can be configured to generate an interrupt based on the IADCBUSY signal.

25.3.2.2. Differential Amplifier Gain

Use SOC.IADCCTL.DAGAIN to set to gain between 1x to 128x.

25.3.2.3. Differential Amplifier Reference

The Differential Amplifier reference is 0.5V.

25.3.2.4. Measuring Current SENSE voltages

The Current SENSE ADC (IADC) is enabled by setting the SOC.SIGMGRCTL2.IADCEN bit.

In order to measure the battery pack current through the SENSE resistor, the IADC multiplexer must be configured to select the differential output by writing a 0d to the SOC.IADCTL.IMUXSEL register. Alternatively, the same multiplexer allows the voltage conversion of the SCP DAC, OCC DAC and OCD DAC.

To start a IADC Conversion, the SOC.IADCCTL.ADCSTART bit must be set to 1d. The read only bit SOC.VADCCTL.IADCBUSY bit will set to 1d once the conversion is completed. The 16 bit conversion can be obtained by reading the SOC.IADCRESHI and SOC.IADCRESLO registers.

Reading the current SENSE value can be achieved concurrently while the cell voltage ADC (VADC) operates and convert independent cell voltage values.

25.3.3. Over-Current Protection

There are three protection comparators that may be used for over-current protection: one for short circuit protection (SCP-COMP), one for over-current discharge protection (OCD-COMP) and one for over-current charge protection (OCC-COMP) . Each of the comparators has a DAC that may be used to set the comparator reference.

When one of the comparators trips, the device will send a signal to the driver manager and can be programmed in various ways to disable the CHG/DSG FETs, as well as interrupt the MCU via an IRQ signal. Determining which source asserted the IRQ signal can be achieved by reading the SOC.SIGFAULT register.

25.3.3.1. Short- Circuit Protection

Short-Circuit protection (SCP) is designed to disable gate drivers if the battery pack suddenly is discharging a large amount of current. The SCP is implemented with the SCP DAC (0.5V Vref) and SCP comparator (0.5V Vref) to compare the DAC setting to the ISNSN signal.. If the comparator threshold is met, the comparator will set the SOC.SIGFAULT.SCPFLT bit. An active SCP Fault can be configured to perform the following:

1) disable the CHG gate driver via the SOC.PROTEN.SCPCPROTEN bit,
2) disable the DSG gate driver via the SOC.PROTEN.SCPDPROTEN bit.
3) Interrupt the MCU via the nIRQ2 signal connected to the PB7 GPIO signal if the SOC.SIGFAULTEN.SCPFAULTEN bit is set.

The SCP DAC value can be set by writing the SOC.SCPDDAC register. The SCP comparator is enabled by setting the SOC.SIGMGRCTL1.SCPEN bit. The SCP comparator blanking time and hysteresis can be configured by writing the SOC.SCPCFG register. Once the SCP DAC value is exceeded at the SCP Comparator input, a short circuit event will be registered and the SOC.SIGFAULT.SCPFAULT flag will be set. The SOC.SIGFAULT.SCPFAULT flag is cleared by writing a 1d.

Short circuit protection comparator output can be polled in real time by reading the SOC.BATRTS.SCP_RTS bit.

25.3.3.2. Over-Current Charge Protection

Over-Current Charge protection (OCC) is designed to is implemented with the OCC DAC (0.5V Vref) and OCC

comparator (0.5V Vref) to compare the DAC setting to the differential amplifier output signal.. If the comparator threshold is met, the comparator will set the SOC.SIGFAULT.OCCFLT bit. An active OCC Fault can be configured to perform the following:

4) disable the CHG gate driver via the SOC.PROTEN.OCCPPROTEN bit,
5) Interrupt the MCU via the nIRQ2 signal connected to the PB7 GPIO signal if the SOC.SIGFAULTEN.OCCFAULTEN bit is set.

The OCC DAC value can be set by writing the SOC.OCCDAC register. The OCC comparator is enabled by setting the SOC.SIGMGRCTL1.OCCEN bit. The OCC comparator blanking time and hysteresis can be configured by writing the SOC.OCCCFG register. Once the OCC DAC value is exceeded at the OCC Comparator input, an Over Current Charge event will be registered and the SOC.SIGFAULT.OCCFAULT flag will be set. The SOC.SIGFAULT.OCCFAULT flag is cleared by writing a 1d.

Over current charge protection comparator output can be polled in real time by reading the SOC.BATRTS.OCC_RTS bit.

25.3.3.3. Over-Current Discharge Protection

Over-Current Discharge protection (OCD) is designed to is implemented with the OCD DAC (0.5V Vref) and OCD comparator (0.5V Vref) to compare the DAC setting to the differential amplifier output signal.. If the comparator threshold is met, the comparator will set the SOC.SIGFAULT.OCDFLT bit. An active OCD Fault can be configured to perform the following:

6) disable the CHG gate driver via the SOC.PROTEN.OCDPPROTEN bit,
7) Interrupt the MCU via the nIRQ2 signal connected to the PB7 GPIO signal if the SOC.SIGFAULTEN.OCDFAULTEN bit is set.

The OCD DAC value can be set by writing the SOC.OCDDAC register. The OCD comparator is enabled by setting the SOC.SIGMGRCTL1.OCDEN bit. The OCD comparator blanking time and hysteresis can be configured by writing the SOC.OCDCFG register. Once the OCD DAC value is exceeded at the OCD Comparator input, an Over Current Discharge event will be registered and the SOC.SIGFAULT.OCDFAULT flag will be set. The SOC.SIGFAULT.OCDFAULT flag is cleared by writing a 1d.Over current discharge protection comparator output can be polled in real time by reading the SOC.BATRTS.OCD_RTS bit.

25.3.4. Battery Over-Voltage Protection

Battery Over Voltage protection (BATOV) is designed to is implemented with the BATOV DAC (2.5V Vref) and BATOV comparator (2.5V Vref) to compare the DAC setting to the scaled down version (1/50) version of the battery pack's PACK+ terminal. If the comparator threshold is met, the comparator will set the SOC.SIGFAULT.BATOVFLT bit. An active BATOV Fault can be configured to perform the following:

8) disable the CHG gate driver via the SOC.PROTEN.BATOVCPPROTEN bit,
9) disable the DSG gate driver via the SOC.PROTEN.BATOVDCPPROTEN bit.
10) Interrupt the MCU via the nIRQ2 signal connected to the PB7 GPIO signal if the SOC.SIGFAULTEN.BATOVCPROTEN bit is set.
11) Interrupt the MCU via the nIRQ2 signal connected to the PB7 GPIO signal if the SOC.SIGFAULTEN.BATOVDPROTEN bit is set.

The BATOV DAC value can be set by writing the SOC.BATOVDAC register. The BATOV comparator is enabled by setting the SOC.SIGMGRCTL1.BATOVEN bit. The BATOV comparator blanking time and hysteresis can be configured by writing the SOC.BATOVCFG register. Once the BATOV DAC value is exceeded at the BATOV

Comparator input, an Over Voltage event will be registered and the SOC.SIGFAULT.BATOVFAULT flag will be set. The SOC.SIGFAULT.BATOVFAULT flag is cleared by writing a 1d.

Battery over voltage protection comparator output can be polled in real time by reading the SOC.BATRTS.BATOV_RTS bit.

25.3.5. Voltage Sensing

The PAC22140 also contains a 16-bit Sigma-Delta ADC that may be used for voltage sensing for the individual cells. There is a MUX that selects each of the individual cell balance nodes (VB1 to VB20) so that they may be sampled by the 16-bit ADC.

The SOC bus is used for the MUX select as well as the ADC operation and fetching of the 16-bit result from the MCU.

25.3.5.1. Measuring Independent Cell Voltages

The cell voltage ADC (VADC) is enabled by setting the SOC.SIGMGRCTL2.VADCEN bit. Cells for which voltage ADC capturing will be performed, must also be enabled by setting their respective enable bit. This can be accomplished by writing to the SOC.CELLEN1.CENx, SOC.CELLEN2.CENx and SOC.CELLEN3.CENx registers. Cells which are not enabled will be removed from the ADC path and can't be digitized.

In order to measure each independent voltage, the VADC multiplexer must be configured to select the cell which will be connected into the VADC converter. The VBMUXSEL value can be configured by writing a number from 0d to 19d to the SOC.VADCTL.VBMUX register. Alternatively, the same multiplexer allows the voltage conversion of the Battery Over Voltage DAC by selecting multiplexer entry 20d.

To start a VADC Conversion, the SOC.VADCCTL.ADCSTART bit must be set to 1d. The read only bit SOC.VADCCTL.VADCBUSY bit will set to 1d once the conversion is completed. The IADCBUSY bit signal is also routed to the MCU PA3 GPIO, which can be configured to generate an interrupt based on the IADCBUSY signal. The 16 bit Cell Voltage conversion can be obtained by reading the SOC.VADCRESHI and SOC.VADCRESLO registers.

Reading cell voltages can be achieved concurrently while the current ADC operates and convert current values.

25.3.6. AFE MUX

The CAFE also contains the AFE MUX that can be used to sample the internal power supply rails on the device, internal die temperature, and other signals.

The AFE MUX select can be controlled from the SOC bus or via the EMUX. The output of the AFE MUX is connected to the 10-bit ADC on the MCU so it may be sampled by the auto-sequencer.

The MUX channels that are available are:

- VCORE – 1.9V Core Logic LDO
- VCC33 – 3.3V Analog LDO
- VCCIO – 3.3V Digital I/O LDO
- VSYS – 5V System Supply
- VP – DC/DC Output
- AIO0A – AIO0 Analog Output
- ISENSE – Current Sense PGA Analog Output
- BAT – Battery Stack Input
- PACK+ - Charger Supply Input
- VCP – Charge Pump Output
- VREF – 2.5V Voltage reference
- FUSE – FUSE output
- CHG – CHG FET gate driver signal
- DSG – DSG FET gate driver signal
- VPTAT – Internal Temperature Sensor
- LOADDET – Load monitoring signal
- SCPDAC – Short Circuit Protection DAC output
- OCCDAC – Over Current Charge DAC output
- OCDDAC – Over Current Discharge DAC output
- VIN – Main Input Supply Voltage

25.3.7. Enabling the CAFE

The CAFE will not respond to signals from the MCU other than the SOC Bridge until the MCU sets **SOC.AFECTL1.MCUALIVE** to 1b. Also, before the CAFE sub-system can be begin any signal conditioning, it must be enabled. To enable the CAFE set **SOC.AFECTL1.SIGEN** to 1b.

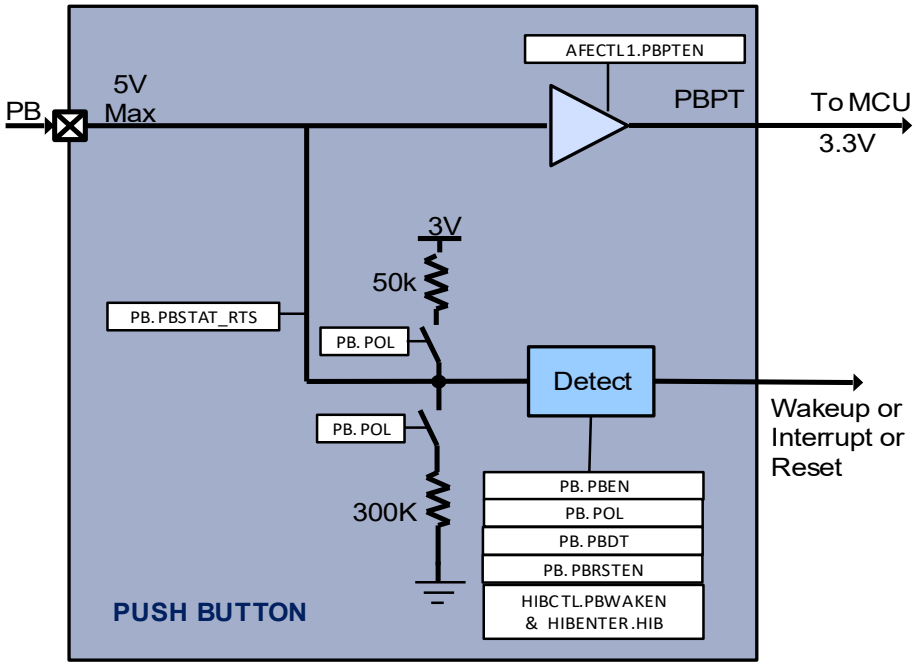
25.3.8. Push-Button (PB) Input

The PAC22140 contains a push-button input pin (PB) and a push-button module (see Figure 7 below). A push-button pass through to the MCU can be enabled so that the MCU can read the state of the digital PB pin directly on the PC1 GPIO.

The push-button module can be used to wake-up the device from hibernate, interrupt the MCU, or reset the device. The push-button polarity can be set to active high or active low using the polarity setting.

The push-button module can be configured so that an active PB will wake up the PAC22140 from hibernate mode.

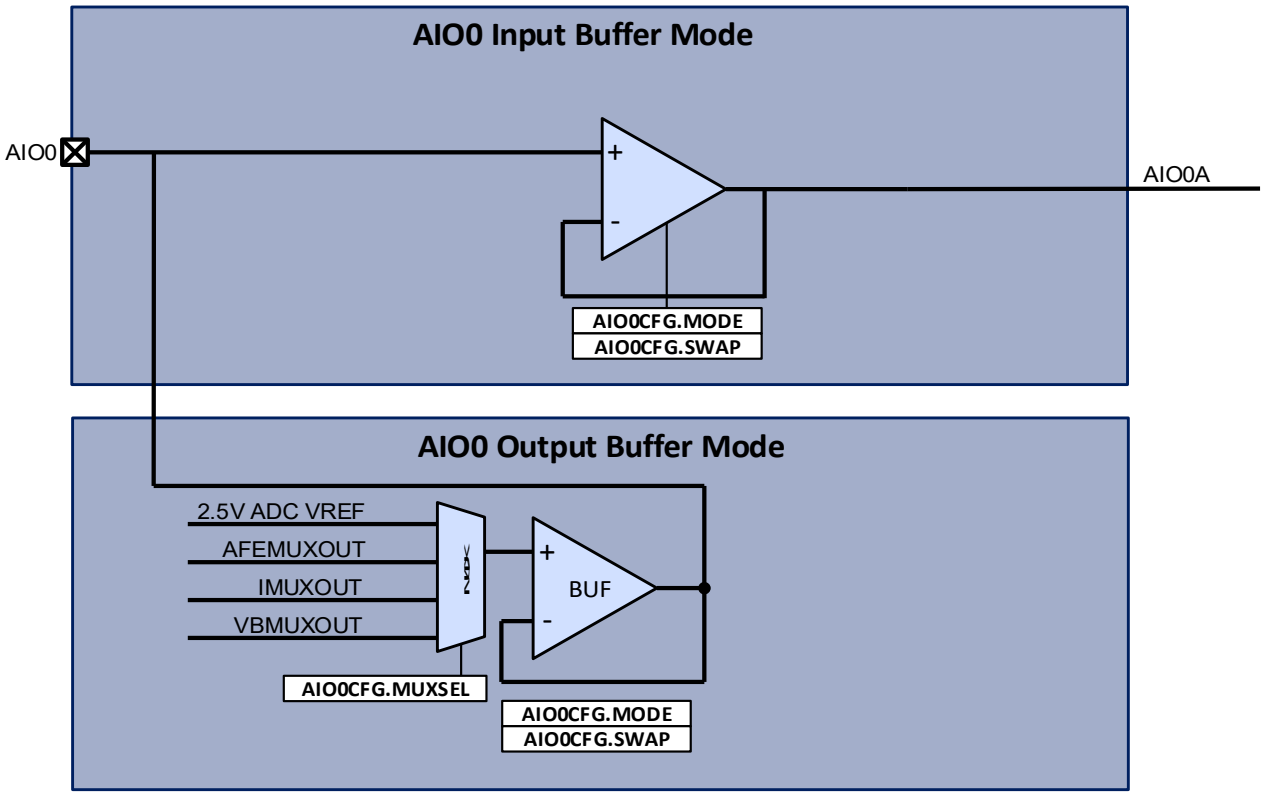
When enabled, the push-button may also be used as a hardware reset when held active for longer than 8s during normal operation. Once the push-button is enabled, the polarity setting will determine whether the PB input will be pulled up to 3V with a 50kΩ resistor or pulled down to GND with a 300kΩ resistor. There is also a programmable de-glitch time that may be configured to 1, 4, 8, or 32ms.



25.4. Analog I/O 0 (AIO0)

The PAC22140 has an AIO0 module and pin that includes a gain amplifier for analog I/O. The AIO0 pin can be configured as an input to the amplifier with the output of the gain amplifier, AIO0A, routed to the AFE Mux for input to the MCU ADC. Or, the AIO0 pin can be configured to output internal signals of the AFE. The following signals are available for output on the AIO0 pin:

- ADCVREF – 2.5V Voltage Reference
- AFEMUXOUT – AFE Mux Output to the MCU ADC
- IMUXOUT – Current Mux Output to the Current ADC
- VBMUXOUT – Battery Cell Voltage Mux Output to the Voltage ADC



26. MISCELLANEOUS

26.1. General-Purpose Register

The device contains an 8-bit general-purpose register in the analog sub-system that is available for user applications. This register may be used to synchronize information between the MCU and analog sub-system for the application.

The user may read or write this register at **SOC.GP**. This register is only reset on a power-on-reset (POR) or a Push Button (PB) reset.

27. AFE REGISTERS

27.1. Analog Front End Register Map

Table 27-1. Analog Front End Register Map

SOC ADDRESS	NAME	DESCRIPTION	RESET VALUE
Configurable Analog Front End			
0x00	**AFECTL1**	AFE Control 1	
0x01	**AFECTL2**	AFE Control 2	
0x02	**DVRCTL**	Driver Control	
0x03	**AFEMUXCTL**	AFE Mux Control	
0x04	**AFEMUXSEL**	AFE Mux Select	
0x05	**HIBCTL**	Hibernate Control	
0x06	**HIBENTER**	Hibernate Enter	
0x07	**RSTSTAT**	Reset Status	
0x08	**PB**	Push Button	
0x09	**AIO0CFG**	AIO 0 Configuration	
0x10	**PROTKEY**	Protection Key	
0x11	**SIGMGRCTL1**	Signal Manager Control 1	
0x12	**SIGMGRCTL2**	Signal Manager Control 2	
0x13	**PROTEN**	Protection Enable	
0x14	**FUSE**	Fuse Driver Control	
0x15	**PWRFAULTEN**	Power Fault Enable	
0x16	**PWRFAULT**	Power Fault	
0x17	**TEMPFAULTEN**	Temperature Fault Enable	
0x18	**TEMPFAULT**	Temperature Fault	
0x19	**SIGFAULTEN**	Signal Manager Fault Enable	
0x1A	**SIGFAULT**	Signal Manager Fault	
0x1B	**BATRTS**	Battery Real-Time Status	
0x20	**BATOVCFG**	Battery Over-Voltage Configuration	
0x21	**BATOVDAC**	Battery Over Voltage DAC Setting	
0x22	**VADCCTL**	Voltage ADC Control	
0x23	**VADCRESHI**	Voltage ADC Result Hi Byte	
0x24	**VADCRESLO**	Voltage ADC Result Low Byte	
0x25	**IADCCTL**	Current ADC Control	
0x26	**IADCRESHI**	Current ADC Result Hi Byte	
0x27	**IADCRESLO**	Current ADC Result Low Byte	

Configurable Analog Front End Register Map Continued

SOC ADDRESS	NAME	DESCRIPTION	RESET VALUE
Configurable Analog Front End			
0x28	**SCPDAC**	Short-Circuit Protection DAC	
0x29	**SCPCFG**	Short-Circuit Protection Configuraiton	
0x2A	**OCDDAC**	Over-Current Discharge Protection DAC	
0x2B	**OCCCFG**	Over Current Charge Protection Configuration	
0x2C	**OCCDAC**	Over Current Charge Protection DAC	
0x2D	**OCDCFG**	Over Current Discharge Protection Configuraiton	
0x30	**CELLEN1**	Cell Enable 1	
0x31	**CELLEN2**	Cell Enable 2	
0x32	**CELLEN3**	Cell Enable 3	
0x33	**CFGCB1**	Configure Cell Balance 1	
0x34	**CFGCB2**	Configure Cell Balance 2	
0x35	**CFGCB3**	Configure Cell Balance 3	
0x40	**GP**	General Purpose	
0x41	**CLKOUTCFG**	Clock Out Configuration	
0x42	**WWDTCTL**	Windowed Watchdog Timer Control	
0x43	**WWDTCTR**	Windowed Watchdog Timer Counter	
0x44	**WWDTCDV**	Windowed Watchdog Timer Count Down Value	
0x45	**WWDTWIN**	Windowed Watchdog Timer Window	
0x46	**WWDTRST**	Windowed Watchdog Timer Reset	

27.1.1. SOC.AFECTL1

Register 27-1. SOC.AFECTL1 (AFE Control 1, SOC 0x00)

BIT	NAME	ACCESS	RESET	DIFFAMP MODE
7	**SRST**	RW	0	Soft Reset Write to 1 to reset the device. The majority of AFE registers will be reset also unless noted. This bit is self clearing and will always be read as 0.
6	**MCUALIVE**	RW	0	MCU Alive This bit should be written to 1 by the MCU after it comes out of reset. Prior to setting this bit, the AFE will only respond to SPI transactions.
5	**RFU**	RW	0	Reserved, write as 0.
4	**DIS_CPOK**	RW	0	Charge Pump OK Overwrite 0: Normal Operations 1: Overwrite to OK
3	**SCPDAC2VADC**	RW	0	Mux Selection for Voltage SD ADC CH20 0: Select BATOVDAC 1: Select SCPDAC
2	**LOADDETEN**	RW	0	Load Detect Enable 0: Disabled 1: Enabled When enabled, measure PACK+ to determine if a load is present.
1	**HVCPEN**	RW	0	High Voltage Charge Pump Enable 0: Disabled 1: Enabled
0	**SIGEN**	RW	0	Signal Manager Tile Enable 0: disabled 1: enabled

27.1.2. SOC.AFECTL2

Register 27-2. SOC.AFECTL2 (AFE Control 2, SOC 0x01)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:3	**RFU**	R	0x0	Reserved
2:1	**BK_FREQ**	R/W	0x1	High Voltage Buck Switching Frequency: 0: 50kHz 1: 100kHz 2: 200kHz 3: 400KHz
0	**VP_PD_DIS**	R/W	0	VP Pull Down Disable

27.1.3. SOC.DRVCTL

Register 27-3. SOC.DRVCTL (Driver Control, SOC 0x02)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:2	**RFU**	R	0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
1:0	**DRVMODE**	RW	0x0	Driver Mode 0: Drivers Disabled 1: Drivers Enabled - MCU signals control CHG/DSG on off. 2: Source Follower Mode 3: Reserved Driver Mode is set to 0 if SEGEN = 0 or if nRST is active

27.1.4. SOC.AFEMUXCTL

Register 27-4. SOC.AFEMUXCTL (AFE Mux Control, SOC 0x03)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:2	**RFU**	R	0	Reserved
1	**BUFFEN**	RW	0x0	ADC Buffer Enable 0: Disabled 1: Enabled Enables the buffer after the AFE Mux that drives the signal feeding channel 0 of the ADC Mux.
0	**EMUX_EN**	RW	0x0	EMUX Enable – Enables the AFE EMUX Module 0: Disabled 1: Enabled When the EMUX is enabled, writes to the AFEMUXSEL will be performed by the EMUX logic. The AFEMUXSEL cannot be written by the SOC Bridge SPI I/F, but can be read over the SOC Bridge SPI I/F. Setting EMUX_EN = 0 will reset the AFE EMUX module.

