Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CY8C4Axx

PSoC® Analog Coprocessor Architecture Technical Reference Manual (TRM)

Document No. 002-10404 Rev. *C

January 9, 2019

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Contents Overview

Contents

Section A: Overview

This section encompasses the following chapters:

Document Revision History

Revision	Issue Date	Origin of Change	Description of Change
**	December 18, 2015	RJVB	Initial version of the PSoC Analog Coprocessor TRM
*A	January 19, 2017	DIMA	Provided trigger mux details in the CTB chapter for hardware controlled switches. Added UAB synchronization details. Updated SARADC chapter to include information about UAB filter on one channel of ADC and updated the figure “SARSEQ Block Diagram” to include UAB, Updated links to PSoC Analog Coprocessor product webpage and updated the number and URL of the Getting Started application note. Also, provided link to the AFE application note in the Getting Started chapter. Updated the Introduction chapter with information on interfacing with multiple sensors, number of PRB channels, and UAB filter channel of ADC.
*B	May 31, 2017	SHEA	Updated logo and copyright information
*C	January 9, 2019	RJVB	Updated Figure 18-1 caption

1. Introduction

PSoC® Analog Coprocessor is a programmable embedded system controller with an ARM® Cortex®-M0+ CPU. This device offers a low-cost integrated solution to interface multiple analog sensors along with programmability using Cortex M0+ processor core. For power efficiency, the PSoC device can form an analog coprocessor unit, offloading the tasks of analog signal processing and data conversion from the main processor in the system. The configurable analog resources in PSoC Analog Coprocessor enable the user to implement configurable analog front end (AFE) required for analog sensor applications.

PSoC Analog Coprocessor have these characteristics:

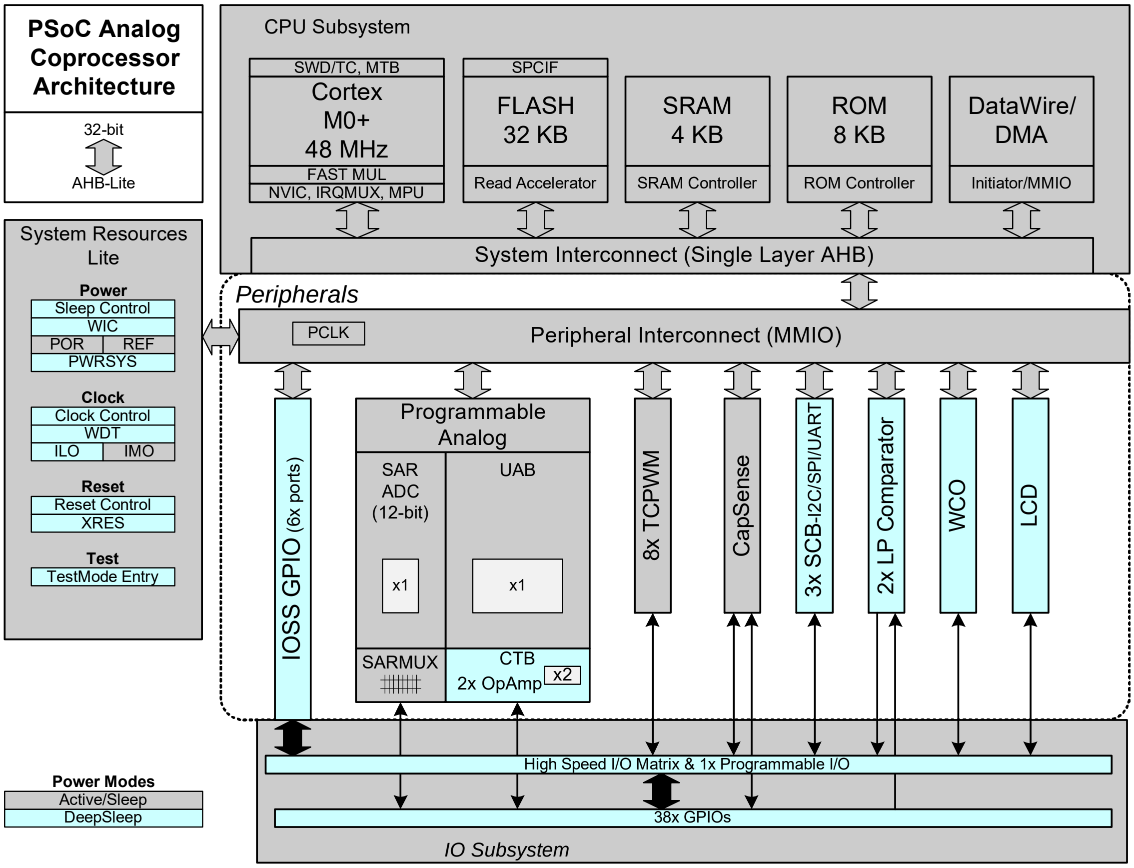
- High-performance, 32-bit single-cycle Cortex-M0+ CPU core
- High-performance analog system
- Self and Mutual Capacitive touch sensing (CapSense®)
- Configurable Timer/Counter/PWM block
- Configurable analog blocks for analog signal conditioning
- Configurable communication block with I^2C, SPI, and UART operating modes
- Low-power operating modes – Sleep and Deep-Sleep