27.1.5. SOC.AFEMUXSEL

Register 27-5. SOC.AFEMUXSEL (AFE Mux Select, SOC 0x04)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:5	**RFU**	R	0	Reserved

BIT	NAME	ACCESS	RESET	DESCRIPTION
4:0	**AFEMUXSEL**	R/W	0x0	AFE MUX Channel Selector: 0; VCORE 1 = VCORE / 2.5 2 = VDDA / 2.5 3 = VCCIO / 2.5 4 = VSYS / 2.5 5 = ISENSE 6 = VPTAT 7 = VP / 10 8 = VREF / 2 9 = FUSE / 10 10 = CHG / 50 11 = DSG / 50 12 = BAT / 50 13 = AIO0A 14 = LOADDET 15 = SCPDAC 16 = OCCDAC 17 = OCDDAC 18 = BATOVDAC 19 = VIN / 50 20 = PACK+ / 50 21 = VCP / 50 This register is only writeable by the SOC Bridge SPI I/F when EMUX_EN=0, but is readable by the SOC Bridge SPI I/F at anytime.

27.1.6. SOC.HIBCTL

Register 27-6. SOC.HIBCTL (Hibernate Control, SOC 0x05)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:5	**WUTIMER**	R/W	0x0	Wake Up Timer Duration 0: Disabled 1: 125ms 2: 250ms 3: 500ms 4: 1s 5: 2s 6: 4s 7: 8s
4:3	**WAKESRC**	R/W	0x0	Wake Up Source 0: PB 1: PACK+ 2: WUTIMER 3: RFU
2	**PACKWAKEVREF**	R/W	0x0	PACK+ Wake Up Voltage Reference Threshold 0: 5V 1: 20V
1	**PACKWAKEEN**	R/W	0x0	PACK+ Wake Up Enable 0: Disabled 1: Enabled
0	**PBWAKEEN**	R/W	0x0	Push Button Wake Up Enable 0: Disabled 1: Enabled

Note: If all wake-up enables are set to Disabled when HIBENTER.HIB is written, then PBWAKEN and PACKWAKEN will be set to 1 as a fail safe to allow the device to be brought out of hibernate.

27.1.7. SOC.HIBENTER

Register 27-7. SOC.HIBENTER (Hibernate Enter, SOC 0x06)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:1	**RFU**	R	0	Reserved, write as 0.
0	**HIB**	R/W	0x0	Hibernate Write to 1 to enter Hibernate. This bit is automatically cleared upon wakeup.

27.1.8. SOC.RSTSTAT

Register 27-8. SOC.RSTSTAT (Reset Status, SOC 0x07)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:5	**RFU**	R	0x0	Reserved
4	**PBRST**	W1C	0x0	Push Button Reset Flag This flag will be set when the Push Button is enabled and has been active for 8 seconds.
3	**HIBRST**	W1C	0x0	Hibernate Reset Flag This flag will be set when the device has been reset following a hibernate wake-up. Read the HIBCTL.WAKESRC bits to determine the source of the hibernate wake-up.
2	**WDTRST**	W1C	0x0	AFE Watchdog Timer Reset Flag This flag will be set when the device has been reset by the AFE Watchdog Timer
1	**SOFTRST**	W1C	0x0	Soft Reset Flag This flag will be set when an AFE Soft Reset has been performed by writing a 1 to SOC.AFECTL1.SRST.
0	**FLTRST**	W1C	0x0	Fault Reset Flag This flag will be set when a power or temperature fault occurs that causes a device reset. The following faults may cause a Fault Reset. Power Faults: VPFLT, VSYSFLT,VDDIOFLT,VDDAFLT,VCOREFLT Temp Faults: TMPFLT Following a Fault Reset, read the PWRFAULT and TEMPFAULT registers to determine which faults caused the reset.

27.1.9. SOC.PB

Register 27-9. SOC.PB (Push Button, SOC 0x08)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**PBSTAT_RTS**	R	0x0	
6	**PBINTF**	W1C	0x0	Push Button Interrupt Flag
5	**PBINTEN**	R/W	0x0	Push Button Interrupt Enable The interrupt is level sensitive.
4	**PBPOL**	R/W	0x0	Push Button Polarity 0: Active Low 1: Active High Depending on other bit settings, an active signal will generate an interrupt, hibernate wake-up, or reset.
3:2	**PBDT**	R/W	0x0	Push Button Deglitch Time 0: 1ms 1: 4ms 2: 8ms 3: 32ms
1	**PBRSTEN**	R/W	0x0	Push Button Reset Enable 0: Disabled 1: Enabled If enabled, when the Push Button has been active for 8 seconds, the device shall be reset.
0	**PBEN**	R/W	0x0	Push Button Enabled 0: Disabled 1: Enabled

27.1.10. SOC.AIO0CFG

Register 27-10. SOC.AIO0CFG (Analog I/O 0 Configuration, SOC 0x09)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**MUXSEL**	R/W	0x0	AIO0 Mux Select - during I/O output mode 0: ADC VREF = 2.5V 1: AFEMUXOUT 2: IMUXOUT 3: VBMUXOUT 15:4: RFU
4	**RFU**	R/W	0x0	Reserved
3:2	**SWAP**	R/W	0x0	Swaps the offset of the buffer. 0: No swap 1: Swap
1	**MODE**	R/W	0x0	AIO0 Buffer Mode 0: Input Buffer Mode 1: Output Buffer mode (2mA drive strength)
0	**AIO0EN**	R/W	0x0	AIO0 Buffer Enable 0: Disabled 1: Enabled

27.1.11. SOC.PROT_KEY

Register 27-11. SOC.PROT_KEY (Protection Key, SOC 0x10)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**KEY**	RW	0	Protection Key Select SOC registers require that PROT_KEY..KEY be written with 0xA5 before those registers can be written. The KEY field is self clearing to 0x0 after any write to one of the select registers.

27.1.12. SOC.SIGMGRCTL1

Register 27-12. SOC.SIGMGRCTL1 (Signal Manager Control 1, SOC 0x11)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:5	**RFU**	R	0x0	Reserved
4	**DAEN**	R/W	0x0	Current Sense Differential Amp Enable
3	**SCPEN**	R/W	0x0	Short Circuit Protection DAC and Comparator Enable
2	**OCCEN**	R/W	0x0	Over Current Charge Protection DAC and Comparator Enable
1	**OCDEN**	R/W	0x0	Over Current Discharge Protection DAC and Comparator Enable
0	**BATOVEN**	R/W	0x0	Battery Over Voltage DAC and Comparator Enable

Note: This register requires a write of PROT_KEY before register can be written.

27.1.13. SOC.SIGMGRCTL2

Register 27-13. SOC.SIGMGRCTL2 (Signal Manager Control 2, SOC 0x12)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:3	**RFU**	R	0x0	Reserved
2	**PBPTEN**	R/W	0x0	Push Button Pass Through Enable
1	**IADCEN**	R/W	0x0	Current ADC Enable
0	**VADCEN**	R/W	0x0	Voltage ADC Enable

27.1.14. SOC.PROTEN

Register 27-14. SOC.PROTEN (Reset Status, SOC 0x13)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:6	**RFU**	R	0x0	Reserved
5	**SCPCPROTEN**	R/W	0x0	Short Circuit Protection CHG Protection Enable – If enabled, then when protection is activated the CHG FET will be disabled
4	**SCPDPROTEN**	R/W	0x0	Short Circuit Protection DSG Protection Enable – If enabled, then when protection is activated the DSG FET will be disabled
3	**OCCPROTEN**	R/W	0x0	Over Current CHG Protection Enable – If enabled, then when protection is activated the CHG FET will be disabled
2	**OCDPROTEN**	R/W	0x0	Over Current DSG Protection Enable – If enabled, then when protection is activated the DSG FET will be disabled
1	**BATOVCPROTEN**	R/W	0x0	Battery Over Voltage CHG Protection Enable – If enabled, then when protection is activated the CHG FET will be disabled
0	**BATOVDPROTEN**	R/W	0x0	Battery Over Voltage DSG Protection Enable – If enabled, then when protection is activated the DSG FET will be disabled

Note: This register requires a write of PROT_KEY before register can be written.

27.1.15. SOC.FUSE

Register 27-15. SOC.FUSE (Fuse Driver Control, SOC 0x14)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:1	**FUSEKEY**	R/W	0	Must write FUSEKEY = 0x66 and PROT_KEY must already be written with 0xA5 for a write to FUSE.FUSEEN to succeed. FUSEKEY is automatically cleared to 0x00 following the write to the FUSE register.
0	**FUSEEN**	R/W	0	Enable FUSE FET gate drive output

Note: This register requires a write of PROT_KEY before register can be written.

27.1.16. SOC.PWRFAULTEN

Register 27-16. SOC.PWRFAULTEN (Power Fault Interrupt Enable, SOC 0x15)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**RFU**	R	0x0	Reserved
6	**DRVFLTEN**	R/W	0x0	Driver Fault Interrupt Enable
5	**VCCIOFLTEN**	R/W	0x0	VCCIO Fault Interrupt Enable
4	**VDDAFLTEN**	R/W	0x0	VDDA Fault Interrupt Enable
3	**VCOREFLTEN**	R/W	0x0	VCORE Fault Interrupt Enable
2	**VSYSFLTEN**	R/W	0x0	VSYS Fault Interrupt Enable
1	**VPFLTEN**	R/W	0x0	VP Fault Interrupt Enable
0	**HVCPFLTEN**	R/W	0x0	HVCP Fault Interrupt Enable

Note: This register requires a write of PROT_KEY before register can be written.

27.1.17. SOC.PWRFAULT

Register 27-17. SOC.PWRFAULT (Power Fault, SOC 0x16)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**RFU**	R	0x0	Reserved
6	**DRVFLT**	W1C	0x0	Driver Fault Flag
5	**VCCIOFLT**	W1C	0x0	VCCIO Fault Flag
4	**VDDAFLT**	W1C	0x0	VDDA Fault Flag
3	**VCOREFLT**	W1C	0x0	VCORE Fault Flag
2	**VSYSFLT**	W1C	0x0	VSYS Fault Flag
1	**VPFLT**	W1C	0x0	VP Fault Flag
0	**HVCPFLT**	W1C	0x0	HVCP Fault Flag

27.1.18. SOC.TEMPFAULTEN

Register 27-18. SOC.TEMPFAULTEN (Temperature Fault Interrupt Enable, SOC 0x17)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:6	**RFU**	R	0x0	Reserved
5	**TWARN2CBDEN**	R/W	0x0	TWARN2 Cell Balance Disable – If set to 1, then when TWARN2 occurs, Cell Balancing will be disabled
4	**TWARN1CBDEN**	R/W	0x0	TWARN1 Cell Balance Disable – If set to 1, then when TWARN1 occurs, Cell Balancing will be disabled
3	**RFU**	R	0x0	Reserved
2	**TMPFLTEN**	R/W	0x0	Temperature Fault Interrupt Enable
1	**TWARN2EN**	R/W	0x0	TWARN2 Fault Interrupt Enable
0	**TWARN1EN**	R/W	0x0	TWARN1 Fault Interrupt Enable

Note: This register requires a write of PROT_KEY before register can be written.

27.1.19. SOC.TEMPFAULT

Register 27-19. SOC.TEMPFAULT (Temperature Fault Flag, SOC 0x18)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**TWARN2C_RTS**	R/W	0x0	TWARN2 Real-Time Status
6	**TWARN1_RTS**	R/W	0x0	TWARN1 Real-Time Status
5:3	**RFU**	R	0x0	Reserved
2	**TMPFLT**	W1C	0x0	Temperature Fault Flag
1	**TWARN2**	W1C	0x0	TWARN2 Fault Flag
0	**TWARN1**	W1C	0x0	TWARN1 Fault Flag

27.1.20. SOC.SIGFAULTEN

Register 27-20. SOC.SIGFAULTEN (Signal Manager Fault Interrupt Enable, SOC 0x19)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:6	**RFU**	R	0x0	Reserved
4	**EMUXFLTEN**	R/W	0x0	EMUX Fault Interrupt Enable
3	**SCPFLTEN**	R/W	0x0	Short Circuit Protection Fault Interrupt Enable
2	**OCCFLTEN**	R/W	0x0	Over Current Charge Fault Interrupt Enable
1	**OCDFLTEN**	R/W	0x0	Over Current Discharge Fault Interrupt Enable
0	**BATOVFLTEN**	R/W	0x0	Battery Over Voltage Fault Interrupt Enable

Note: This register requires a write of PROT_KEY before register can be written.

27.1.21. SOC.SIGFAULT

Register 27-21. SOC.SIGFAULT (Signal Manager Fault Flag, SOC 0x1A)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**RFU**	R	0x0	Reserved
6	**CHGFLT**	W1C	0x0	CHG Fault Flag – This flag will be set if a CHG Protection is enabled in the PROTEN register and one of the protections trips and disables the CHG FET. – This bit must be written with a 1 to clear it before the CHG FET can be enabled again.
5	**DSGFLT**	W1C	0x0	DSG Fault Flag – This flag will be set if a DSG Protection is enabled in the PROTEN register and one of the protections trips and disables the DSG FET. – This bit must be written with a 1 to clear it before the DSG FET can be enabled again.
4	**EMUXFLT**	W1C	0x0	EMUX Fault Flag – flag is set if EMUX bits [7:5] ! = 010b
3	**SCPFLT**	W1C	0x0	Short Circuit Protection Fault Flag
2	**OCCFLT**	W1C	0x0	Over Current Charge Fault Flag
1	**OCDFLT**	W1C	0x0	Over Current Discharge Fault Flag
0	**BATOVFLT**	W1C	0x0	Battery Over Voltage Fault Flag

27.1.22. SOC.BATRTS

Register 27-22. SOC.BATRTS (Battery Protection Comparator Real-Time Status, SOC 0x1B)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**RFU**	R	0x0	Reserved
3	**SCP_RTS**	R/W	0x0	Short Circuit Protection Real-Time Status
2	**OCC_RTS**	R/W	0x0	Over Current Charge Real-Time Status
1	**OCD_RTS**	R/W	0x0	Over Current Discharge Real-Time Status
0	**BATOV_RTS**	R/W	0x0	Battery Over Voltage Real-Time Status

27.1.23. SOC.BATOVCFG

Register 27-23. SOC.BATOVCFG (Battery Over Voltage Comparator Configuration, SOC 0x20)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**BLANKSF**	R/W	0x0	Blanking Scale Factor 0 = 1, 1 = 2, 2 = 3.....14 = 15, 15 = 16
3:0	**TIMEBASE**	R/W	0x0	Time Base: 0 = 32uS, 1 = 64uS, 2 = 128uS, 3 = 256uS …....15 = 1,048,576 uS

Notes:

This register requires a write of PROT_KEY before register can be written.

Blanking Time = TIMEBASE * BLANKSF

Example: TIMEBASE = 1, BLANKSF = 2; So, Blanking Time = 64uS * 3 = 192uS

BATOV Comparator Hysteresis is fixed at 100mV

27.1.24. SOC.BATOVDAC

Register 27-24. SOC.BATOVDAC (Battery Over Voltage DAC, SOC 0x21)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**BATOVDAC**	RW	0	BATOV DAC Setting – This is the comparator threshold for the BATOV comparator.

Note: This register requires a write of PROT_KEY before register can be written.

27.1.25. SOC.VADCCTL

Register 27-25. SOC.VADCCTL (Voltage ADC Control, SOC 0x22)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**VADCSTART**	R/W	0x0	Cell Voltage ADC Start Conversion
6	**VADCBUSY**	R	0x0	Cell Voltage ADC Busy - This bit is set to 1 during conversion and set to 0 when complete.
5	**RFU**	R	0	Reserved
4:0	**VBMUXSEL**	R/W	0x0	Voltage ADC MUX Select: 0: VB1 1: VB2 2: VB3 3: VB4 4: VB5 5: VB6 6: VB7 7: VB8 8: VB9 9: VB10 10: VB11 11: VB12 12: VB13 13: VB14 14: VB15 15: VB16 16: VB17 17: VB18 18: VB19 19: VB20 20: BATOVDAC or SCPDAC (see AFECTL1) 21 - 31: RFU

27.1.26. SOC.VADCRESHI

Register 27-26. SOC.VADCRESHI (Voltage ADC Result High, SOC 0x23)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**VADCRES[15:8]**	RW	0	Voltage ADC Result MSByte

27.1.27. SOC.VADCRESLO

Register 27-27. SOC.VADCRESLO (Voltage ADC Result Low, SOC 0x24)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**VADCRES[7:0]**	RW	0	Voltage ADC Result LSByte

27.1.28. SOC.IADCCTL

Register 27-28. SOC.IADCCTL (Current ADC Control, SOC 0x25)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**IADCSTART**	R/W	0x0	Current ADC Start Conversion
6	**IADCBUSY**	R	0x0	Current ADC Busy - This bit is set to 1 during conversion and set to 0 when complete.
5	**RFU**	R	0	Reserved
4:3	**IMUXSEL[1:0]**	R/W	0x0	0: Isense Diff Amp Output 1: SCP DAC 2: OCC DAC 3: OCD DAC
2:0	**DAGAIN[2:0]**	R/W	0x0	Differential Amplifier Gain: 0: 1X 1: 2X 2: 4X 3: 8X 4: 16X 5: 32X 6: 64X 7: 128X

27.1.29. SOC.IADCRESHI

Register 27-29. SOC.IADCRESHI (Current ADC Result High, SOC 0x26)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**IADCRES[15:8]**	RW	0	Current ADC Result MSByte

27.1.30. SOC.IADCRESLO

Register 27-30. SOC.IADCRESLO (Current ADC Result Low, SOC 0x27)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**IADCRES[7:0]**	RW	0	Current ADC Result LSByte

27.1.31.

27.1.32. SOC.SCPDAC

Register 27-31. SOC.SCPDAC (SCP DAC, SOC 0x28)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**SCPDAC**	RW	0	SCP DAC Setting – This is the comparator threshold for the SCP comparator.

Note: This register requires a write of PROT_KEY before register can be written.

27.1.33. SOC.SCPCFG

Register 27-32. SOC.SCPCFG (SCP Comparator Configuration, SOC 0x29)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**BLANKSF**	R/W	0x0	Blanking Scale Factor 0 = 1, 1 = 2, 2 = 3.....14 = 15, 15 = 16
3:0	**TIMEBASE**	R/W	0x0	Time Base: 0 = 1uS, 1 = 2uS, 2 = 4uS, 3 = 256uS …....15 = 32768 uS

Notes:

This register requires a write of PROT_KEY before register can be written.

Blanking Time = TIMEBASE * BLANKSF

Example: TIMEBASE = 1, BLANKSF = 2; So, Blanking Time = 2uS * 3 = 6uS

SCP Comparator Hysteresis is fixed at 25mV

27.1.34. SOC.OCDDAC

Register 27-33. SOC.OCDDAC (OCD DAC, SOC 0x2A)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**OCDDAC**	RW	0	OCD DAC Setting – This is the comparator threshold for the OCD comparator.

Note: This register requires a write of PROT_KEY before register can be written.

27.1.35. SOC.OCDCFG

Register 27-34. SOC.OCDCFG (OCD Comparator Configuration, SOC 0x2D)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**BLANKSF**	R/W	0x0	Blanking Scale Factor 0 = 1, 1 = 2, 2 = 3.....14 = 15, 15 = 16
3:0	**TIMEBASE**	R/W	0x0	Time Base: 0 = 1uS, 1 = 2uS, 2 = 4uS, 3 = 256uS …....15 = 32768 uS

Notes:

This register requires a write of PROT_KEY before register can be written.

Blanking Time = TIMEBASE * BLANKSF

Example: TIMEBASE = 1, BLANKSF = 2; So, Blanking Time = 2uS * 3 = 6uS

OCD Comparator Hysteresis is fixed at 25mV

27.1.36. SOC.OCCDAC

Register 27-35. SOC.OCCDAC (OCC DAC, SOC 0x2C)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**OCCDAC**	RW	0	OCC DAC Setting – This is the comparator threshold for the OCC comparator.

Note: This register requires a write of PROT_KEY before register can be written.

27.1.37. SOC.OCCCFG

Register 27-36. SOC.OCCCFG (OCC Comparator Configuration, SOC 0x2B)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**BLANKSF**	R/W	0x0	Blanking Scale Factor 0 = 1, 1 = 2, 2 = 3.....14 = 15, 15 = 16
3:0	**TIMEBASE**	R/W	0x0	Time Base: 0 = 1uS, 1 = 2uS, 2 = 4uS, 3 = 256uS …....15 = 32768 uS

Notes:

This register requires a write of PROT_KEY before register can be written.

Blanking Time = TIMEBASE * BLANKSF

Example: TIMEBASE = 1, BLANKSF = 2; So, Blanking Time = 2uS * 3 = 6uS

OCC Comparator Hysteresis is fixed at 25mV

27.1.38.

27.1.39. SOC.CELLEN1

Register 27-37. SOC.CELLEN1 (Cell Enable 1, SOC 0x30)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**CEN8**	R/W	0x0	Cell 8 Enable
6	**CEN7**	R/W	0x0	Cell 7 Enable
5	**CEN6**	R/W	0x0	Cell 6 Enable
4	**CEN5**	R/W	0x0	Cell 5 Enable
3	**CEN4**	R/W	0x0	Cell 4 Enable
2	**CEN3**	R/W	0x0	Cell 3 Enable
1	**CEN2**	R/W	0x0	Cell 2 Enable
0	**CEN1**	R/W	0x0	Cell 1 Enable

27.1.40. SOC.CELLEN2

Register 27-38. SOC.CELLEN2 (Cell Enable 2, SOC 0x31)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**CEN16**	R/W	0x0	Cell 16 Enable
6	**CEN15**	R/W	0x0	Cell 15 Enable
5	**CEN14**	R/W	0x0	Cell 14 Enable
4	**CEN13**	R/W	0x0	Cell 13 Enable
3	**CEN12**	R/W	0x0	Cell 12 Enable
2	**CEN11**	R/W	0x0	Cell 11 Enable
1	**CEN10**	R/W	0x0	Cell 10 Enable
0	**CEN9**	R/W	0x0	Cell 9 Enable

27.1.41. SOC.CELLEN3

Register 27-39. SOC.CELLEN3 (Cell Enable 3, SOC 0x32)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**RFU**	R	0x0	Reserved
3	**CEN20**	R/W	0x0	Cell 20 Enable
2	**CEN19**	R/W	0x0	Cell 19 Enable
1	**CEN18**	R/W	0x0	Cell 18 Enable
0	**CEN17**	R/W	0x0	Cell 17 Enable

27.1.42. SOC.CFGCB1

Register 27-40. SOC.CFGCB1 (Configure Cell Balance 1, SOC 0x33)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**VB8**	R/W	0x0	Cell 8 Cell Balance Enable
6	**VB7**	R/W	0x0	Cell 7 Cell Balance Enable
5	**VB6**	R/W	0x0	Cell 6 Cell Balance Enable
4	**VB5**	R/W	0x0	Cell 5 Cell Balance Enable
3	**VB4**	R/W	0x0	Cell 4 Cell Balance Enable
2	**VB3**	R/W	0x0	Cell 3 Cell Balance Enable
1	**VB2**	R/W	0x0	Cell 2 Cell Balance Enable
0	**VB1**	R/W	0x0	Cell 1 Cell Balance Enable

27.1.43. SOC.CFGCB2

Register 27-41. SOC.CFGCB2 (Configure Cell Balance 2, SOC 0x34)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**VB16**	R/W	0x0	Cell 16 Cell Balance Enable
6	**VB15**	R/W	0x0	Cell 15 Cell Balance Enable
5	**VB14**	R/W	0x0	Cell 14 Cell Balance Enable
4	**VB13**	R/W	0x0	Cell 13 Cell Balance Enable
3	**VB12**	R/W	0x0	Cell 12 Cell Balance Enable
2	**VB11**	R/W	0x0	Cell 11 Cell Balance Enable
1	**VB10**	R/W	0x0	Cell 10 Cell Balance Enable
0	**VB9**	R/W	0x0	Cell 9 Cell Balance Enable

27.1.44. SOC.CFGCB3

Register 27-42. SOC.CFGCB3 (Configure Cell Balance 3, SOC 0x35)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:4	**RFU**	R	0x0	Reserved
3	**VB20**	R/W	0x0	Cell 20 Cell Balance Enable
2	**VB19**	R/W	0x0	Cell 19 Cell Balance Enable
1	**VB18**	R/W	0x0	Cell 18 Cell Balance Enable
0	**VB17**	R/W	0x0	Cell 17 Cell Balance Enable

27.1.45. SOC.GP

Register 27-43. SOC.GP (General-Purpose Register, SOC 0x40)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**GP**	RW	0x0	General-purpose read-write register.

27.1.46. SOC.CLKOUTCFG

Register 27-44. SOC.CLKOUTCFG (Clock Out Configuration, SOC 0x41)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:3	**RFU**	R	0	Reserved
2:1	**CLKOUTFREQ[1:0]**	RW	0x0	Low-Speed Clock Output Frequency Setting 0: 250Hz 1: 500Hz 2: 1kHz 3: 2kHz
0	**CLKOUTEN**	RW	0x0	Low Speed Clock Output Enable

Note: Used during clock test that can help meet Class B Safety

27.1.47. SOC.WWDTCTL

Register 27-45. SOC.WWDTCTL (Windowed Watchdog Timer Control, SOC 0x42)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:3	**KEY**	R	0	Write KEY = 0x14 to modify WWDT registers. All other values disallow writes to WWDT registers except WWDTRST register which can be written at anytime.
2:1	**CLKDIV[1:0]**	RW	0x0	WWDT Clock Divider – The WWDT Clock = 32kHz / CLKDIV 0: /2 1: /16 2: /128 3: /1024
0	**EN**	RW	0x0	WWDT Enable

Note: The WWDT runs off of a 32kHz clock that is independent of the 4MHz CLKREF that the MCU runs off. When the WWDT is used, it can help meet Class B Safety requirements.

27.1.48. SOC.WWDTCTR

Register 27-46. SOC.WWDTCTR (Windowed Watchdog Timer Counter, SOC 0x43)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**CTR[7:0]**	RW	0x0	WWDT Counter Value

27.1.49. SOC.WWDTCDV

Register 27-47. SOC.WWDTCDV (Windowed Watchdog Timer Count Down Value, SOC 0x44)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**CDV7:0]**	RW	0x0	WWDT Count Down Value

27.1.50. SOC.WWDTWIN

Register 27-48. SOC.WWDTWIN (Windowed Watchdog Timer Window, SOC 0x45)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**WINDOW[7:0]**	RW	0x0	WWDT Window Value - If WWDTRST is written when CTR >= WINDOW, then the WWDT will issue a device reset.

27.1.51. SOC.WWDTRST

Register 27-49. SOC.WWDTRST (Windowed Watchdog Timer Reset, SOC 0x46)

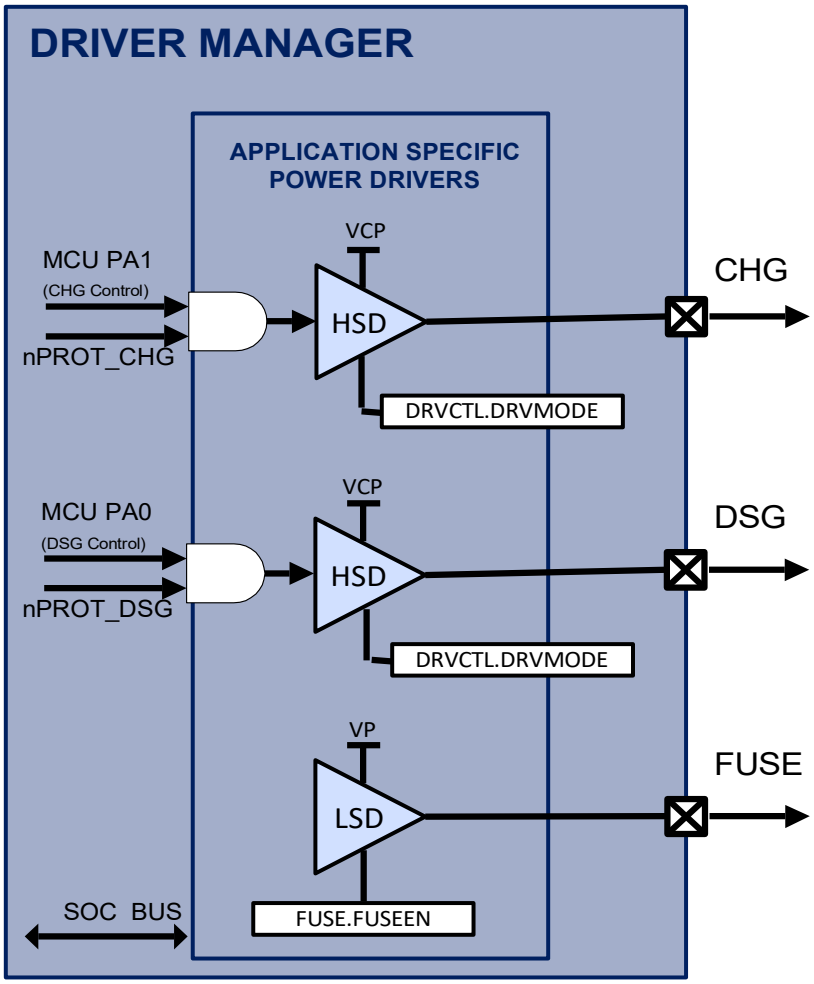
BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**RESET[7:0]**	RW	0x0	WWDT Reset (Pet the Dog / Feed the Dog) - Write this register with 0xAC o keep it from resetting the device. - A Write of 0xAC will reset the WWDT by reloading the CDV value into the CTR. - If CTR >= WINDOW when 0xAC is written, then the WWDT shall issue a device reset. - If CTR counts down to 0 before RESET is written with 0xAC the WWDT will issue a device reset. - This register shall automatically be cleared to 0x00, and shall always read 0x00.

28. DRIVER MANAGER

28.1. Features

- High-Side gate drivers for CHG and DSG FET
- Low-side gate driver for external FUSE

28.2. Block Diagram



28.3. Functional Description

The Application Specific Power Drivers™ (ASPD) module drives the gate of the external CHG and DSG FETs and external protection fuse FET for the battery pack.

The CHG and DSG FET gates are driven from the CHG and DSG pins. The gate drive voltage for the CHG and DSG FETs is VCP (BAT + 9V). The FUSE output is a low-side switch supplied by VP which is intended for driving the gate on an external FET, which is used to blow an external fuse in series with PACK+ and the battery stack.

The ASPD also integrates gate driver over-current, under-voltage and over-voltage protection. Over current and battery over voltage protections can be enabled using the SIGMGRCTL1 register and the PROTEN register. These protections control the nPROT_CHG and nPROT_DSG signals that disable the gate drivers when active.

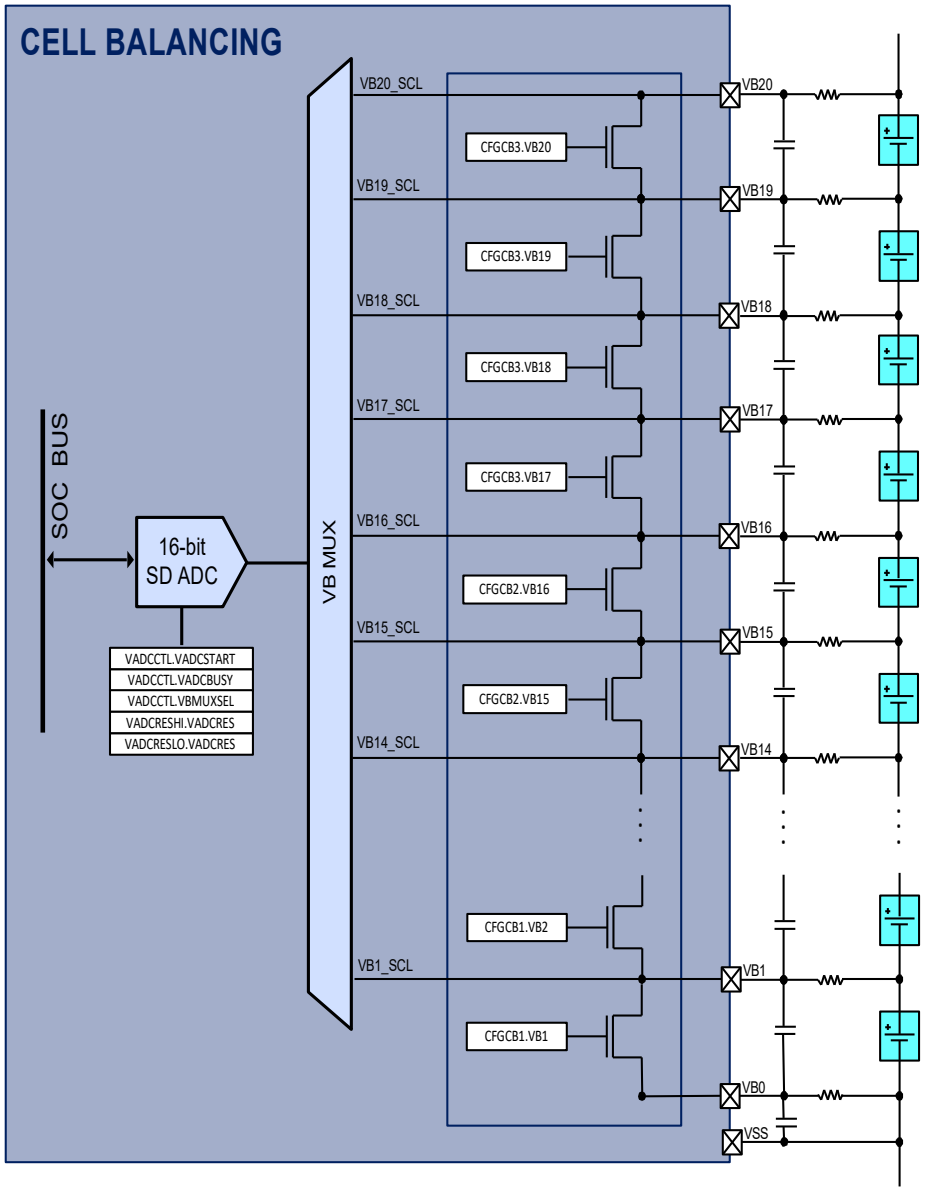
29. CELL BALANCING

The PAC22140 contains integrated cell balancing FETs for up to 20 cells.

29.1. Features

- Cell Balancing FETs for up to 20 cells
- Allows for discharge of individual cells
- Voltage ADC for sensing the voltage of each cell

29.2. Block Diagram



29.3. Functional Description

The integrated cell balancing contains FETs for up to 20 battery cells. Cell balancing can be performed through firmware programming. Each of the battery cell voltages from the VB[1..20] pins are available for sampling from the 16-bit ADC.

Adjacent cells should not be balanced at the same time. In the event that too many cells are being balanced at the same time and Thermal protection occurs, then the cell balancing will be shut down first.

Only enabled cells can be balanced. Each independent cell can be enabled by setting their respective CENx enable bit. This can be accomplished by writing to the SOC.CELLEN1.CENx, SOC.CELLEN2.CENx and SOC.CELLEN3.CENx registers.

To start the cell balancing process, simply set the respective VBx cell balancing enable bit. This can be accomplished by writing to the SOC.CFGCB1, SOC.CFGCB2 and SOC.CFGCB3 registers.

30. ARM CORTEX-M0 REFERENCE

30.1. Introduction

30.1.1. Overview

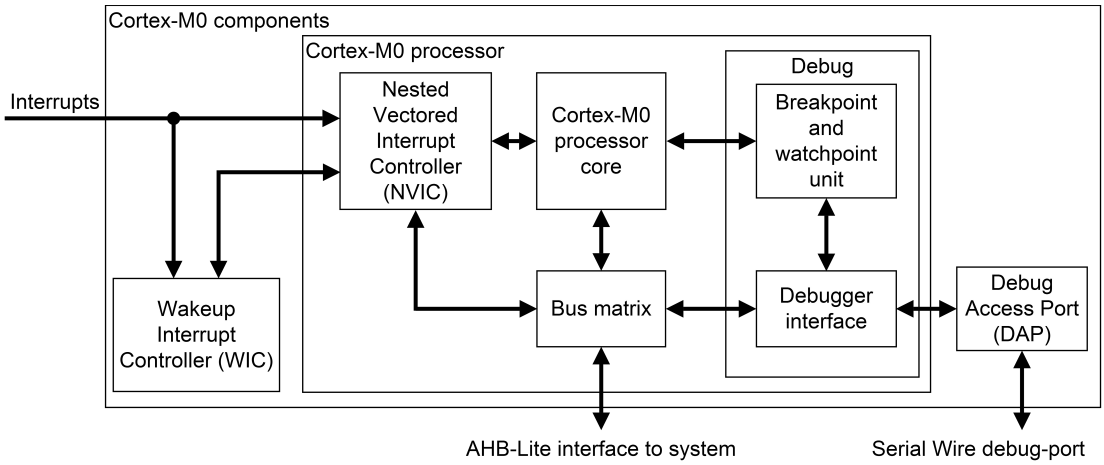
This chapter is taken from the ARM Cortex-M0 User Guide with minimal modifications made to account for the specific Cortex-M0 implementation.

30.1.2. About the Cortex-M0 processor and core peripherals

The Cortex™-M0 processor is an entry-level 32-bit ARM Cortex processor designed for a broad range of embedded applications. It offers significant benefits to developers, including:

- a simple architecture that is easy to learn and program
- ultra-low power, energy efficient operation
- excellent code density
- deterministic, high-performance interrupt handling
- upward compatibility with Cortex-M processor family.

Figure 30-1. Cortex-M0 implementation



The Cortex-M0 processor is built on a highly area and power optimized 32-bit processor core, with a 3-stage pipeline von Neumann architecture. The processor delivers exceptional energy efficiency through a small but powerful instruction set and extensively optimized design, providing high-end processing hardware including a single-cycle multiplier.

The Cortex-M0 processor implements the ARMv6-M architecture, which is based on the 16-bit Thumb® instruction set and includes Thumb-2 technology. This provides the exceptional performance expected of a modern 32-bit architecture, with a higher code density than other 8-bit and 16-bit microcontrollers.

The Cortex-M0 processor closely integrates a configurable Nested Vectored Interrupt Controller (NVIC), to deliver industry-leading interrupt performance. The NVIC:

- includes a non-maskable interrupt (NMI)

- provides zero jitter interrupt option
- provides four interrupt priority levels.

The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to abandon and restart load-multiple and store-multiple operations. Interrupt handlers do not require any assembler wrapper code, removing any code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down.

30.1.2.1. System-level interface

The Cortex-M0 processor provides a single system-level interface using AMBA® technology to provide high speed, low latency memory accesses.

30.1.2.2. Integrated configurable debug

The Cortex-M0 processor implements a complete hardware debug solution, with extensive hardware breakpoint and watchpoint options. This provides high system visibility of the processor, memory and peripherals through a Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices. The device supports 4 hardware breakpoints.

30.1.2.3. Cortex-M0 processor features summary

- high code density with 32-bit performance
- tools and binary upwards compatible with Cortex-M processor family
- integrated ultra low-power sleep modes
- efficient code execution permits slower processor clock or increases sleep mode time
- single-cycle 32-bit hardware multiplier
- zero jitter interrupt handling
- extensive debug capabilities.

30.1.2.4. Cortex-M0 core peripherals

These are:

30.1.2.4.1. NVIC

The NVIC is an embedded interrupt controller that supports low latency interrupt processing.

30.1.2.4.2. System Control Block

The System Control Block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

30.1.2.4.3. System timer

The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

30.2. The Cortex-M0 Processor

30.2.1. Programmers Model

This section describes the Cortex-M0 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and stacks.

30.2.1.1. Processor modes

The processor modes are:

Thread mode

Used to execute application software. The processor enters Thread mode when it comes out of reset.

Handler mode

Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

30.2.1.2. Stacks

The processor uses a full descending stack. This means the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the main stack and the process stack, with independent copies of the stack pointer, see Stack Pointer in chapter 30.2.1.3.2 on page 264.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see CONTROL register in chapter 30.2.1.3.12 on page 268. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 30-1. Summary of processor mode and stack use options

PROCESSOR MODE	USED TO EXECUTE	STACK USED
Thread	Applications	Main stack or process stack*
Handler	Exception handlers	Main stack

* See CONTROL Register in chapter 30.2.1.3.12 on page 268

30.2.1.3. Core Registers

Figure 30-2. Core Registers

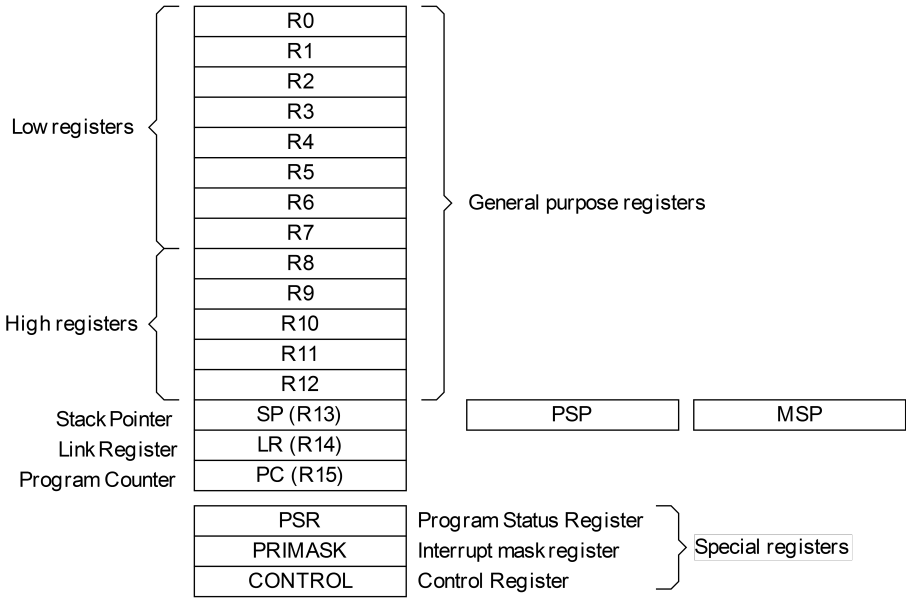


Table 30-2. Core register set summary

NAME	TYPE*	RESET VALUE	DESCRIPTION
R0 – R12	RW	Unknown	General Purpose Register,see chapter 30.2.1.3.1 on page 264
MSP	RW	See description	Stack pointer, see chapter 30.2.1.3.2 on page 264
PSP	RW	Unknown	Stack pointer, see chapter 30.2.1.3.2 on page 264
LR	RW	Unknown	Link Register, see chapter 30.2.1.3.3 on page 265
PC	RW	See description	Program Counter, see chapter 26630.2.1.3.4 on page 265
PSR	RW	Unknown **	Program Status Register, see chapter 30.2.1.3.5 on page 265
APSR	RW	Unknown	Application Program status register, see chapter 30.2.1.3.6 on page 266
IPSR	RW	0x0000 0000	Interrupt Program status register, see chapter 30.2.1.3.7 on page 266
ESPR	RW	Unknown **	Execution Program status register, see chapter 30.2.1.3.8 on page 267
PRIMASK	RW	0x0000 0000	Priority Mask Register, see chapter 30.2.1.3.11 on page 268
CONTROL	RW	0x0000 0000	Control Register, see chapter 30.2.1.3.12 on page 268

*. Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

**. Bit[24] is the T-bit and is loaded from bit[0] of the reset vector.

30.2.1.3.1. General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations.

30.2.1.3.2. Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = Main Stack Pointer (MSP). This Is the reset value.
- 1 = Process Stack Pointer (PSP).

On reset, the processor loads the MSP with the value from address 0x0000 0000

30.2.1.3.3. Link Register

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the LR value is Unknown.

30.2.1.3.4. Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x0000 0004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

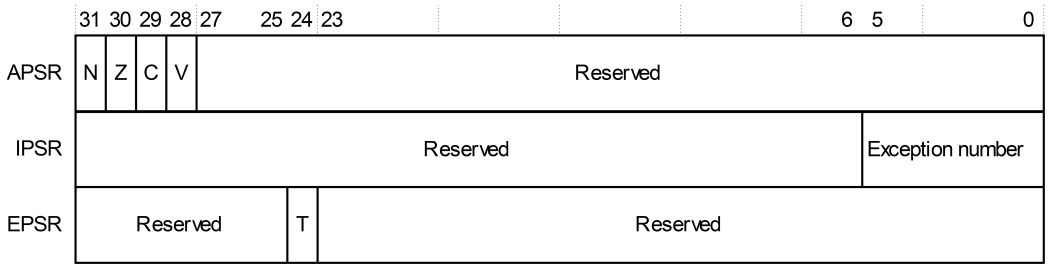
30.2.1.3.5. Program Status Register

The Program Status Register (PSR) combines:

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The PSR bit assignments are:

Figure 30-3. PSR



Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the `MSR` or `MRS` instructions. For example:

- read all of the registers using PSR with the `MRS` instruction
- write to the APSR using APSR with the `MSR` instruction.

The PSR combinations and attributes are:

Table 30-3. Core register set summary

REGISTER	TYPE*	COMBINATION
PSR	RW *, **	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	RW,*	APSR and IPSR
EAPST	RW,**	APSR and EPSR

* The processor ignores writes to the IPSR bits.

** Reads of the EPSR bits return zero, and the processor ignores writes to the these bits

See the instruction descriptions `MRS` in chapter 30.3.7.6 on page 316 and `MSR` in chapter 30.3.7.7 on page 316 for more information about how to access the program status registers.

30.2.1.3.6. Application Program Status Register

The APSR contains the current state of the condition flags, from previous instruction executions. See the register summary in Table 30-2. Core register set summary on page 264 for its attributes. The bit assignments are:

Table 30-4. APSR bit assignments

BIT	NAME	FUNCTION
31	**N**	Negative Flag
30	**Z**	Zero Flag
29	**C**	Carry or Borrow Flag
28	**V**	Overflow Flag
27:0	**Reserved**	Reserved

See The condition flags in chapter 30.3.3.6.1 on page 291 for more information about the APSR negative, zero, carry or borrow, and overflow flags.

30.2.1.3.7. Interrupt Program Status Register

The IPSR contains the exception number of the current Interrupt Service Routine (ISR). See the register summary in Table 30-2. Core register set summary on page 264 for its attributes. The bit assignments are:

Table 30-5. IPSR bit assignments

BIT	NAME	FUNCTION
31:6	**Reserved**	Reserved

BIT	NAME	FUNCTION
5:0	**Exception Number**	This is the number of the current exception: 63-48: Reserved 47: IRQ31 . . . 16: IRQ0 15: SysTick 14: PendSV 13-12: Reserved 11: SVCALL 10-4: Reserved 3: HardFault 2: NMI 1: Reserved 0: Thread mode see Exception types in chapter 30.2.3.2 on page 275

30.2.1.3.8. Execution Program Status Register

The EPSR contains the Thumb state bit.

See the register summary in Table 30-2. Core register set summary on page 264 for ESPR attributes. The bit assignments are:

Table 30-6. EPSR bit assignments

BIT	NAME	FUNCTION
31:25	**Reserved**	Reserved
24	**T**	Thumb state bit
23:0	**Reserved**	Reserved

Attempts by application software to read the EPSR directly using the `MRS` instruction always return zero. Attempts to write the EPSR using the `MSR` instruction are ignored. Fault handlers can examine the EPSR value in the stacked PSR to determine the cause of the fault. See Exception entry and return in chapter 30.2.3.6 on page 279. The following can clear the T bit to 0:

- instructions `BLX`, `BX` and `POP{PC}`
- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry.

Attempting to execute instructions when the T bit is 0 results in a HardFault or lockup.

See Lockup in chapter 30.2.4.1 on page 281 for more information.

30.2.1.3.9. Interruptible-restartable instructions

The interruptible-restartable instructions are `LDM` and `STM`, and the multiply instruction. When an interrupt occurs during the execution of one of these instructions, the processor abandons execution of the instruction.

After servicing the interrupt, the processor restarts execution of the instruction from the beginning.

30.2.1.3.10. Exception mask register

The exception mask register disables the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks or code sequences requiring atomicity.

To disable or re-enable exceptions, use the `MSR` and `MRS` instructions, or the `CPS` instruction, to change the value of PRIMASK. See `MRS` in chapter 30.3.7.6 on page 316, `MSR` in chapter 30.3.7.7 on page 316, and `CPS` in chapter 30.3.7.2 on page 313 for more information.

30.2.1.3.11. Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority.

See the register summary in Table 30-2. Core register set summary on page 264 for its attributes. The bit assignments are:

Figure 30-4. PRIMASK

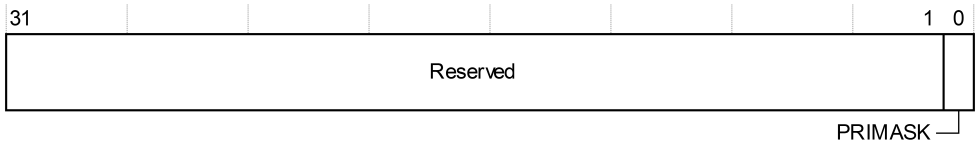


Table 30-7. PRIMASK register bit assignments

BIT	NAME	FUNCTION
31:1	**Reserved**	Reserved
0	**PRIMASK**	1: prevents activation of all exceptions with configurable priority 0: no effect

30.2.1.3.12. Control Register

The CONTROL register controls the stack used when the processor is in Thread mode.

See the register summary in Table 30-2. Core register set summary on page 264 for its attributes. The bit assignments are:

Figure 30-5. CONTROL

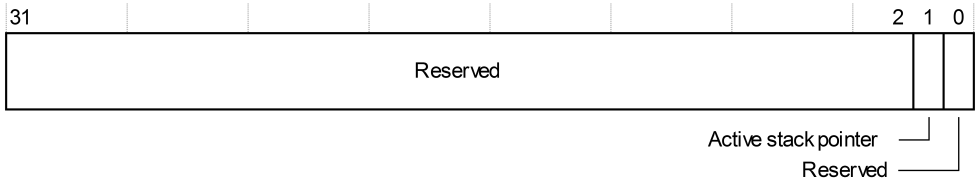


Table 30-8. CONTROL register bit assignments

BIT	NAME	FUNCTION
31:1	**Reserved**	Reserved
0	**Active Stack Pointer**	Defines the current stack: 1: MSP is the current stack pointer 0: PSP is the current stack pointer In Handler mode this bit reads as zero and ignores writes.
0	**Reserved**	Reserved

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms update the CONTROL register.

In an OS environment, it is recommended that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, use the `MSR` instruction to set the Active stack pointer bit to 1, see `MSR` in chapter 30.3.7.7 on page 316.

Note

When changing the stack pointer, software must use an `ISB` instruction immediately after the `MSR` instruction. This ensures that instructions after the ISB execute using the new stack pointer. See `ISB` in chapter 30.3.7.5 on page 315.

30.2.1.4. Exceptions and interrupts

The Cortex-M0 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An interrupt or exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset. See Exception entry in chapter 30.2.3.6.5 on page 279 and Exception return in chapter 30.2.3.6.6 on page 280 for more information.

The NVIC registers control interrupt handling. See Nested Vectored Interrupt Controller in chapter 30.4.2 on page 321 for more information.

30.2.1.5. Data Types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- manages all data memory accesses as little-endian. Instruction memory and Private Peripheral Bus (PPB) accesses are always little-endian. See Memory regions, types and attributes in chapter 30.2.2.1 on page 271 for more information.

30.2.1.6. The Cortex Microcontroller Software Interface Standard

ARM provides the Cortex Microcontroller Software Interface Standard (CMSIS) for programming Cortex-M0 microcontrollers. The CMSIS is an integrated part of the device driver library. For a Cortex-M0 microcontroller system, CMSIS defines:

- a common way to:
  - access peripheral registers
  - define exception vectors
- the names of:
  - the registers of the core peripherals
  - the core exception vectors
- a device-independent interface for RTOS kernels.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M0 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

The CMSIS simplifies software development by enabling the reuse of template code, and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

Note

This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

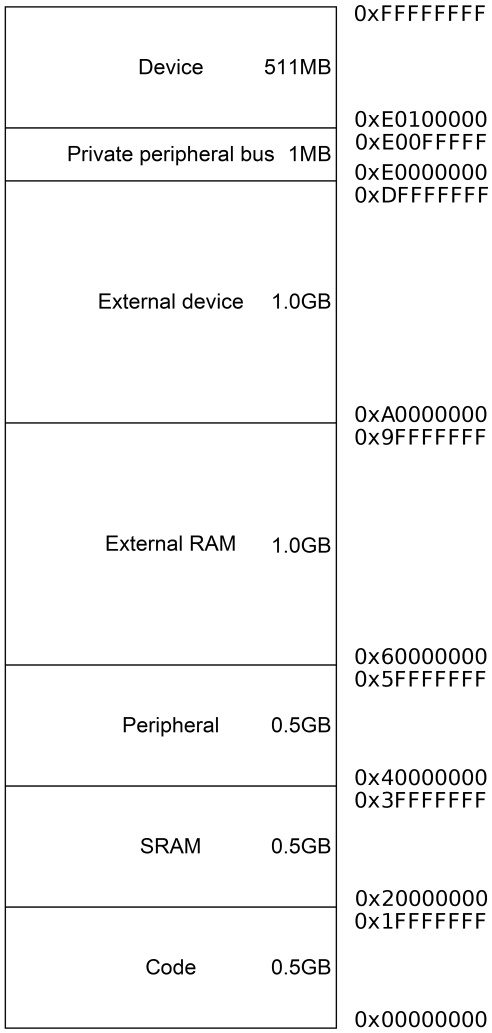
The following sections give more information about the CMSIS:

- Power management programming hints in chapter 30.2.5.5 on page 284
- Intrinsic functions in chapter 30.3.2 onpage 286
- Accessing the Cortex-M0 NVIC registers using CMSIS in chapter 30.4.2.1 on page 322
- NVIC programming hints in chapter 30.4.2.8.1 on page 326

30.2.2. Memory model

This section describes the processor memory map and the behavior of memory accesses. The processor has a fixed memory map that provides up to 4GB of addressable memory. The memory map is:

Figure 30-7. Memory Map



The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see About the Cortex-M0 processor and core peripherals in chapter 30.1.2 on page 261

30.2.2.1. Memory regions, types and attributes

The memory map is split into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

30.2.2.1.1. Normal

The processor can re-order transactions for efficiency, or perform speculative reads.

30.2.2.1.2. Device

The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.

30.2.2.1.3. Strongly-ordered

The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include.

30.2.2.1.4. Execute Never (XN)

Means the processor prevents instruction accesses. A HardFault exception is generated on executing an instruction fetched from an XN region of memory.

30.2.2.2. Memory system ordering of memory accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing any re-ordering does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions, see Software ordering of memory accesses in chapter 30.2.2.4 on page 273.

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses caused by two instructions is:

Figure 30-8. Memory Ordering Restrictions

A1 \ A2	Normal access	Device access		Strongly-ordered access
		Non-shareable	Shareable	
Normal access	-	-	-	-
Device access, non-shareable	-	<	-	<
Device access, shareable	-	-	<	<
Strongly-ordered access	-	<	<	<

Where:

- Means that the memory system does not guarantee the ordering of the accesses.

< Means that accesses are observed in program order, that is, A1 is always observed before A2.

30.2.2.3. Behavior of memory accesses

The behavior of accesses to each region in the memory map is:

Table 30-9. Memory Access Behavior

ADDRESS RANGE	MEMORY REGION	MEMORY TYPE	XN*	DESCRIPTION
0xFFFF FFFF – 0xE010 0000	**Device**	Device	XN	Reserved
0xE00F FFFF – 0xE000 0000	**Private Peripheral Bus**	Strongly -ordered	XN	This region includes the NVIC, System timer, and System Control Block. Only word accesses can be used in this region.
0xDFFF FFFF – 0xA000 0000	**External Device**	Device	XN	External device memory
0x9FFF FFFF – 0x6000 0000	**External RAM**	Normal	-	Executable region for data
0x5FFF FFFF – 0x4000 0000	**Peripheral**	Device	XN	External device memory
0x3FFF FFFF – 0x2000 0000	**SRAM**	Normal	-	Executable region for data. You can also put code here
0x1FFF FFFF - 0x0000 0000	**Code**	Normal	-	Executable region for program code. You can also put data here

* See Memory regions, types and attributes in chapter 30.2.2.1 on page 271 for more information.

The Code, SRAM, and external RAM regions can hold programs.

30.2.2.4. Software ordering of memory accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence
- memory or devices in the memory map might have different wait states
- some memory accesses are buffered or speculative.

Memory system ordering of memory accesses in chapter 30.2.2.2 on page 272 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

30.2.2.4.1. DMB

The Data Memory Barrier (`DMB`) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See `DMB` in chapter 30.3.7.3 on page 314.

30.2.2.4.2. DSB

The Data Synchronization Barrier (`DSB`) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See `DSB` in chapter 30.3.7.4 on page 314.

30.2.2.4.3. ISB

The Instruction Synchronization Barrier (`ISB`) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See `ISB` in chapter 30.3.7.5 on page315.

The following are examples of using memory barrier instructions:

30.2.2.4.4. Vector table

f the program changes an entry in the vector table, and then enables the corresponding exception, use a `DMB` instruction between the operations. This ensures that if the exception is taken immediately after being enabled the processor uses the new exception vector.

30.2.2.4.5. Self-modifying code

If a program contains self-modifying code, use an `ISB` instruction immediately after the code modification in the program. This ensures subsequent instruction execution uses the updated program.

30.2.2.4.6. Memory map switching

If the system contains a memory map switching mechanism, use a `DSB` instruction after switching the memory map. This ensures subsequent instruction execution uses the updated memory map.

Memory accesses to Strongly-ordered memory, such as the System Control Block, do not require the use of `DMB` instructions.

30.2.2.5. Memory endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. Little-endian format in chapter 30.2.2.5.1 on page 274 describes how words of data are stored in memory.

30.2.2.5.1. Little-endian format

In little-endian format, the processor stores the least significant byte (lsbyte) of a word at the lowest-numbered byte, and the most significant byte (msbyte) at the highest-numbered byte. For example:

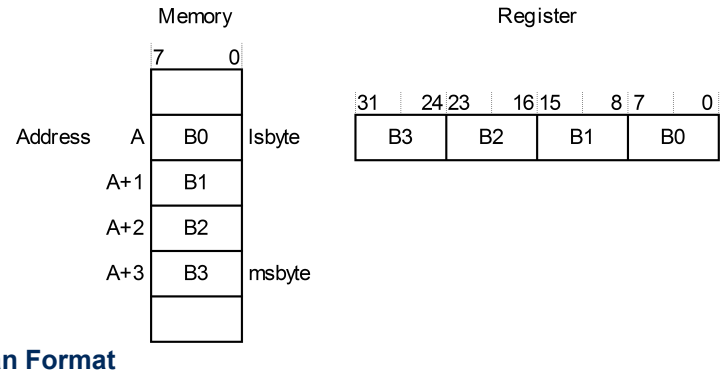


Figure 30-9. Little Endian Format

30.2.3. Exception model

This section describes the exception model.

30.2.3.1. Exception states

Each exception is in one of the following states:

30.2.3.1.1. Inactive

The exception is not active and not pending.

30.2.3.1.2. Pending

The exception is waiting to be serviced by the processor.

An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.

30.2.3.1.3. Active

An exception that is being serviced by the processor but has not completed.

Note

An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.

30.2.3.1.4. Active and pending

The exception is being serviced by the processor and there is a pending exception from the same source.

30.2.3.2. Exception types

The exception types are:

30.2.3.2.1. Reset

Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts in Thread mode.

30.2.3.2.2. NMI

A NonMaskable Interrupt (NMI) can be signaled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be:

- masked or prevented from activation by any other exception
- preempted by any exception other than Reset.

30.2.3.2.3. HardFault

A HardFault is an exception that occurs because of an error during normal or exception processing. HardFaults

have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

30.2.3.2.4. SVCall

A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

30.2.3.2.5. PendSV

PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

30.2.3.2.6. SysTick

A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

30.2.3.2.7. Interrupt (IRQ)

An interrupt, or IRQ, is an exception signaled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor

Table 30-10. Properties of the different exception types

EXCEPTION NUMBER*	IRQ NUMBER*	EXCEPTION TYPE	PRIORITY	VECTOR ADDRESS**	ACTIVATION
1	-	Reset	-3, the highest	0x0000 0004	Asynchronous
2	-14	NMI	-2	0x0000 0008	Asynchronous
3	-13	HardFault	-1	0x0000 000C	Synchronous
4-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable***	0x0000 002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable***	0x0000 0038	Asynchronous
15	-1	SysTick	Configurable***	0x0000 003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable***	0x0000 0040 and above****	Asynchronous

*To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see Interrupt Program Status Register in chapter 30.2.1.3.5 on page 265.

**See Vector table for more information.

***See Interrupt Priority Registers in chapter 30.4.2.6 on page 324.

****Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute additional instructions between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 30-7. PRIMASK register bit assignments on page 268 shows as having configurable priority, see Interrupt Clear-enable Register in chapter 30.4.2.3 on page 323.

For more information about HardFaults, see Fault handling in chapter 30.2.4 on page 281.

30.2.3.3. Exception handlers

The processor handles exceptions using:

30.2.3.3.1. Interrupt Service Routines (ISRs)

Interrupts IRQ0 to IRQ31 are the exceptions handled by ISRs.

30.2.3.3.2. Fault handler

HardFault is the only exception handled by the fault handler.

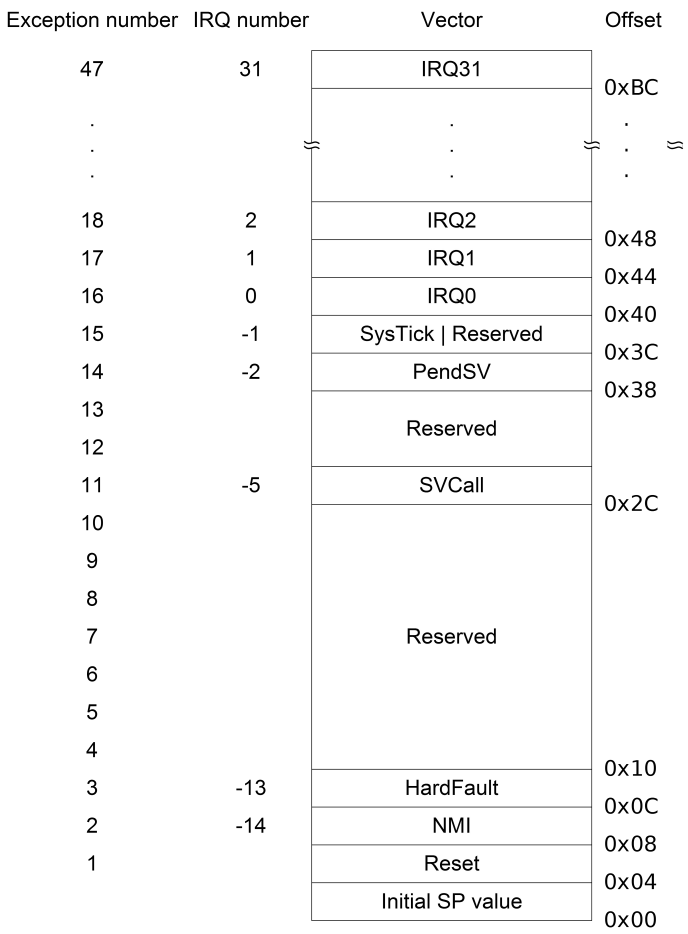
30.2.3.3.3. System handlers

NMI, PendSV, SVCall SysTick, and HardFault are all system exceptions handled by system handlers.

30.2.3.4. Vector table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. Figure 30-6. CONTROL on page 268 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is written in Thumb code.

Figure 30-10. Vector Table



The vector table is fixed at address 0x0000 0000.

30.2.3.5. Exception priorities

As Table 30-7. PRIMASK register bit assignments on page 268 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- System Handler Priority Registers in chapter 30.4.3.7 on page 332
- Interrupt Priority Registers in chapter 30.4.2.6 on page 324.

Note

Configurable priority values are in the range 0-192, in steps of 64. The Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

Assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

30.2.3.6. Exception entry and return

Descriptions of exception handling use the following terms:

30.2.3.6.1. Preemption

When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled.

When one exception preempts another, the exceptions are called nested exceptions. See Exception entry in chapter 30.2.3.6.5 on page 279 for more information.

30.2.3.6.2. Return

This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Exception return in chapter 30.2.3.6.6 on page 280 for more information.

30.2.3.6.3. Tail-chaining

This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.

30.2.3.6.4. Late-arriving

This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved would be the same for both exceptions. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

30.2.3.6.5. Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and

either:

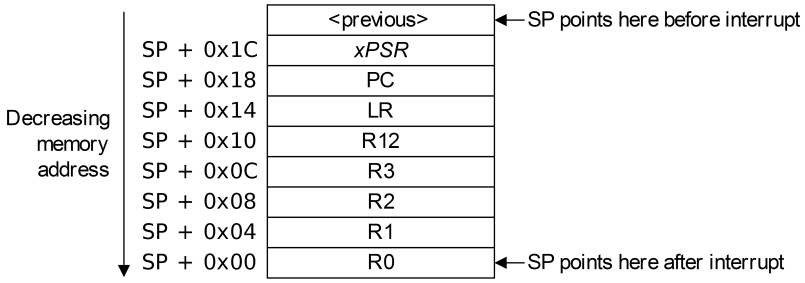
- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the exception being handled.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has greater priority than any limit set by the mask register, see Exception mask register in chapter 30.2.1.3.10 on page 268. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as stacking and the structure of eight data words is referred as a stack frame. The stack frame contains the following information:

Figure 30-11. Exception Entry Stack Contents



Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The stack frame is aligned to a double-word address.

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

The processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

30.2.3.6.6. Exception return

Exception return occurs when the processor is in Handler mode and execution of one of the following instructions attempts to set the PC to an EXC_RETURN value:

- a `POP` instruction that loads the PC
- a `BX` instruction using any register.

The processor saves an EXC_RETURN value to the LR on exception entry. The exception mechanism relies on

this value to detect when the processor has completed an exception handler. Bits[31:4] of an EXC_RETURN value are 0xFFF FFFF. When the processor loads a value matching this pattern to the PC it detects that the operation is a not a normal branch operation and, instead, that the exception is complete. Therefore, it starts the exception return sequence. Bits[3:0] of the EXC_RETURN value indicate the required return stack and processor mode, as Table 30-11. Execution return behavior on page 281 shows.

Table 30-11. Execution return behavior

EXEC_RETURN	DESCRIPTION
0xFFFF FFF1	Return to Handler mode. Exception return gets state from the main stack. Execution uses MSP after return.
0xFFFF FFF9	Return to Thread mode. Exception return gets state from MSP. Execution uses MSP after return.
0xFFFF FFFD	Return to Thread mode. Exception return gets state from PSP. Execution uses PSP after return.
All other values	Reserved

30.2.4. Fault handling

Faults are a subset of exceptions, see Exception model in chapter 30.2.3 on page 275. All faults result in the HardFault exception being taken or cause lockup if they occur in the NMI or HardFault handler. The faults are:

- execution of an `SVC` instruction at a priority equal or higher than SVCall
- execution of a `BKPT` instruction without a debugger attached
- a system-generated bus error on a load or store
- execution of an instruction from an XN memory address
- execution of an instruction from a location for which the system generates a bus fault
- a system-generated bus error on a vector fetch
- execution of an Undefined instruction
- execution of an instruction when not in Thumb-State as a result of the T-bit being previously cleared to 0
- • an attempted load or store to an unaligned address.

Note

Only Reset and NMI can preempt the fixed priority HardFault handler. A HardFault can preempt any exception other than Reset, NMI, or another hard fault.

30.2.4.1. Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers, or if the system generates a bus error when unstacking the PSR on an exception return using the MSP. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until one of the following occurs:

- it is reset
- a debugger halts it

- an NMI occurs and the current lockup is in the HardFault handler.
-

Note

If lockup state occurs in the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

30.2.5. Power management

The Cortex-M0 processor sleep modes reduce power consumption:

- a sleep mode, that stops the processor clock
- a deep sleep mode, that stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see System Control Register in chapter 30.4.3.5 on page 331.

This section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

30.2.5.1. Entering sleep mode

This section describes the mechanisms software can use to put the processor into sleep mode.

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back in to sleep mode.

30.2.5.1.1. Wait for interrupt

The Wait For Interrupt instruction, WFI, causes immediate entry to sleep mode. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode. See `WFI` in chapter 30.3.7.12 on page 320 for more information.

30.2.5.1.2. Wait for event

The Wait For Event instruction, `WFE`, causes entry to sleep mode conditional on the value of a one-bit event register. When the processor executes a `WFE` instruction, it checks the value of the event register:

0: The processor stops executing instructions and enters sleep mode

1: The processor sets the register to zero and continues executing instructions without entering sleep mode.

See `WFE` in chapter 30.3.7.11 on page 319 for more information.

If the event register is 1b, this indicates that the processor must not enter sleep mode on execution of a `WFE` instruction. Typically, this is because of the assertion of an external event, or because another processor in the system has executed a `SEV` instruction, see `SEV` in chapter 30.3.7.9 on page 318. Software cannot access this register directly.

30.2.5.1.3. Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of an exception handler and returns to Thread mode it immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an interrupt occurs.

30.2.5.2. Wakeup from sleep mode

The conditions for the processor to wakeup depend on the mechanism that caused it to enter sleep mode.

30.2.5.2.1. Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK, see Exception mask register in chapter 30.2.1.3.10 on page 268.

30.2.5.2.2. Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry.
- it detects an external event signal, see The external event input in chapter 30.2.5.4 on page 284.
- in a multiprocessor system, another processor in the system executes a `SEV` instruction.

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see System Control Register in chapter 30.4.3.5 on page 331.

30.2.5.3. The Wakeup Interrupt Controller

The Wakeup Interrupt Controller (WIC) is a peripheral that can detect an interrupt and wake the processor from deep sleep mode. The WIC is enabled only when the DEEPSLEEP bit in the SCR is set to 1b, see System Control Register in chapter 30.4.3.5 on page 331.

The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

When the WIC is enabled and the processor enters deep sleep mode, the power management unit in the system can power down most of the Cortex-M0 processor. This has the side effect of stopping the SysTick timer. When the WIC receives an interrupt, it takes a number of clock cycles to wakeup the processor and restore its state, before it can process the interrupt. This means interrupt latency is increased in deep sleep mode.

30.2.5.4. The external event input

The processor provides an external event input signal. This signal is not available on this device.

30.2.5.5. Power management programming hints

ISO/IEC C cannot directly generate the WFI, WFE, and SEV instructions. The CMSIS provides the following intrinsic functions for these instructions:

```
void __WFE(void) // Wait for Event
void __WFI(void) // Wait for Interrupt
void __SEV(void) // Send Event
```

30.3. The Cortex-M0 Instruction Set

This chapter is the reference material for the Cortex-M0 instruction set description in a User Guide. The following sections give general information:

- Instruction set summary in chapter 30.3.1 on page 284.
- Intrinsic functions in chapter 30.3.2 on page 286.
- About the instruction descriptions in chapter 30.3.3 on page 287.

Each of the following sections describes a functional group of Cortex-M0 instructions.

Together they describe all the instructions supported by the Cortex-M0 processor:

- Memory access instructions in chapter 30.3.4 on page 292.
- General data processing instructions in chapter 30.3.5 on page 299.
- Branch and control instructions in chapter 30.3.6 on page 310.
- Miscellaneous instructions in chapter 30.3.7 on page 312.

30.3.1. Instruction set summary

The processor implements a version of the Thumb instruction set. Table 30-12. Cortex-M0 instructions lists the supported instructions.

Note

In Table 30-12. Cortex-M0 instructions:

- angle brackets, <>, enclose alternative forms of the operand
- braces, {}, enclose optional operands and mnemonic parts
- the Operands column is not exhaustive.

For more information on the instructions and operands, see the instruction descriptions.

Table 30-12. Cortex-M0 instructions

MNEMONIC	OPERANDS	BRIEF DESCRIPTION	FLAGS	CHAPTER, PAGE
ADCS	*{Rd,} Rn, Rm*	Add with Carry	N, Z, C, V	Chapter 30.3.5.1, page 300
ADD{S}	*{Rd,} Rn, <Rm|#imm>*	Add	N, Z, C, V	Chapter 30.3.5.1, page 300
ADR	*Rd, label*	PC-relative Address to Register	-	Chapter 30.3.4.1, page 292
ANDS	*{Rd,} Rn, Rm*	Bitwise AND	N, Z	Chapter 30.3.5.1, page 300
ASRS	*{Rd,} Rn, <Rm|#imm>*	Arithmetic Shift Right	N, Z, C	Chapter 30.3.5.3, page 303
B{cc}	*label*	Branch {conditionally}	-	Chapter 30.3.6.1, page 310
BICS	*{Rd,} Rn, Rm*	Bit Clear	N, Z	Chapter 30.3.5.2, page 302
BKPT	*#imm*	Breakpoint	-	Chapter 30.3.7.1, page 312
BL	*label*	Branch with Link	-	Chapter 30.3.6.1, page 310
BLX	*Rm*	Branch indirect with Link	-	Chapter 30.3.6.1, page 310
BX	*Rm*	Branch indirect	-	Chapter 30.3.6.1, page 310
CMN	*Rn, RM*	Compare Negative	N, Z, C, V	Chapter 30.3.5.4, page 304
CMP	*Rn, <Rm|#imm>*	Compare	N, Z, C, V	Chapter 30.3.5.4, page 304
CPSID	*i*	Change Processor State, Disable Interrupts	-	Chapter 30.3.7.2, page 313
CPSIE	*i*	Change Processor State, Enable Interrupts	-	Chapter 30.3.7.2, page 313
DMB	-	Data Memory Barrier	-	Chapter 30.3.7.3, page 314
DSB	-	Data Synchronization Barrier	-	Chapter 30.3.7.4, page 314
EORS	*{Rd,} Rn, Rm*	Exclusive OR	N, Z	Chapter 30.3.5.2, page 302
ISB	-	Instruction Synchronization Barrier	-	Chapter 30.3.7.5, page 315
LDM	*Rn{!}, reglist*	Load Multiple Registers, increment after	-	Chapter 30.3.4.5, page 296
LDR	*Rt, label*	Load Register from PC-relative Address	-	Chapter 30.3.4.4, page 295
LDR	*Rt, [Rn, <Rm|#imm>]*	Load Register with Word	-	Chapter 30.3.4, page 292
LDRB	*Rt, [Rn, <Rm|#imm>]*	Load Register with Byte	-	Chapter 30.3.4, page 292
LDRH	*Rt, [Rn, <Rm|#imm>]*	Load Register with Half-Word	-	Chapter 30.3.4, page 292
LDRSB	*Rt, [Rn, <Rm|#imm>]*	Load Register with signed Byte	-	Chapter 30.3.4, page 292
LDRSH	*Rt, [Rn, <Rm|#imm>]*	Load Register with signed Half-Word	-	Chapter 30.3.4, page 292
LSLS	*{Rd,} Rn, <Rs|#imm>*	Logical Shift Left	N, Z, C	Chapter 30.3.5.3, page 303
LSRS	*{Rd,} Rn, <Rs|#imm>*	Logical Shift Right	N, Z, C	Chapter 30.3.5.3, page 303
MOV{S}	*Rd, Rm*	Move	N, Z	Chapter 30.3.5.5, page 305
MRS	*Rd, spec_reg*	Move to General Register from Special Register	-	Chapter 30.3.7.6, page 316
MSR	*spec_reg, Rm*	Move to special register from General Register	N, Z, C, V	Chapter 30.3.7.7, page 316
MULS	*Rd, Rn, Rm*	Multiply, 32-bit result	N, Z	Chapter 30.3.5.6, page 306
MVNS	*Rd, Rm*	Bitwise NOT	N, Z	Chapter 30.3.5.5, page 305
NOP	-	No Operation	-	Chapter 30.3.7.8, page 317
ORRS	*{Rd,} Rn, Rm*	Logical OR	N, Z	Chapter 30.3.5.2, page 302
POP	*reglist*	Pop registers from stack	-	Chapter 30.3.4.6, page 298
PUSH	*reglist*	Push registers onto stack	-	Chapter 30.3.4.6, page 298
REV	*Rd, Rm*	Byte-Reverse word	-	Chapter 30.3.5.7, page 307
REV16	*Rd, Rm*	Byte-Reverse packed halfwords	-	Chapter 30.3.5.7, page 307
REVSH	*Rd, Rm*	Byte-Reverse signed halfword	-	Chapter 30.3.5.7, page 307

MNEMONIC	OPERANDS	BRIEF DESCRIPTION	FLAGS	CHAPTER, PAGE
RORS	*{Rd,] Rn, Rs*	Rotate Right	N, Z, C	Chapter 30.3.5.3, page 303
RSBS	*{Rd,] Rn, #0*	Reverse Subtract	N, Z, C, V	Chapter 30.3.5.1, page 300
SBCS	*{Rd,] Rn, Rm*	Subtract with Carry	N, Z, C, V	Chapter 30.3.5.1, page 300
SEV	-	Send Event	-	Chapter 30.3.7.9, page 318
STM	*Rn!, reglist*	Store Multiple Registers, Increment After	-	Chapter 30.3.4.5, page 296
STR	*Rt, [Rn, <Rm|#imm>]*	Store Register as word	-	Chapter 30.3.4, page 292
STRB	*Rt, [Rn, <Rm|#imm>]*	Store Register as byte	-	Chapter 30.3.4, page 292
STRH	*Rt, [Rn, <Rm|#imm>]*	Store Register as half word	-	Chapter 30.3.4, page 292
SUB{S}	*Rt, Rn, <Rm|#imm>*	Subtract	N,Z,C,V	Chapter 30.3.5.1, page 300
SVC	*#imm*	Supervisor Call	-	Chapter 30.3.7.10, page 318
SXTB	*Rd, Rm*	Sign extend byte	-	Chapter 30.3.5.8, page 308
SXTH	*Rd, Rm*	Sign extend half word	-	Chapter 30.3.5.8, page 308
TST	*Rd, Rm*	Logical AND based test	N, Z	Chapter 30.3.5.9, page 309
UXTB	*Rd, Rm*	Zero extend a byte	-	Chapter 30.3.5.8, page 308
UXTH	*Rd, Rm*	Zero extend a halfword	-	Chapter 30.3.5.8, page 308
WFE	-	Wait for Event	-	Chapter 30.3.7.11, page 319
WFI	-	Wait for Interrupt	-	Chapter 30.3.7.12, page 320

30.3.2. Intrinsic Functions

ISO/IEC C code cannot directly access some Cortex-M0 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access the relevant instruction.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

Table 30-13. CMSIS intrinsic functions to generate some Cortex-M0 instructions

INSTRUCTION	CMSIS INTRINSIC FUNCTION
CPSIE i	`void __enable_irq (void)`
CPSID i	`void __disable_irq (void)`
ISB	`void __ISB(void)`
DSB	`void __DSB(void)`
DMB	`void __DMB(void)`
NOP	`void __NOP(void)`
REV	`uint32_t REV(uint32_t int value)`
REV16	`uint32_t REV16(uint32_t int value)`
REVSH	`uint32_t REVSH(uint32_t int value)`
SEV	`void __SEV(void)`
WFE	`void __WFE(void)`
WFI	`void __WFI(void)`

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

Table 30-14. CMSIS intrinsic functions to access special registers

SPECIAL REGISTER	ACCESS	CMSIS FUNCTION
PRIMASK	Read	`uint32_t __get_PRIMASK (void)`
	Write	`void __set_PRIMASK (uint32_t value)`
CONTROL	Read	`uint32_t __get_CONTROL (void)`
	Write	`void __set_CONTROL (uint32_t value)`
MSP	Read	`uint32_t __get_MSP (void)`
	Write	`void __set_MSP (uint32_t TopOfMainStack)`
PSP	Read	`uint32_t __get_PSP (void)`
	Write	`void __set_PSP (uint32_t TopOfMainStack)`

30.3.3. About the Instruction Descriptions

The following sections give more information about using the instructions:

- Operands in chapter 30.3.3.1 on page 287
- Restrictions when using PC or SP in chapter 30.3.3.2on page 287
- Shift Operations in chapter 30.3.3.3 on page 288
- Address alignment in chapter 30.3.3.4 on page 290
- PC-relative expressions in chapter 30.3.3.5 on page 290
- Conditional execution in chapter 30.3.3.6 on page 290

30.3.3.1. Operands

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the other operands.

30.3.3.2. Restrictions when using PC or SP

Many instructions are unable to use, or have restrictions on whether you can use, the Program Counter (PC) or Stack Pointer (SP) for the operands or destination register. See instruction descriptions for more information.

Note

When you update the PC with a `BX`, `BLX`, or `POP` instruction, bit[0] of any address must be 1 for correct execution. This is because this bit indicates the destination instruction set, and the Cortex-M0 processor only supports Thumb instructions. When a `BL` or `BLX` instruction writes the value of bit[0] into the LR it is automatically assigned the value 1.

30.3.3.3. Shift Operations

Register shift operations move the bits in a register left or right by a specified number of bits, the shift length. Register shift can be performed directly by the instructions `ASR`, `LSR`, `LSL`, and `ROR` and the result is written to a destination register.

The permitted shift lengths depend on the shift type and the instruction, see the individual instruction description. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions, Rm is the register containing the value to be shifted, and n is the shift length.

30.3.3.3.1. ASR

Arithmetic shift right by n bits moves the left-hand 32-n bits of the register *Rm*, to the right by n places, into the right-hand 32-n bits of the result, and it copies the original bit[31] of the register into the left-hand n bits of the result. See Figure 30-4. PRIMASK on page 268.

You can use the `ASR` operation to divide the signed value in the register *Rm* by 2n, with the result being rounded towards negative-infinity.

When the instruction is `ASRS` the carry flag is updated to the last bit shifted out, bit[n-1], of the register *Rm*.

Note

- If n is 32 or more, then all the bits in the result are set to the value of bit[31] of *Rm*.
- If n is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of *Rm*.

Figure 30-12. ASR #3

30.3.3.3.2. LSR

Logical shift right by n bits moves the left-hand 32-n bits of the register *Rm*, to the right by n places, into the right-hand 32-n bits of the result, and it sets the left-hand n bits of the result to 0. See Figure 30-2. Core Registers on page 264.

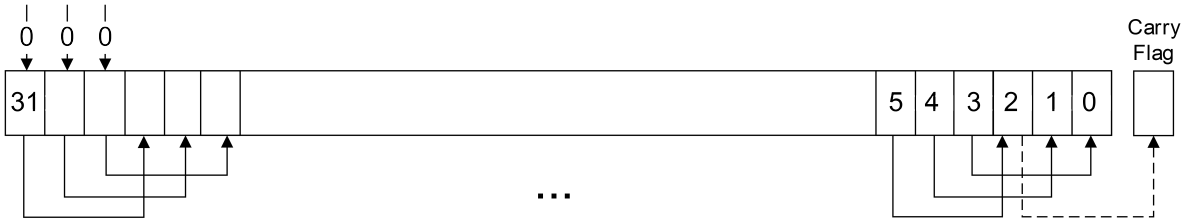
You can use the `LSR` operation to divide the value in the register *Rm* by 2n, if the value is regarded as an unsigned integer.

When the instruction is `LSRS`, the carry flag is updated to the last bit shifted out, bit[n-1], of the register *Rm*.

Note

- If n is 32 or more, then all the bits in the result are cleared to 0.
- If n is 33 or more and the carry flag is updated, it is updated to 0.

Figure 30-13. LSR #3



30.3.3.3.3. LSL

Logical shift left by n bits moves the right-hand 32-n bits of the register *Rm*, to the left by n places, into the left-hand 32-n bits of the result, and it sets the right-hand n bits of the result to 0. See Figure 30-3. PSR on page 265.

You can use the `LSL` operation to multiply the value in the register *Rm* by 2n, if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is `LSLS` the carry flag is updated to the last bit shifted out, bit[32-n], of the register *Rm*. These instructions do not affect the carry flag when used with `LSL #0`.

Note

- If n is 32 or more, then all the bits in the result are cleared to 0.
- If n is 33 or more and the carry flag is updated, it is updated to 0.

Figure 30-14. LSL #3

30.3.3.3.4. ROR

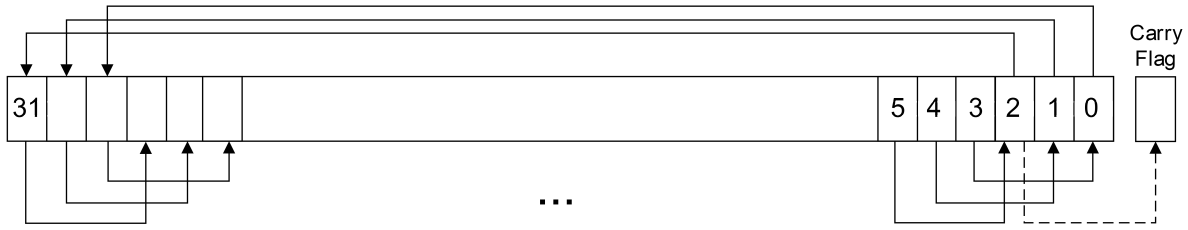
Rotate right by n bits moves the left-hand 32-n bits of the register *Rm*, to the right by n places, into the right-hand 32-n bits of the result, and it moves the right-hand n bits of the register into the left-hand n bits of the result. See Figure 30-1. Cortex-M0 implementation on page 261.

When the instruction is `RORS` the carry flag is updated to the last bit rotation, bit[n-1], of the register *Rm*.

Note

- If n is 32, then the value of the result is same as the value in *Rm*, and if the carry flag is updated, it is updated to bit[31] of Rm.
- `ROR` with shift length, n, greater than 32 is the same as `ROR` with shift length n-32.

Figure 30-15. ROR #3



30.3.3.4. Address alignment

An aligned access is an operation where a word-aligned address is used for a word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

There is no support for unaligned accesses on the Cortex-M0 processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

30.3.3.5. PC-relative expressions

A PC-relative expression or label is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset. The assembler calculates the required offset from the label and the address of the current instruction. If the offset is too big, the assembler produces an error.

Note

- For most instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form *[PC, #imm]*.

30.3.3.6. Conditional execution

Most data processing instructions update the condition flags in the Application Program Status Register (APSR) according to the result of the operation, see Application Program Status Register in chapter 30.2.1.3.6 on page 266. Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

You can execute a conditional branch instruction, based on the condition flags set in another instruction, either:

- immediately after the instruction that updated the flags
- after any number of intervening instructions that have not updated the flags.

On the Cortex-M0 processor, conditional execution is available by using conditional branches.

This section describes:

- The condition flags in chapter 30.3.3.6.1 on page 291
- Condition code suffixes in chapter 30.3.3.6.2 on page 291

30.3.3.6.1. The condition flags

The APSR contains the following condition flags:

N Set to 1 when the result of the operation was negative, cleared to 0 otherwise.

Z Set to 1 when the result of the operation was zero, cleared to 0 otherwise.

C Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.

V Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR see Program Status Register in chapter 30.2.1.3.5 on page 265

A carry occurs:

- if the result of an addition is greater than or equal to 2^32
- if the result of a subtraction is positive or zero
- as the result of a shift or rotate instruction.

Overflow occurs when the sign of the result, in bit[31], does not match the sign of the result had the operation been performed at infinite precision, for example:

- if adding two negative values results in a positive value
- if adding two positive values results in a negative value
- if subtracting a positive value from a negative value generates a positive value
- if subtracting a negative value from a positive value generates a negative value.

The Compare operations are identical to subtracting, for `CMP`, or adding, for `CMN`, except that the result is discarded. See the instruction descriptions for more information.

30.3.3.6.2. Condition code suffixes

Conditional branch is shown in syntax descriptions as *B{cond}*. A branch instruction with a condition code is only taken if the condition code flags in the APSR meet the specified condition, otherwise the branch instruction is ignored. Table 30-15. Condition code suffixes on page 291 shows the condition codes to use.

Table 30-15. Condition code suffixes on page 291 also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 30-15. Condition code suffixes

SUFFIX	FLAGS	MEANING
`EQ`	Z = 1	Equal, last flag setting result was zero
`NE`	Z = 0	Not equal, last flag setting result was non-zero
`CS or HS`	C = 1	Higher or same, unsigned

SUFFIX	FLAGS	MEANING
CC or L0	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C =1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z =1 and N != V	Less than or equal, signed
AL	Can have any value	Always. This is the default when no suffix is specified

30.3.4. Memory access instructions

Table 30-16. Memory access instructions on page 292 shows the memory access instructions:

Table 30-16. Memory access instructions

MNEMONIC	BRIEF DESCIPTION	SEE
ADR	Generate PC-Relative Address	ADR in chapter 30.3.4.1on page 292
LDM	Load Multiple Registers	LDM and STM in chapter 30.3.4.5 on page 296
LDR{type}	Load Register using Immediate Offset	LDR and STR, immediate offset in chapter 30.3.4.2 on page 293
LDR{type}	Load Register Using Register Offset	LDR and STR, register offset in chapter 30.3.4.3 on page 294
LDR	Load Register from PC-Relative Address	LDR, PC-relative in chapter 30.3.4.4 on page 295
POP	Pop Registers From Stack	PUSH and POP in chapter 30.3.4.6 on page 298
PUSH	Push Registers To Stack	PUSH and POP in chapter 30.3.4.6 on page 298
STM	Store Multiple Registers	LDM and STM in chapter 30.3.4.5 on page 296
STR{type}	Store Register Using Immediate Offset	LDR and STR in chapter 30.3.4.2 on page 293
STR{type}	Store Register Using Register Offset	LDR and STR in chapter 30.3.4.3 on page 294

30.3.4.1. ADR

Generates a PC-relative address.

30.3.4.1.1. Syntax

ADR *Rd, label*

where:

Rd is the destination register.

Label is a PC-relative expression. See PC-relative expressions in chapter 30.3.3.5 on page 290.

30.3.4.1.2. Operation

`ADR` generates an address by adding an immediate value to the PC, and writes the result to the destination register.

`ADR` facilitates the generation of position-independent code, because the address is PC-relative.

If you use `ADR` to generate a target address for a `BX` or `BLX` instruction, you must ensure that bit[0] of the address you generate is set to 1 for correct execution.

30.3.4.1.3. Restrictions

In this instruction *Rd* must specify R0-R7. The data-value addressed must be word aligned and within 1020 bytes of the current PC.

30.3.4.1.4. Condition flags

This instruction does not change the flags.

30.3.4.1.5. Examples

```
ADR R1, TextMessage           ; Write address value of a location labeled as
                              ; TextMessage to R1
ADR R3, [PC,#996]             ; Set R3 to value of PC + 996.
```

30.3.4.2. LDR and STR, immediate offset

Load and Store with immediate offset.

30.3.4.2.1. Syntax

`LDR` *Rt*`, [<`*Rn* `|` *SP*`> {, #`*imm*`}]`

`LDR<B|H>` *Rt*`, [`*Rn* `{, #`*imm*`}]`

`STR` *Rt*`, [<`*Rn* `| SP>, {,#`*imm*`}]`

`STR<B|H>` *Rt*`, [`*Rn* `{,#`*imm*`}]`

where:

Rt is the register to load or store.

Rn is the register on which the memory address is based.

imm is an offset from *Rn*. If *imm* is omitted, it is assumed to be zero.

30.3.4.2.2. Operation

LDR, LDRB and LDRH instructions load the register specified by *Rt* with either a word, byte or halfword data value from memory. Sizes less than word are zero extended to 32-bits before being written to the register specified by *Rt*.

STR, STRB and STRH instructions store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* in to memory. The memory address to load from or store to is the sum of the value in the register specified by either *Rn* or SP and the immediate value *imm*.

30.3.4.2.3. Restrictions

In these instructions:

- *Rt* and *Rn* must only specify R0-R7.
- *imm* must be between:
  - 0 and 1020 and an integer multiple of four for LDR and STR using SP as the base register
  - 0 and 124 and an integer multiple of four for LDR and STR using R0-R7 as the base register
  - 0 and 62 and an integer multiple of two for LDRH and STRH
  - 0 and 31 for LDRB and STRB
- The computed address must be divisible by the number of bytes in the transaction, see Address alignment in chapter 30.3.3.4 on page 290.

30.3.4.2.4. Condition flags

These instructions do not change the flags.

30.3.4.2.5. Examples

```
LDR R4, [R7]                  ; Loads R4 from the address in R7.
STR R2, [R0,#const-struc]     ; const-struc is an expression evaluating
                              ; to a constant in the range 0-1020.
```

30.3.4.3. LDR and STR, register offset

Load and Store with register offset.

30.3.4.3.1. Syntax

LDR *Rt*, [*Rn*, *Rm*]

LDR<B|H> *Rt*, [*Rn*, *Rm*]

LDR<SB|SH> *Rt*, [*Rn*, *Rm*]

`STR` *Rt*, `[`*Rn*, *Rm*`]`

`STR<B|H>` *Rt*, `[`*Rn*, *Rm*`]`

where:

Rt is the register to load or store.

Rn is the register on which the memory address is based.

Rm is a register containing a value to be used as the offset.

30.3.4.3.2. Operation

`LDR`, `LDRB`, `LDRH`, `LDRSB` and `LDRSH` load the register specified by *Rt* with either a word, zero extended byte, zero extended halfword, sign extended byte or sign extended halfword value from memory.

`STR`, `STRB` and `STRH` store the word, least-significant byte or lower halfword contained in the single register specified by *Rt* into memory.

The memory address to load from or store to is the sum of the values in the registers specified by *Rn* and *Rm*.

30.3.4.3.3. Restrictions

In these instructions:

- *Rt*, *Rn*, and *Rm* must only specify R0-R7.
- the computed memory address must be divisible by the number of bytes in the load or store, see Address alignment in chapter 30.3.3.4 on page 290.

30.3.4.3.4. Condition flags

These instructions do not change the flags.

30.3.4.3.5. Examples

```
STR R0, [R5, R1]        ; Store value of R0 into an address equal to
                        ; sum of R5 and R1
LDRSH R1, [R2, R3]      ; Load a halfword from the memory address
                        ; specified by (R2 + R3), sign extend to 32-bits
                        ; and write to R1.
```

30.3.4.4. LDR, PC-relative

Load register (literal) from memory.

30.3.4.4.1. Syntax

`LDR` *Rt, label*

where:

Rt is the register to load.

label is a PC-relative expression. See PC-relative expressions in chapter 30.3.3.5 on page 290.

30.3.4.4.2. Operation

Loads the register specified by *Rt* from the word in memory specified by label.

30.3.4.4.3. Restrictions

In these instructions, label must be within 1020 bytes of the current PC and word aligned.

30.3.4.4.4. Condition flags

These instructions do not change the flags.

30.3.4.4.5. Examples

```
LDR R0, LookUpTable           ; Load R0 with a word of data from an address
                              ; labeled as LookUpTable.
LDR R3, [PC, #100]            ; Load R3 with memory word at (PC + 100).
```

30.3.4.5. LDM and STM

Load and Store Multiple registers.

30.3.4.5.1. Syntax

`LDM` *Rn*`{!},` *reglist*

`STM` *Rn*`!,` *reglist*

where:

Rn is the register on which the memory addresses are based.

! writeback suffix.

reglist is a list of one or more registers to be loaded or stored, enclosed in braces.

It can contain register ranges. It must be comma separated if it contains more than one register or register range, see Examples in chapter 30.3.4.5.5 on page 297.

`LDMIA` and `LDMFD` are synonyms for `LDM`. `LDMIA` refers to the base register being Incremented After each access. `LDMFD` refers to its use for popping data from Full Descending stacks.

`STMIA` and `STMEA` are synonyms for `STM`. `STMIA` refers to the base register being Incremented After each access. `STMEA` refers to its use for pushing data onto Empty Ascending stacks.

30.3.4.5.2. Operation

`LDM` instructions load the registers in reglist with word values from memory addresses based on *Rn*.

`STM` instructions store the word values in the registers in reglist to memory addresses based on *Rn*.

The memory addresses used for the accesses are at 4-byte intervals ranging from the value in the register specified by *Rn* to the value in the register specified by *Rn* + 4 * (n-1), where n is the number of registers in *reglist*. The accesses happens in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address. If the writeback suffix is specified, the value in the register specified by *Rn* + 4 *n is written back to the register specified by *Rn*.

30.3.4.5.3. Restrictions

In these instructions:

- •*reglist* and *Rn* are limited to R0-R7.
- the writeback suffix must always be used unless the instruction is an `LDM` where *reglist* also contains *Rn*, in which case the writeback suffix must not be used.
- the value in the register specified by *Rn* must be word aligned. See Address alignment in chapter 30.3.3.4 on page 290.
- for `STM`, if *Rn* appears in *reglist*, then it must be the first register in the list.

30.3.4.5.4. Condition flags

These instructions do not change the flags.

30.3.4.5.5. Examples

```
LDM R0,{R0,R3,R4}              ; LDMIA is a synonym for LDM
STMIA R1!,{R2-R4,R6}
```

30.3.4.5.6. Incorrect examples

```
STM R5!,{R4,R5,R6}             ; Value stored for R5 is unpredictable
LDM R2,{}                      ; There must be at least one register in the list
```

30.3.4.6. PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

30.3.4.6.1. Syntax

`PUSH` *reglist*

`POP` *reglist*

where:

reglist is a non-empty list of registers, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

30.3.4.6.2. Operation

`PUSH` stores registers on the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

`POP` loads registers from the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

`PUSH` uses the value in the SP register minus four as the highest memory address, `POP` uses the value in the SP register as the lowest memory address, implementing a full-descending stack. On completion, `PUSH` updates the SP register to point to the location of the lowest store value, `POP` updates the SP register to point to the location above the highest location loaded.

If a `POP` instruction includes PC in its *reglist*, a branch to this location is performed when the `POP` instruction has completed. Bit[0] of the value read for the PC is used to update the APSR T-bit. This bit must be 1 to ensure correct operation.

30.3.4.6.3. Restrictions

In these instructions:

- *reglist* must use only R0-R7.
- The exception is LR for a `PUSH` and PC for a `POP`.

30.3.4.6.4. Condition flags

These instructions do not change the flags.

30.3.4.6.5. Examples

```
PUSH {R0,R4-R7}             ; Push R0,R4,R5,R6,R7 onto the stack
PUSH {R2,LR}                ; Push R2 and the link-register onto the stack
POP {R0,R6,PC}              ; Pop r0,r6 and PC from the stack, then branch to
                            ; the new PC.
```

30.3.5. General data processing instructions

Table 30-17. Data processing instructions on page 299 shows the data access instructions:

Table 30-17. Data processing instructions

MNEMONIC	BRIEF DESCIPTION	SEE
ADCS	Add with Carry	ADC, ADD, RSB, SBC, and SUB in chapter 30.3.5.1on page 300
ADD{S}	Add	ADC, ADD, RSB, SBC, and SUB in chapter 30.3.5.1on page 300
ANDS	Logical AND	AND, ORR, EOR, and BIC on page
ASRS	Arithmetic Shift Right	ASR, LSL, LSR, and ROR in chapter 30.3.5.3 on page 303
BICS	Bit Clear	AND, ORR, EOR, and BIC in chapter 30.3.5.2 on page 302
CMN	Compare Negative	CMP and CMN in chapter 30.3.5.4 on page 304
CMP	Compare	CMP and CMN in chapter 30.3.5.4 on page 304
EORS	Exclusive OR	AND, ORR, EOR, and BIC in chapter 30.3.5.2 on page 302
LSLS	Logical Shift Left	ASR, LSL, LSR, and ROR in chapter 30.3.5.3 on page 303
LSRS	Logical Shift Right	ASR, LSL, LSR, and ROR in chapter 30.3.5.3 on page 303
MOV{S}	Move	MOV and MVN in chapter 30.3.5.5 on page 305
MULS	Multiply	MULS in chapter 30.3.5.6 on page 306
MVNS	Move NOT	MOV and MVN in chapter 30.3.5.5 on page 305
ORRS	Logical OR	AND, ORR, EOR, and BIC in chapter 30.3.5.2 on page 302
REV	Reverse Byte Order In A Word	REV, REV16, and REVSH in chapter 30.3.5.7 on page 307
REV16	Reverse Byte Order In each Half Word	REV, REV16, and REVSH in chapter 30.3.5.7 on page 307
REVSH	Reverse Byte Order In Bottom Half Word and Sign Extend	REV, REV16, and REVSH in chapter 30.3.5.7 on page 307
RORS	Rotate Right	ASR, LSL, LSR, and ROR in chapter 30.3.5.3 on page 303
RSBS	Reverse Subtract	ADC, ADD, RSB, SBC, and SUB in chapter 30.3.5.1on page 300
SBCS	Subtract with Carry	ADC, ADD, RSB, SBC, and SUB in chapter 30.3.5.1on page 300
SUBS	Subtract	ADC, ADD, RSB, SBC, and SUB in chapter 30.3.5.1on page 300
SXTB	Sign Extend A Byte	SXT and UXT in chapter 30.3.5.8 on page 308
SXTH	Sign Extend a Halfword	SXT and UXT in chapter 30.3.5.8 on page 308
UXTB	Zero Extend a Byte	SXT and UXT in chapter 30.3.5.8 on page 308

MNEMONIC	BRIEF DESCIPTION	SEE
UXTH	Zero Extend a Halfword	SXT and UXT in chapter 30.3.5.8 on page 308
TST	Test	TST in chapter 30.3.5.9 on page 309

30.3.5.1. ADC, ADD, RSB, SBC, and SUB

Add with carry, Add, Reverse Subtract, Subtract with carry, and Subtract.

30.3.5.1.1. Syntax

`ADCS {`*Rd*`,} `*Rn*`, `*Rm*

`ADD{S} {`*Rd*`,} `*Rn*`, <`*Rm*`|#`*imm*`>`

`RSBS {`*Rd*`,} `*Rn*`, `*Rm*`, #0`

`SBCS {`*Rd*`,} `*Rn*`, `*Rm*

`SUB{S} {`*Rd*`,} `*Rn*`, <`*Rm*`|#`*imm*`>`

Where:

S causes an ADD or SUB instruction to update flags

Rd specifies the result register

Rn specifies the first source register

Rm specifies the second source register

imm specifies a constant immediate value.

When the optional *Rd* register specifier is omitted, it is assumed to take the same value as *Rn*, for example `ADDS R1,R2` is identical to `ADDS R1,R1,R2`.

30.3.5.1.2. Operation

The `ADCS` instruction adds the value in *Rn* to the value in *Rm*, adding a further one if the carry flag is set, places the result in the register specified by *Rd* and updates the N, Z, C, and V flags.

The `ADD` instruction adds the value in *Rn* to the value in *Rm* or an immediate value specified by *imm* and places the result in the register specified by *Rd*.

The `ADDS` instruction performs the same operation as `ADD` and also updates the N, Z, C and V flags.

The `RSBS` instruction subtracts the value in *Rn* from zero, producing the arithmetic negative of the value, and places the result in the register specified by *Rd* and updates the N, Z, C and V flags.

The `SBCS` instruction subtracts the value of *Rm* from the value in *Rn*, deducts a further one if the carry flag is set. It places the result in the register specified by *Rd* and updates the N, Z, C and V flags.

The `SUB` instruction subtracts the value in *Rm* or the immediate specified by *imm*. It places the result in the register specified by *Rd*.

The `SUBS` instruction performs the same operation as `SUB` and also updates the N, Z, C and V flags.

Use ADC and SBC to synthesize multiword arithmetic, see Examples in chapter 30.3.5.1.4 on page 301.

See also ADR in chapter 30.3.4.1 on page 292.

30.3.5.1.3. Restrictions

Table 30-18. ADC, ADD, RSB, SBC, and SUB operand restrictions on page 301 lists the legal combinations of register specifiers and immediate values that can be used with each instruction.

Table 30-18. ADC, ADD, RSB, SBC, and SUB operand restrictions

INSTRUCTION	RD	Rn	Rm	imm	RESTRICTIONS
ADCS	R0-R7	R0-R7	R0-R7	-	*Rd* and *Rn* must specify the same register
ADD	R0-R15	R0-R15	R0-PC	-	Rd and Rn must specify the same register Rn and Rm must not both specify PC
	R0-R7	SP or PC	-	0-1020	Immediate value must be an integer multiple of four
	SP	SP	-	0-508	Immediate value must be an integer multiple of four
ADDS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	*Rd* and *Rn* must specify the same register
	R0-R7	R0-R7	R0-R7	-	-
RSBS	R0-R7	R0-R7	-	-	-
SBCS	R0-R7	R0-R7	R0-R7	-	*Rd* and *Rn* must specify the same register
SUB	SP	SP	-	0-508	Immediate value must be an integer multiple of four
SUBS	R0-R7	R0-R7	-	0-7	-
	R0-R7	R0-R7	-	0-255	*Rd* and *Rn* must specify the same register
	R0-R7	R0-R7	R0-R7	-	-

30.3.5.1.4. Examples

Example below shows two instructions that add a 64-bit integer contained in R0 and R1 to another 64-bit integer contained in R2 and R3, and place the result in R0 and R1.

```
ADDS R0, R0, R2          ; add the least significant words
ADCS R1, R1, R3          ; add the most significant words with carry
```

Multiword values do not have to use consecutive registers. Example below shows instructions that subtract a 96-bit integer contained in R1, R2, and R3 from another contained in R4, R5, and R6. The example stores the result in R4, R5, and R6.

```
SUBS R4, R4, R1          ; subtract the least significant words
SBCS R5, R5, R2          ; subtract the middle words with carry
SBCS R6, R6, R3          ; subtract the most significant words with carry
```

Example below the RSBS instruction used to perform a 1's complement of a single register.

```
RSBS R7, R7, #0          ; subtract R7 from zero
```

30.3.5.2. AND, ORR, EOR, and BIC

Logical AND, OR, Exclusive OR, and Bit Clear.

30.3.5.2.1. Syntax

ANDS {*Rd*,} *Rn*, *Rm*

ORRS {*Rd*,} *Rn*, *Rm*

EORS {*Rd*,} *Rn*, *Rm*

BICS {*Rd*,} *Rn*, *Rm*

where:

Rd is the destination register.

Rn is the register holding the first operand and is the same as the destination register.

Rm second register.

30.3.5.2.2. Operation

The AND, EOR, and ORR instructions perform bitwise AND, exclusive OR, and inclusive OR operations on the values in *Rn* and *Rm*.

The BIC instruction performs an AND operation on the bits in *Rn* with the logical negation of the corresponding bits in the value of *Rm*.

The condition code flags are updated on the result of the operation, see Condition flags in chapter 30.3.4.6.4 on page 298.

30.3.5.2.3. Restrictions

In these instructions, *Rd*, *Rn*, and *Rm* must only specify R0-R7.

30.3.5.2.4. Condition flags

These instructions:

- update the N and Z flags according to the result
- do not affect the C or V flag.

30.3.5.2.5. Examples

```
ANDS R2, R2, R1
ORRS R2, R2, R5
ANDS R5, R5, R8
EORS R7, R7, R6
BICS R0, R0, R1
```

30.3.5.3. ASR, LSL, LSR, and ROR

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, and Rotate Right.

30.3.5.3.1. Syntax

`ASRS {`*Rd*`,} `*Rm*`, `*Rs*

`ASRS {`*Rd*`,} `*Rm*`, #`*imm*

`LSLS {`*Rd*`,} `*Rm*`, `*Rs*

`LSLS {`*Rd*`,} `*Rm*`, #`*imm*

`LSRS {`*Rd*`,} `*Rm*`, `*Rs*

`LSRS {`*Rd*`,} `*Rm*`, #`*imm*

`RORS {`*Rd*`,} `*Rm*`, `*Rs*

where:

Rd is the destination register. If *Rd* is omitted, it is assumed to take the same value as *Rm*.

Rm is the register holding the value to be shifted.

Rs is the register holding the shift length to apply to the value in *Rm*.

imm is the shift length. The range of shift length depends on the instruction:

`ASR` shift length from 1 to 32

`LSL` shift length from 0 to 31

`LSR` shift length from 1 to 32.

Note

`MOVS` *Rd*, *Rm* is a pseudonym for `LSLS` *Rd*, *Rm*, *#0*.

30.3.5.3.2. Operation

`ASR`, `LSL`, `LSR`, and `ROR` perform an arithmetic-shift-left, logical-shift-left, logical-shift-right or a right-rotation of the bits in the register *Rm* by the number of places specified by the immediate *imm* or the value in the least-significant byte of the register specified by *Rs*.

For details on what result is generated by the different instructions, see Shift Operations in chapter 30.3.3.3 on page 288.

30.3.5.3.3. Restrictions

In these instructions, *Rd*, *Rm*, and *Rs* must only specify R0-R7. For non-immediate instructions, *Rd* and *Rm* must specify the same register.

30.3.5.3.4. Condition flags

These instructions update the N and Z flags according to the result.

The C flag is updated to the last bit shifted out, except when the shift length is 0, see Shift Operations in chapter 30.3.3.3 on page 288. The V flag is left unmodified.

30.3.5.3.5. Examples

```
ASRS R7, R5, #9          ; Arithmetic shift right by 9 bits
LSLS R1, R2, #3          ; Logical shift left by 3 bits with flag update
LSRS R4, R5, #6          ; Logical shift right by 6 bits
RORS R4, R4, R6          ; Rotate right by the value in the bottom byte of R6.
```

30.3.5.4. CMP and CMN

Compare and Compare Negative.

30.3.5.4.1. Syntax

`CMN` *Rn, Rm*

`CMP` *Rn, #imm*

`CMP` *Rn, Rm*

where:

Rn is the register holding the first operand.

Rm is the register to compare with.

imm is the immediate value to compare with.

30.3.5.4.2. Operation

These instructions compare the value in a register with either the value in another register or an immediate value. They update the condition flags on the result, but do not write the result to a register.

The `CMP` instruction subtracts either the value in the register specified by *Rm*, or the immediate *imm* from the

value in *Rn* and updates the flags. This is the same as a `SUBS` instruction, except that the result is discarded.

The `CMN` instruction adds the value of *Rm* to the value in *Rn* and updates the flags. This is the same as an `ADDS` instruction, except that the result is discarded.

30.3.5.4.3. Restrictions

For the:

- `CMN` instruction *Rn*, and *Rm* must only specify R0-R7.
- `CMP` instruction:
  - *Rn* and *Rm* can specify R0-R14
  - immediate must be in the range 0-255.

30.3.5.4.4. Condition flags

These instructions update the N, Z, C and V flags according to the result

30.3.5.4.5. Examples

```
CMP R2, R9
CMN R0, R2
```

30.3.5.5. MOV and MVN

Move and Move NOT.

30.3.5.5.1. Syntax

`MOV{S}` *Rd, Rm*

`MOVS` *Rd, #imm*

`MVNS` *Rd, Rm*

where:

S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see Conditional execution in chapter 30.3.3.6 on page 290.

Rd is the destination register.

Rm is a register.

imm is any value in the range 0-255.

30.3.5.5.2. Operation

The `MOV` instruction copies the value of *Rm* into *Rd*.

The `MOVS` instruction performs the same operation as the `MOV` instruction, but also updates the N and Z flags.

The `MVNS` instruction takes the value of *Rm*, performs a bitwise logical negate operation on the value, and places the result into *Rd*.

30.3.5.5.3. Restrictions

In these instructions, *Rd*, and *Rm* must only specify R0-R7.

When *Rd* is the PC in a `MOV` instruction:

- Bit[0] of the result is discarded.
- A branch occurs to the address created by forcing bit[0] of the result to 0. The T-bit remains unmodified.
-

Note

Though it is possible to use `MOV` as a branch instruction, ARM strongly recommends the use of a `BX` or `BLX` instruction to branch for software portability.

30.3.5.5.4. Condition flags

If S is specified, these instructions:

- update the N and Z flags according to the result
- do not affect the C or V flags.

30.3.5.5.5. Example

```
MOVS R0, #0x000B        ; Write value of 0x000B to R0, flags get updated
MOVS R1, #0x0           ; Write value of zero to R1, flags are updated
MOV R10, R12            ; Write value in R12 to R10, flags are not updated
MOVS R3, #23            ; Write value of 23 to R3
MOV R8, SP              ; Write value of stack pointer to R8
MVNS R2, R0             ; Write inverse of R0 to the R2 and update flags
```

30.3.5.6. MULS

Multiply using 32-bit operands, and producing a 32-bit result.

30.3.5.6.1. Syntax

`MULS` *Rd, Rn, Rm*

where:

Rd is the destination register.

Rn, *Rm* are registers holding the values to be multiplied.

30.3.5.6.2. Operation

The `MUL` instruction multiplies the values in the registers specified by *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*. The condition code flags are updated on the result of the operation, see Conditional execution in chapter 30.3.3.6 on page 290.

The results of this instruction does not depend on whether the operands are signed or unsigned.

30.3.5.6.3. Restrictions

In this instruction:

- *Rd*, *Rn*, and *Rm* must only specify R0-R7
- *Rd* must be the same as *Rm*.

30.3.5.6.4. Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

30.3.5.6.5. Examples

```
MULS R0, R2, R0          ; Multiply with flag update, R0 = R0 x R2
```

30.3.5.7. REV, REV16, and REVSH

Reverse bytes.

30.3.5.7.1. Syntax

`REV` *Rd, Rn*

`REV16` *Rd, Rn*

`REVSH` *Rd, Rn*

where:

Rd is the destination register.

Rn is the source register.

30.3.5.7.2. Operation

Use these instructions to change endianness of data:

`REV` converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.

`REV16` converts two packed 16-bit big-endian data into little-endian data or two packed 16-bit little-endian data into big-endian data.

`REVSH` converts 16-bit signed big-endian data into 32-bit signed little-endian data or 16-bit signed little-endian data into 32-bit signed big-endian data.

30.3.5.7.3. Restrictions

In these instructions, *Rd*, and *Rn* must only specify R0-R7.

30.3.5.7.4. Condition flags

These instructions do not change the flags.

30.3.5.7.5. Examples

```
REV R3, R7              ; Reverse byte order of value in R7 and write it to R3
REV16 R0, R0            ; Reverse byte order of each 16-bit halfword in R0
REVSH R0, R5            ; Reverse signed halfword
```

30.3.5.8. SXT and UXT

Sign extend and Zero extend.

30.3.5.8.1. Syntax

`SXTB` *Rd*, *Rm*

`SXTH` *Rd*, *Rm*

`UXTB` *Rd*, *Rm*

`UXTH` *Rd*, *Rm*

where:

Rd is the destination register.

Rm is the register holding the value to be extended.

30.3.5.8.2. Operation

These instructions extract bits from the resulting value:

- `SXTB` extracts bits[7:0] and sign extends to 32 bits
- `UXTB` extracts bits[7:0] and zero extends to 32 bits
- `SXTH` extracts bits[15:0] and sign extends to 32 bits
- `UXTH` extracts bits[15:0] and zero extends to 32 bits.

30.3.5.8.3. Restrictions

In these instructions, *Rd* and *Rm* must only specify R0-R7.

30.3.5.8.4. Condition flags

These instructions do not affect the flags.

30.3.5.8.5. Examples

```
SXTH R4, R6             ; Obtain the lower halfword of the
                        ; value in R6 and then sign extend to
                        ; 32 bits and write the result to R4.
UXTB R3, R1             ; Extract lowest byte of the value in R10 and zero
                        ; extend it, and write the result to R3
```

30.3.5.9. TST

Test bits.

30.3.5.9.1. Syntax

`TST Rn, Rm`

where:

Rn is the register holding the first operand.

Rm the register to test against.

30.3.5.9.2. Operation

This instruction tests the value in a register against another register. It updates the condition flags based on the

result, but does not write the result to a register.

The `TST` instruction performs a bitwise AND operation on the value in *Rn* and the value in *Rm*. This is the same as the `ANDS` instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the `TST` instruction with a register that has that bit set to 1 and all other bits cleared to 0.

30.3.5.9.3. Restrictions

In these instructions, *Rn* and *Rm* must only specify R0-R7.

30.3.5.9.4. Condition flags

This instruction:

- updates the N and Z flags according to the result
- does not affect the C or V flags.

30.3.5.9.5. Examples

```
TST R0, R1              ; Perform bitwise AND of R0 value and R1 value,
                        ; condition code flags are updated but result is discarded
```

30.3.6. Branch and control instructions

Table 30-19. Branch and Control instructions on page 310shows the branch and control instructions:

Table 30-19. Branch and Control instructions

MNEMONIC	BRIEF DESCIPTION	SEE
`B{CC}`	Branch {conditionally}	B, BL, BX, and BLX in chapter 30.3.6.1 on page 310
`BL`	Branch with Link	B, BL, BX, and BLX in chapter 30.3.6.1 on page 310
`BLX`	Branch indirect with Link	B, BL, BX, and BLX in chapter 30.3.6.1 on page 310
`BX`	Branch indirect	B, BL, BX, and BLX in chapter 30.3.6.1 on page 310

30.3.6.1. B, BL, BX, and BLX

Branch instructions.

30.3.6.1.1. Syntax

`B{cond}` *label*

`BL` *label*

`BX` *Rm*

`BLX` *Rm*

where:

cond is an optional condition code, see Conditional execution in chapter 30.3.3.6 on page 290.

label is a PC-relative expression. See PC-relative expressions in chapter 30.3.3.5 on page 290.

Rm is a register providing the address to branch to.

30.3.6.1.2. Operation

All these instructions cause a branch to the address indicated by label or contained in the register specified by *Rm*. In addition:

- The `BL` and `BLX` instructions write the address of the next instruction to LR, the link register R14.
- The `BX` and `BLX` instructions result in a HardFault exception if bit[0] of Rm is 0.

`BL` and `BLX` instructions also set bit[0] of the LR to 1. This ensures that the value is suitable for use by a subsequent POP {PC} or BX instruction to perform a successful return branch.

Table 30-20. Branch ranges on page 311 shows the ranges for the various branch instructions.

Table 30-20. Branch ranges

INSTRUCTION	BRANCH RANGE
B label	-2KB to + 2KB
Bcond label	-256bytes to +254 bytes
BL label	-16MB to +16MB
BX Rm	Any value in register
BLX Rm	Any value in register

30.3.6.1.3. Restrictions

In these instructions:

- Do not use SP or PC in the `BX` or `BLX` instruction.
- For `BX` and `BLX`, bit[0] of *Rm* must be 1 for correct execution. Bit[0] is used to update the EPSR T-bit and is discarded from the target address.
-

Note

`BCOND` is the only conditional instruction on the Cortex-M0 processor.

Condition flags

These instructions do not change the flags.

Examples

```
B loopA                 ; Branch to loopA
BL funC                 ; Branch with link (Call) to function funC, return address
                        ; stored in LR
BX LR                   ; Return from function call
BLX R0                  ;  Branch with link and exchange (Call) to an address
stored
                        ; in R0
BEQ labelD              ; Conditionally branch to labelD if last flag setting
                        ; instruction set the Z flag, else do not branch.
```

30.3.7. Miscellaneous instructions

Table 30-21. Miscellaneous instructions on page 312 shows the remaining Cortex-M0 instructions:

Table 30-21. Miscellaneous instructions

MNEMONIC	BRIEF DESCIPTION	SEE
BKPT	Breakpoint	BKPT in chapter 30.3.7.1 on page 312
CPSID	Change Processor State, Diosable Interrupts	CPS in chapter 30.3.7.2 on page 313
CPSIE	Change Processor State, Enable Interrupts	CPS in chapter 30.3.7.2 on page 313
DMB	Data Memory Barrier	DMB in chapter 30.3.7.3 on page 314
DSB	Data Synchronization Barrier	DSB in chapter 30.3.7.4 on page 314
ISB	Instruction Synchronization Barrier	ISB in chapter 30.3.7.5 on page 315
MRS	Move from special register to register	MRS in chapter 30.3.7.6 on page 316
MSR	Move form register to special register	MSR in chapter 30.3.7.7 on page 316
NOP	No operation	NOP in chapter 30.3.7.8 on page 317
SEV	Send Event	SEV in chapter 30.3.7.9 on page 318
SVC	Supervisor Call	SVC in chapter 30.3.7.10 on page 318
WFE	Wait for Event	WFE in chapter 30.3.7.11 on page 319
WFI	Wait for interrupt	WFI in chapter 30.3.7.12 on page 320

30.3.7.1. BKPT

Breakpoint.

30.3.7.1.1. Syntax

`BKPT #`*imm*

where:

imm is an integer in the range 0-255.

30.3.7.1.2. Operation

The `BKPT` instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

imm is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The processor might also produce a HardFault or go in to lockup if a debugger is not attached when a `BKPT` instruction is executed. See Lockup in chapter 30.2.4.1 on page 281 for more information.

30.3.7.1.3. Restrictions

There are no restrictions.

30.3.7.1.4. Condition flags

This instruction does not change the flags.

30.3.7.1.5. Examples

```
BKPT #0                 ; Breakpoint with immediate value set to 0x0.
```

30.3.7.2. CPS

Change Processor State.

30.3.7.2.1. Syntax

```
CPSID i
CPSIE i
```

30.3.7.2.2. Operation

CPS changes the PRIMASK special register values. CPSID causes interrupts to be disabled by setting PRIMASK. CPSIE cause interrupts to be enabled by clearing PRIMASK. See Exception mask register in chapter 30.2.1.3.10 on page 268 for more information about these registers.

30.3.7.2.3. Restrictions

There are no restrictions.

30.3.7.2.4. Condition flags

This instruction does not change the condition flags.

30.3.7.2.5. Examples

```
CPSID i                 ; Disable all interrupts except NMI (set PRIMASK)
CPSIE i                 ; Enable interrupts (clear PRIMASK)
```

30.3.7.3. DMB

Data Memory Barrier.

30.3.7.3.1. Syntax

```
DMB
```

30.3.7.3.2. Operation

`DMB` acts as a data memory barrier. It ensures that all explicit memory accesses that appear in program order before the `DMB` instruction are observed before any explicit memory accesses that appear in program order after the `DMB` instruction. `DMB` does not affect the ordering of instructions that do not access memory.

30.3.7.3.3. Restrictions

There are no restrictions.

30.3.7.3.4. Condition flags

This instruction does not change the flags.

30.3.7.3.5. Examples

```
DMB                     ; Data Memory Barrier
```

30.3.7.4. DSB

Data Synchronization Barrier.

30.3.7.4.1. Syntax

```
DSB
```

30.3.7.4.2. Operation

`DSB` acts as a special data synchronization memory barrier. Instructions that come after the `DSB`, in program order, do not execute until the `DSB` instruction completes. The `DSB` instruction completes when all explicit memory accesses before it complete.

30.3.7.4.3. Restrictions

There are no restrictions.

30.3.7.4.4. Condition flags

This instruction does not change the flags.

30.3.7.4.5. Examples

```
DSB                     ; Data Synchronisation Barrier
```

30.3.7.5. ISB

Instruction Synchronization Barrier.

30.3.7.5.1. Syntax

```
ISB
```

30.3.7.5.2. Operation

`ISB` acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the `ISB` are fetched from cache or memory again, after the `ISB` instruction has been completed.

30.3.7.5.3. Restrictions

There are no restrictions.

30.3.7.5.4. Condition flags

This instruction does not change the flags.

30.3.7.5.5. Examples

```
ISB                     ; Instruction Synchronisation Barrier
```

30.3.7.6. MRS

Move the contents of a special register to a general-purpose register.

30.3.7.6.1. Syntax

`MRS` *Rd, spec_reg*

where:

Rd is the general-purpose destination register.

spec_reg is one of the special-purpose registers: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

30.3.7.6.2. Operation

`MRS` stores the contents of a special-purpose register to a general-purpose register. The `MRS` instruction can be combined with the `MSR` instruction to produce read-modify-write sequences, which are suitable for modifying a specific flag in the PSR.

See `MSR` in chapter 30.3.7.7 on page 316.

30.3.7.6.3. Restrictions

In this instruction, *Rd* must not be SP or PC.

30.3.7.6.4. Condition flags

This instruction does not change the flags.

30.3.7.6.5. Examples

```
MRS R0, PRIMASK              ; Read PRIMASK value and write it to R0
```

30.3.7.7. MSR

Move the contents of a general-purpose register into the specified special register.

30.3.7.7.1. Syntax

`MSR` *spec_reg, Rn*

where:

Rn is the general-purpose source register.

spec_reg is the special-purpose destination register: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, or CONTROL.

30.3.7.7.2. Operation

`MSR` updates one of the special registers with the value from the register specified by *Rn*.

See `MRS`on 30.3.7.6 on page 316.

30.3.7.7.3. Restrictions

In this instruction, *Rn* must not be SP and must not be PC.

30.3.7.7.4. Condition flags

This instruction updates the flags explicitly based on the value in *Rn*.

30.3.7.7.5. Examples

```
MSR CONTROL, R1               ; Read R1 value and write it to the CONTROL register
```

30.3.7.8. NOP

No Operation.

30.3.7.8.1. Syntax

`NOP`

30.3.7.8.2. Operation

`NOP` performs no operation and is not guaranteed to be time consuming. The processor might remove it from the pipeline before it reaches the execution stage. Use `NOP` for padding, for example to place the subsequent instructions on a 64-bit boundary.

30.3.7.8.3. Restrictions

There are no restrictions.

30.3.7.8.4. Condition flags

This instruction does not change the flags.

30.3.7.8.5. Examples

```
NOP                     ; No operation
```

30.3.7.9. SEV

Send Event.

30.3.7.9.1. Syntax

`SEV`

30.3.7.9.2. Operation

`SEV` causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register, see Power management in chapter 30.2.5 on page 282.

See also `WFE` in in chapter 30.3.7.11 on page 319.

30.3.7.9.3. Restrictions

There are no restrictions.

30.3.7.9.4. Condition flags

This instruction does not change the flags.

30.3.7.9.5. Examples

```
SEV                     ; Send Event
```

30.3.7.10. SVC

Supervisor Call.

30.3.7.10.1. Syntax

`SVC #`*imm*

where:

imm is an integer in the range 0-255.

30.3.7.10.2. Operation

The `SVC` instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

30.3.7.10.3. Restrictions

There are no restrictions.

30.3.7.10.4. Condition flags

This instruction does not change the flags.

30.3.7.10.5. Examples

```
SVC #0x32          ; Supervisor Call (SVC handler can extract the immediate value
                   ; by locating it via the stacked PC)
```

30.3.7.11. WFE

Wait For Event.

30.3.7.11.1. Syntax

`WFE`

30.3.7.11.2. Operation

If the event register is 0, `WFE` suspends execution until one of the following events occurs:

- an exception, unless masked by the exception mask registers or the current priority level
- an exception enters the Pending state, if SEVONPEND in the System Control Register is set
- a Debug Entry request, if debug is enabled
- an event signaled by a peripheral or another processor in a multiprocessor system using the `SEV` instruction.

If the event register is 1, `WFE` clears it to 0 and completes immediately.

For more information see see Power management in chapter 30.2.5 on page 282.

Note

`WFE` is intended for power saving only. When writing software assume that `WFE` might behave as `NOP`.

30.3.7.11.3. Restrictions

There are no restrictions.

30.3.7.11.4. Condition flags

This instruction does not change the flags.

30.3.7.11.5. Examples

```
WFE                     ; Wait for event
```

30.3.7.12. WFI

Wait for Interrupt.

30.3.7.12.1. Syntax

```
WFI
```

30.3.7.12.2. Operation

`WFI` suspends execution until one of the following events occurs:

- an exception
- an interrupt becomes pending which would preempt if PRIMASK was clear
- a Debug Entry request, regardless of whether debug is enabled.
-

Note

`WFI` is intended for power saving only. When writing software assume that `WFI` might behave as a NOP operation.

30.3.7.12.3. Restrictions

There are no restrictions.

30.3.7.12.4. Condition flags

This instruction does not change the flags.

30.3.7.12.5. Examples

```
WFI                     ; Wait for interrupt
```

30.4. Cortex-M0 Peripherals

The following sections are the reference material for the ARM Cortex-M0 core peripherals descriptions in a User Guide:

- About the Cortex-M0 peripherals in chapter 30.4.1 on page 321
- Nested Vectored Interrupt Controller in chapter 30.4.2 on page 321
- System Control Block in chapter 30.4.3 on page 327
- System timer, SysTick in chapter 30.4.4 on page 334

30.4.1. About the Cortex-M0 peripherals

The address map of the Private peripheral bus (PPB) is:

Table 30-22. Core peripheral register regions

ADDRESS	CORE PERIPHERAL	DESCRIPTION
0xE000 E008 – 0xE000 E00F	System Control Block	Table 30-8. CONTROL register bit assignmentson page 268
0xE000 E010 – 0xE000 E01F	System Timer	Table 30-24. CMSIS access NVIC functions on page 322
0xE000 E100 – 0xE000 E4EF	Nested Vectored Interrupt Controller	Table 30-23. NVIC register summary on page 322
0xE000 ED00 – 0xE000 ED3F	System Control Block	Table 30-8. CONTROL register bit assignmentson page 268
0xE000 EF00 – 0xE000 EF03	Nested Vectored Interrupt Controller	Table 30-23. NVIC register summary on page 322

In register descriptions, the register type is described as follows:

RW Read and write.

RO Read-only.

WO Write-only.

30.4.2. Nested Vectored Interrupt Controller

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- 1 to 32 interrupts.
- A programmable priority level of 0-192 in steps of 64 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Interrupt tail-chaining.
- An external Non-maskable interrupt (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the

NVIC registers is:

Table 30-23. NVIC register summary

ADDRESS	NAME	TYPE	RESET VALUE	DESCRIPTION
0xE000 E100	ISER	RW	0x0000 0000	Interrupt Set-enable Register in chapter 30.4.2.2 on page 322
0xE000 E180	ICER	RW	0x0000 0000	Interrupt Clear-enable Register in chapter 30.4.2.3 on page 323
0xE000 E200	ISPR	RW	0x0000 0000	Interrupt Set-pending Register in chapter 30.4.2.4 on page 323
0xE000 E280	ICPR	RW	0x0000 0000	Interrupt Clear-pending Register in chapter 30.4.2.5 on page 324
0xE000 E400 – 0xE000 E41C	IPR0-7	RW	0x0000 0000	Interrupt Priority Registers in chapter 30.4.2.6 on page 324

30.4.2.1. Accessing the Cortex-M0 NVIC registers using CMSIS

CMSIS functions enable software portability between different Cortex-M profile processors.

To access the NVIC registers when using CMSIS, use the following functions:

Table 30-24. CMSIS access NVIC functions

CMSIS FUNCTION	DESCRIPTION
`void NVIC_EnableIRQ(IRQn_Type IRQn)*`	Enables an interrupt or exception
`void NVIC_DisableIRQ(IRQn_Type IRQn)*`	Disables an interrupt or exception
`void NVIC_SetPendingIRQ(IRQn_Type IRQn)*`	Set the pending status of interrupt or exception to 1
`void NVIC_ClearPendingIRQ(IRQn_Type IRQn)*`	Clears the pending status of interrupt or exception to 0
`uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)*`	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to1.
`void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)*`	Sets the priority of an interrupt or exception with configurable priority level to 1
`uint32_t NVIC_GetPriority(IRQn_Type IRQn)*`	Reads the priority of an interrupt or exception with configurable priority level. This function returns the current priority level

* The input parameter IRQn is the IRQ number see Table 30-7. PRIMASK register bit assignments on page 268 for more information

30.4.2.2. Interrupt Set-enable Register

The ISER enables interrupts, and shows which interrupts are enabled. See the register summary in Table 30-23. NVIC register summary on page 322 for the register attributes.

The bit assignments are:

Figure 30-16. ISER

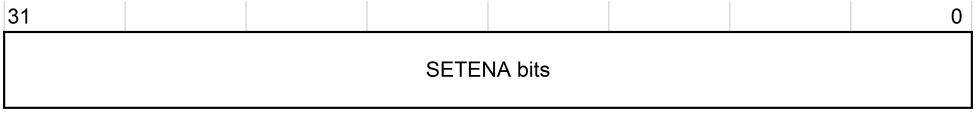


Table 30-25. ISER bit assignments

BIT	NAME	FUNCTION
31:0	**SETENA**	Interrupt set-enable bits. Write: 1: enable interrupt 0: no effect Read: 1: interrupt enabled 0: interrupt disabled

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority.

If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

30.4.2.3. Interrupt Clear-enable Register

The ICER disables interrupts, and show which interrupts are enabled. See the register summary in Table 30-23. NVIC register summary on page 322 for the register attributes.

The bit assignments are:

Figure 30-17. ICER

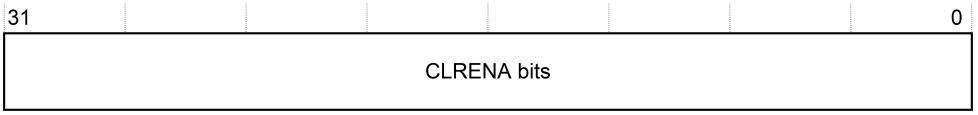


Table 30-26. ICER bit assignments

BIT	NAME	FUNCTION
31:0	**CLEARENA**	Interrupt clear-enable bits. Write: 1: disable interrupt 0: no effect Read: 1: interrupt enabled 0: interrupt disabled

30.4.2.4. Interrupt Set-pending Register

The ISPR forces interrupts into the pending state, and shows which interrupts are pending. See the register summary in Table 30-23. NVIC register summary on page 322 for the register attributes.

The bit assignments are:

Figure 30-18. ISPR

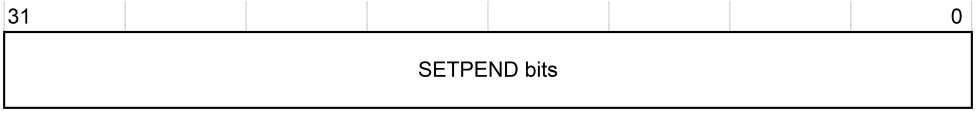


Table 30-27. ISPR bit assignments

BIT	NAME	FUNCTION
31:0	**SETPEND**	Interrupt set-pending bits. Write: 1: changes interrupt state to pending 0: no effect Read: 1: changes interrupt state to pending 0: interrupt not pending

Note
Writing 1 to the ISPR bit corresponding to:

- an interrupt that is pending has no effect
- a disabled interrupt sets the state of that interrupt to pending.

30.4.2.5. Interrupt Clear-pending Register

The ICPR removes the pending state from interrupts, and shows which interrupts are pending. See the register summary in Table 30-23. NVIC register summary on page 322 for the register attributes.
The bit assignments are:

Figure 30-19. ICPR

Table 30-28. ICPR bit assignments

BIT	NAME	FUNCTION
31:0	**CLRPEND**	Interrupt clear-pending bits. Write: 1: removes pending state an interrupt 0: no effect Read: 1: interrupt is pending 0: interrupt not pending

Note

Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt

30.4.2.6. Interrupt Priority Registers

The IPR0-IPR7 registers provide an 8-bit priority field for each interrupt. These registers are only word-accessible. See the register summary in Table 30-23. NVIC register summary on page 322 for their attributes. Each register holds four priority fields as shown:

Figure 30-20. IPR

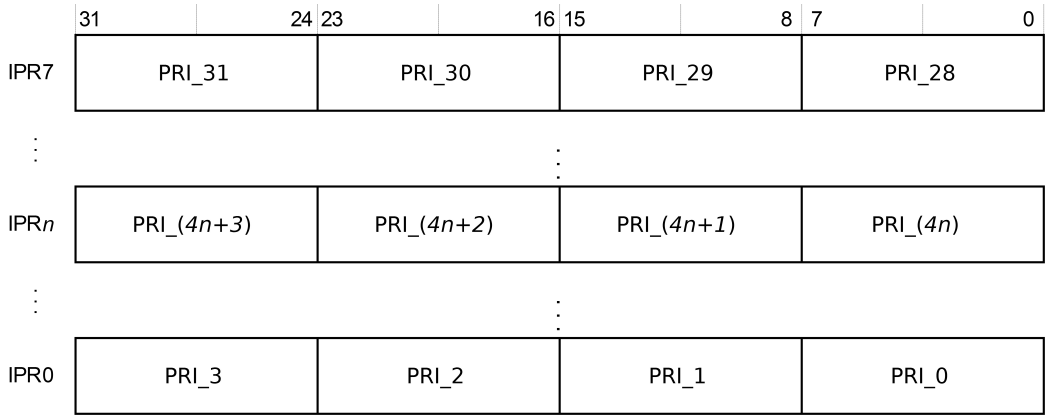


Table 30-29. IPR bit assignments

BIT	NAME	FUNCTION
31:24	**Priority, byte offset 3**	Each priority field holds a priority value, 0-192. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:6] of each field, bits [5:0] read as zero and ignore writes. This means writing 255 to a priority register saves value 192 to the register
23:16	**Priority, byte offset 2**	
15:8	**Priority, byte offset 1**	
7:0	**Priority, byte offset 3**	

See Accessing the Cortex-M0 NVIC registers using CMSIS in chapter 30.4.2.1 on page 322 for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt M as follows:

- the corresponding IPR number, N, is given by N = N DIV 4
- the byte offset of the required Priority field in this register is M MOD 4, where:
  - byte offset 0 refers to register bits[7:0]
  - byte offset 1 refers to register bits[15:8]
  - byte offset 2 refers to register bits[23:16]
  - byte offset 3 refers to register bits[31:24]

30.4.2.7. Level-sensitive and pulse interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC

detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see Hardware and software control of interrupts in chapter 30.4.2.7.1 on page 326. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

30.4.2.7.1. Hardware and software control of interrupts

The Cortex-M0 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is active and the corresponding interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see Interrupt Set-pending Register in chapter 30.4.2.4 on page 323

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit. For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive. For a pulse interrupt, state of the interrupt changes to:
  - inactive, if the state was pending
  - active, if the state was active and pending.

30.4.2.8. NVIC usage hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

30.4.2.8.1. NVIC programming hints

Software uses the CPSIE i and CPSID i instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 30-30. CMSIS access NVIC functions

CMSIS FUNCTION	DESCRIPTION
`void NVIC_EnableIRQ(IRQn_Type IRQn`	Enable IRQn
`void NVIC_DisableIRQ(IRQn_Type IRQn)`	Disable IRQn
`uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)`	Return true (1) if IRQn is pending
`void NVIC_SetPendingIRQ(IRQn_Type IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ(IRQn_Type IRQn)`	Clear IRQn pending state
`void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`	Set priority for IRQn
`uint32_t NVIC_GetPriority(IRQn_Type IRQn)`	Read priority of IRQn
`void NVIC_SystemReset (void)`	Reset the system

The input parameter IRQn is the IRQ number, see Table 30-7. PRIMASK register bit assignments on page 268 more information. For more information about these functions, see the CMSIS documentation.

30.4.3. System Control Block

The System Control Block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The SCB registers are:

Table 30-31. Summary of the SCB register

ADDRESS	NAME	TYPE	RESET VALUE	DESCRIPTION
0xE000 ED00	CPUID	RO	0x410C C200	CPUID Register in chapter 30.4.3.2 on page 327
0xE000 ED04	ICSR	RW*	0x0000 0000	Interrupt Control and State Register in chapter 30.4.3.3 on page 328
0xE00 ED0C	AIRCR	RW*	0xFA05 0000	Application Interrupt and Reset Control Register in chapter 30.4.3.4 on page 330
0xE000 ED10	SCR	RW	0x0000 0000	System Control Register in chapter 30.4.3.5 on page 331
0xE000 ED14	CCR	RW	0x0000 0204	Configuration and Control register in chapter 30.4.3.6 on page 332
0xE000 ED1C	SHPR2	RW	0x0000 0000	System Handler Priority register in chapter 30.4.3.7 on page 332
0xE000 ED20	SHPR3	RW	0x0000 0000	System Handler Priority register in chapter 30.4.3.7 on page 332

* See the register description for more information

30.4.3.1. The CMSIS mapping of the Cortex-M0 SCB registers

To improve software efficiency, the CMSIS simplifies the SCB register presentation. In the CMSIS, the array SHP[1] corresponds to the registers SHPR2-SHPR3

30.4.3.2. CPUID Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in Table 30-22. Core peripheral register regions on page 321 for its attributes. The bit

assignments are:

Figure 30-21. CPUID

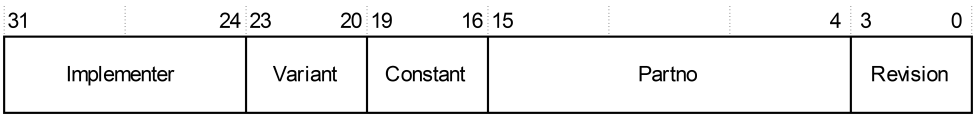


Table 30-32. CPUID register bit assignments

BIT	NAME	FUNCTION
31:24	**Implementer**	Implementer code: 0x41 = ARM
23:20	**Variant**	Variant number, the r value in the rnpn product revision identifier: 0x0 = Revision 0
19:16	**Constant**	Constant that defines the architecture of the processor:, reads as 0xC = ARMv6-M architecture
15:4	**Partno**	Part number of the processor: 0xC20 = Cortex-M0
3:0	**Revision**	Revision number, the p value in the rnpn product revision identifier: 0x0 = Patch 0

30.4.3.3. Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the Non-Maskable Interrupt (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- •indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

See the register summary in Table 30-14. CMSIS intrinsic functions to access special registers on page 287 for the ICSR attributes. The bit assignments are:

Figure 30-22. ICSR

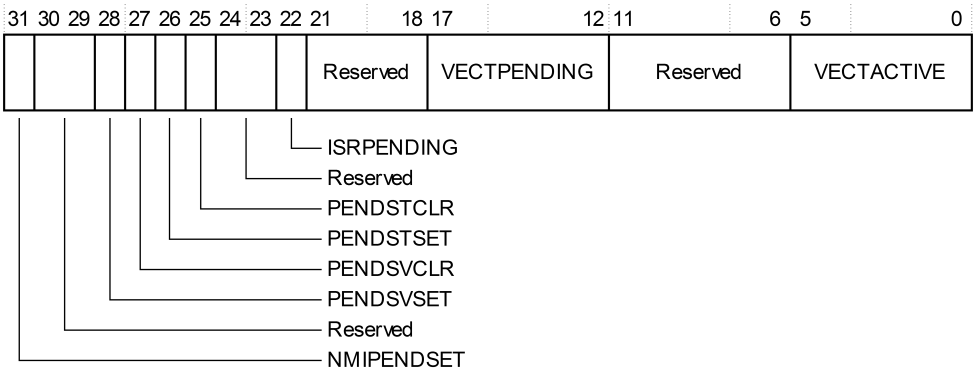


Table 30-33. ICSR register bit assignments

BIT	NAME	TYPE	FUNCTION
31	**NMIPENDSET**	RW	NMI set-pending bit Write: 1: changes NMI exception state to pending 0: no effect Read: 1: NMI exception is pending 0: NMI exception is not pending Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it detects a write of 1 to this bit. Entering the handler then clears this bit to 0. This means a read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
30:29	**Reserved**	RW	Reserved
28	**PENDSVSET**	RW	PendSV set-pending bit Write: 1: changes PendSV exception state to pending 0: no effect Read: 1: PendSV exception is pending 0: PendSV exception is not pending Writing 1 to this bit is the only way to set the PendSV state to pending
27	**PENDSVCLR**	WO	PendSV clear-pending bit Write: 1: removes the pending state from the PendSV exception 0: no effect
26	**PENDSTSET**	RW	SysTick exception set-pending bit. Write: 1: changes SysTick exception state to pending 0: no effect Read: 1: SysTick exception is pending 0: SysTick exception is not pending
25	**PENDSTCLR**	WO	SysTick exception clear-pending bit. Write: 1: removes the pending state from the SysTick exception This bit is WO. On a register read it's value is unknown
24:23	**Reserved**	RW	Reserved
22	**ISRPENDING**	RO	Interrupt pending flag, excluding NMI and Faults: 1: Interrupt pending 0: Interrupt not pending

BIT	NAME	TYPE	FUNCTION
21:18	**Reserved**	RW	Reserved
17:12	**VECTPENDING**	RO	Indicates the exception number of the highest priority pending enabled exception: Nonzero: the exception number of the highest priority pending enabled exception 0: no pending exceptions
11:6	**Reserved**	RW	Reserved
5:0	**VECTACTIVE***	RO	Contains the active exception number: Nonzero: The exception number* of the currently active exception 0: Thread mode **Note:** Subtract 16 from this value to obtain the CMSIS IRQ number that identifies the corresponding bit in the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-pending, and Priority Register, see Table 30-5. IPSR bit assignments on page 266

* This is the same value as IPSR bits[5:0], see Table 30-5. IPSR bit assignments on page 266

When you write to the ICSR, the effect is Unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

30.4.3.4. Application Interrupt and Reset Control Register

The AIRCR provides endian status for data accesses and reset control of the system. See the register summary in Table 30-8. CONTROL register bit assignments on page 268 and Table 30-13. CMSIS intrinsic functions to generate some Cortex-M0 instructions on page 286 for its attributes.

To write to this register, you must write 0x05FA to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

Figure 30-23. AIRCR

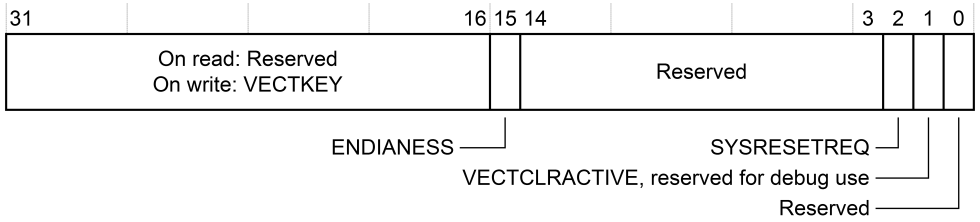


Table 30-34. AIRCR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:16	**VECTKEY**	RW	Register key Read: unknown Write: write 0x05FA to VECTKEY, otherwise the write is ignored
15	**ENDIANESS**	RO	Data endianess implemented 0x0: Little Endian

BIT	NAME	TYPE	FUNCTION
14:3	**Reserved**	RW	Reserved
2	**SYSRESETREQ**	WO	System reset request: Read: This bit reads as 0. Write: 1: requests a system level reset. 0: no effect
1	**VECTCLRACTIVE**	WO	Reserved for debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
0	**Reserved**	RW	Reserved

30.4.3.5. System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary inTable 30-10. Properties of the different exception types on page 276 for its attributes. The bit assignments are:

Figure 30-24. SCR

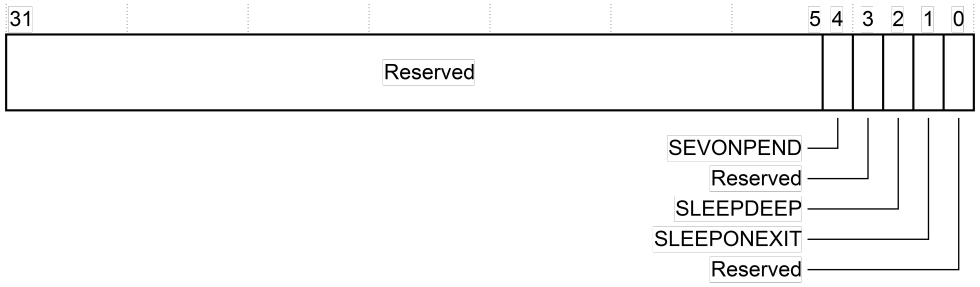


Table 30-35. SCR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:5	**Reserved**	R	Reserved
4	**SEVONPEND**	RW	Send Event on Pending bit: 1: enabled events and all interrupts, including disabled interrupts, can wakeup the processor 0: only enabled interrupts or events can wakeup the processor ,disabled interrupts are excluded When an event or interrupt enters pending state, the event signal wakes up the processor from `WFE`. If the processor is not waiting for an event ,the event is registered and affects the next `WFE`. The processor also wakes up on execution of an `SEV` instruction or external event
3	**Reserved**	R	Reserved
2	**SLEEPDEEP**	RW	Controls whether the processor uses sleep or deep sleep as its low power mode: 1: deep sleep 0: sleep

BIT	NAME	TYPE	FUNCTION
1	**SLEEPONEXIT**	RW	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 1: enter sleep, or deep sleep, on return from an ISR to Thread mode. 0: do not sleep when returning to Thread mode. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application
0	**Reserved**	R	Reserved

30.4.3.6. Configuration and Control Register

The CCR is a read-only register and indicates some aspects of the behavior of the Cortex-M0 processor. See the register summary in Table 30-9. Memory Access Behavior in page 272 for the CCR attributes.
The bit assignments are:

Figure 30-25. CCR

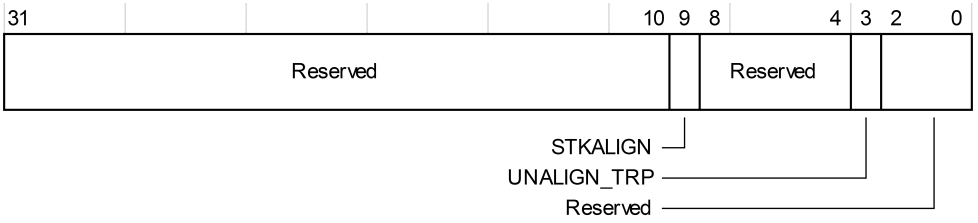


Table 30-36. CCR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:10	**Reserved**	R	Reserved
9	**STKALIGN**	RW	Always reads as one, indicates 8-byte stack alignment on exception entry. On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment.
8:4	**Reserved**	R	Reserved
3	**UNALIGN_TRP**	RW	Always reads as one, indicates that all unaligned accesses generate a HardFault.
0	**Reserved**	R	Reserved

30.4.3.7. System Handler Priority Registers

The SHPR2-SHPR3 registers set the priority level, 0 to 192, of the exception handlers that have configurable priority.

SHPR2-SHPR3 are word accessible. See the register summary in Table 30-8. CONTROL register bit assignments on page 268 for their attributes.

To access to the system exception priority level using CMSIS, use the following CMSIS functions:

- `uint32_t NVIC_GetPriority(IRQn_Type IRQn)`
- `void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)`

The input parameter IRQn is the IRQ number, see Table 30-7. PRIMASK register bit assignments on page 268 for more information.

The system fault handlers, and the priority field and register for each handler are:

Table 30-37. System fault handler priority fields

HANDLER	FIELD	REGISTER DESCRIPTION
SVCall	**PRI_11**	System handler Priority Register 2 on page
PendSV	**PRI_14**	System Handler Priority register 3 on page
SysTick	**PRI_15**	System Handler Priority register 3 on page

Each PRI_N field is 8 bits wide, but the processor implements only bits[7:6] of each field, and bits[5:0] read as zero and ignore writes.

30.4.3.7.1. System Handler Priority Register 2

The bit assignments are:

Figure 30-26. SHPR2

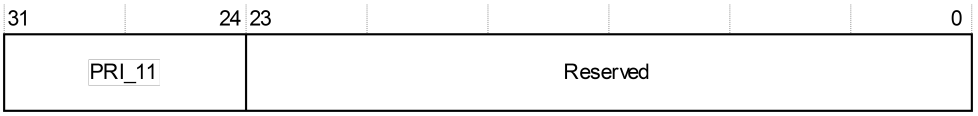


Table 30-38. SHPR2 register bit assignments

BIT	NAME	TYPE	FUNCTION
31:24	**PRI_11**	RW	Priority of system Handler 11, SVCall
23:0	**Reserved**	R	Reserved

30.4.3.7.2. System Handler Priority Register 3

The bit assignments are:

Figure 30-27. SHPR3

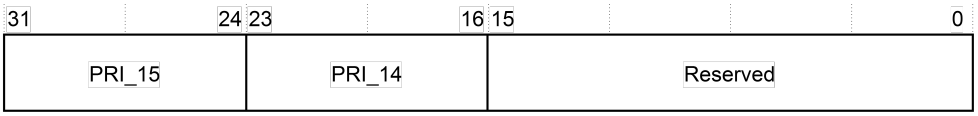


Table 30-39. SHPR3 register bit assignments

BIT	NAME	TYPE	FUNCTION
31:24	**PRI_15**	RW	Priority of system Handler 15, SysTick exception
23:16	**PRI_14**	RW	Priority of system Handler 14, PendSV
15:0	**Reserved**	R	Reserved

30.4.3.8. SCB usage hints and tips

Ensure software uses aligned 32-bit word size transactions to access all the SCB registers.

30.4.4. System timer, SysTick

When enabled, the timer counts down from the reload value to zero, reloads (wraps to) the value in the SYST_RVR on the next clock cycle, then decrements on subsequent clock cycles. Writing a value of zero to the SYST_RVR disables the counter on the next wrap. When the counter transitions to zero, the COUNTFLAG status bit is set to 1. Reading SYST_CSR clears the COUNTFLAG bit to 0.

Writing to the SYST_CVR clears the register and the COUNTFLAG status bit to 0. The write does not trigger the SysTick exception logic. Reading the register returns its value mat the time it is accessed.

Note

When the processor is halted for debugging the counter does not decrement. The system timer registers are:

Table 30-40. System timer register summary

ADDRESS	NAME	TYPE	RESET VALUE	DESCRIPTION
0xE000 E010	SYST_CSR	RW	0x0000 0000	Systick Control and Status Register in chapter 30.4.4.1 on page 334
0xE000 E014	SYST_RVR	RW	Unknown	Systick Reload Value Register in chapter 30.4.4.2 on page 335
0xE000 E018	SYST_CVR	RW	Unknown	Systick Current Register in chapter 30.4.4.3 on page 335
0xE000 E01C	SYST_CALIB	RO	0x0000 0000	Systick Calibration value Register in chapter 30.4.4.4 on page 336

30.4.4.1. SysTick Control and Status Register

The SYST_CSR enables the SysTick features. See the register summary in Table 30-6. EPSR bit assignments on page 267 for its attributes. The bit assignments are:

Figure 30-28. SYST_CSR

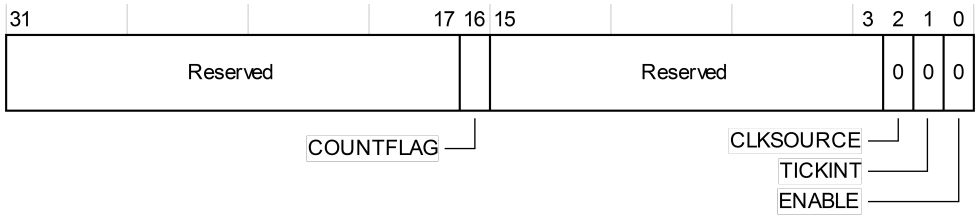


Table 30-41. SYST_CSR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:17	**Reserved**	R	Reserved
16	**COUNTFLAG**	RW	Returns 1 if timer counted to 0 since the last read of this register

BIT	NAME	TYPE	FUNCTION
15:3	**Reserved**	R	Reserved
2	**CLKSOURCE**	RW	Selects the SysTick timer clock source 1: HCLK 0: FCLK / 3
1	**TICKINT**	RW	Enables SysTick exception request 1: counting down to zero to assert the SysTick exception request 0: counting down to zero does not assert the SysTick exception request
0	**ENABLE**	RW	Enables the counter 1: counter enabled 0: counter disabled

30.4.4.2. SysTick Reload Value Register

The SYST_RVR specifies the start value to load into the SYST_CVR. See the register summary in Table 30-4. APSR bit assignments on page 266 for its attributes. The bit assignments are:

Figure 30-29. SYST_RVR

Table 30-42. SYST_RVR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:24	**Reserved**	R	Reserved
23:0	**RELOAD**	RW	Value to load into the SYST_CVR when the counter is enabled and when it reaches 0, see Calculating the RELOAD value.

30.4.4.2.1. Calculating the RELOAD value

The RELOAD value can be any value in the range 0x000 00001 – 0x00FF FFFF. You can program a value of 0, but this has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

30.4.4.3. SysTick Current Value Register

The SYST_CVR contains the current value of the SysTick counter. See the register summary in Table 30-3. Core register set summary on page 266 for its attributes. The bit assignments are:

Figure 30-30. SYST_CVR

Table 30-43. SYST_CVR register bit assignments

BIT	NAME	TYPE	FUNCTION
31:24	**Reserved**	R	Reserved
23:0	**CURRENT**	RW	Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SYST_CSR.COUNTFLAG bit to 0.

30.4.4.4. SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties. See the register summary in Table 30-1. Summary of processor mode and stack use options on page 263 for its attributes. The bit assignments are:

Figure 30-31. SYST_CALIB

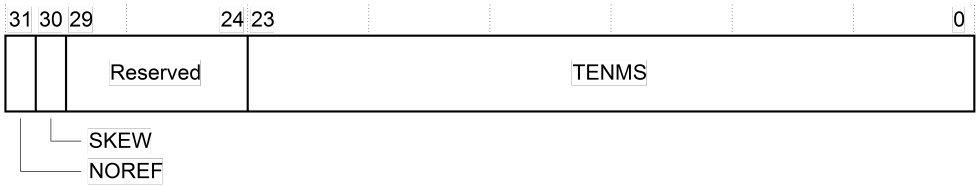


Table 30-44. SYST_CALIB register bit assignments

BIT	NAME	TYPE	FUNCTION
31	**NOREF**	R	Reads as one. Indicates no separate reference clock is provided
30	**SKEW**	R	Reads as one. Calibration value for the 10ms inexact timing is not known because TENMS is not known. This can affect the suitability of SysTick as a software real time clock.
29:24	**Reserved**	R	Reserved
23:0	**TENMS**	RW	Reads as zero. Indicates calibration value is not known

If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

30.4.4.5. SysTick usage hints and tips

The interrupt controller clock updates the SysTick counter.

Ensure software uses word accesses to access the SysTick registers.

If the SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

31. CONTACT INFORMATION

For the latest specifications, additional product information, worldwide sales and distribution locations:

Web: www.qorvo.com

Tel: 1-844-890-8163

Email: customer.support@qorvo.com

32. IMPORTANT NOTICE

The information contained in this Users Guide and any associated documents ("Users Guide Information") is believed to be reliable; however, Qorvo makes no warranties regarding the Users Guide Information and assumes no responsibility or liability whatsoever for the use of said information. All Users Guide Information is subject to change without notice. Customers should obtain and verify the latest relevant Users Guide Information before placing orders for Qorvo® products. Users Guide Information or the use thereof does not grant, explicitly, implicitly or otherwise any rights or licenses to any third party with respect to patents or any other intellectual property whether with regard to such Users Guide Information itself or anything described by such information.

USERS GUIDE INFORMATION DOES NOT CONSTITUTE A WARRANTY WITH RESPECT TO THE PRODUCTS DESCRIBED HEREIN, AND QORVO HEREBY DISCLAIMS ANY AND ALL WARRANTIES WITH RESPECT TO SUCH PRODUCTS WHETHER EXPRESS OR IMPLIED BY LAW, COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE OF TRADE OR OTHERWISE, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Without limiting the generality of the foregoing, Qorvo® products are not warranted or authorized for use as critical components in medical, life-saving, or life-sustaining applications, or other applications where a failure would reasonably be expected to cause severe personal injury or death. Applications described in the Users Guide Information are for illustrative purposes only. Customers are responsible for validating that a particular product described in the Users Guide Information is suitable for use in a particular application.