This document describes each functional block of the PSoC Analog Coprocessor device in detail. This information will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of the PSoC Analog Coprocessor architecture.

Figure 1-1. PSoC Analog Coprocessor Block Diagram



1.2 Features

The PSoC Analog Coprocessor has these major components:

- 32-bit Cortex-M0+ CPU with single-cycle multiply, delivering up to 0.9 DMIPS/MHz
- Up to 32 KB flash and 4 KB SRAM
- Direct memory access (DMA)
- Eight center-aligned pulse-width modulator (PWM) with complementary, dead-band programmable outputs
- One watchdog timer and three general-purpose timers with interrupt capability
- Twelve-bit SAR ADC (with a sampling rate of 1 Msps) with hardware sequencing for multiple channels
- Programmable analog blocks for signal conditioning - Two Continuous Time Blocks (CTB) with four opamps and one Universal Analog Block (UAB) with two opamps
- Programmable reference block (PRB), which generates four different reference voltages for use with the analog blocks
- Two low-power comparators
- Three serial communication blocks (SCB) that can work as SPI, UART, I^2C, and local interconnect network (LIN) slave serial communication channels
- A Smart I/O block, which provides the ability to perform Boolean functions in the I/O signal path
- CapSense
- Segment LCD direct drive
- Low-power operating modes: Sleep and Deep-Sleep
- Programming and debugging system through serial wire debug (SWD)
- Fully supported by PSoC Creator™ IDE tool

1.3 CPU System

1.3.1 Processor

The heart of the PSoC Analog Coprocessor is a 32-bit Cortex-M0+ CPU core running up to 48 MHz. It is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set. This instruction set enables fully compatible binary upward migration of the code to higher performance processors such as Cortex M3 and M4.

The CPU has a hardware multiplier that provides a 32-bit result in one cycle.

1.3.2 Interrupt Controller

The CPU subsystem includes a nested vectored interrupt controller (NVIC) with 25 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode.

1.3.3 Direct Memory Access

The DMA engine is capable of independent data transfers anywhere within the memory map (peripheral-to-peripheral and peripheral-to/from-memory) with a programmable descriptor chain.

1.4 Memory

The memory subsystem consists of flash and SRAM. A supervisory ROM, containing boot and configuration routines, is also present.

1.4.1 Flash

The PSoC Analog Coprocessor has a flash module, with a flash accelerator tightly coupled to the CPU, to improve average access times from the flash block. The flash accelerator delivers 85 percent of single-cycle SRAM access performance on an average.

1.4.2 SRAM

It provides SRAM, which is retained in all power modes of the device.

1.5 System-Wide Resources

1.5.1 Clocking System

The clocking system consists of the internal main oscillator (IMO) and internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock and watch crystal oscillator (WCO).

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the device. The default IMO frequency is 24 MHz and can be adjusted between 24 MHz and 48 MHz in steps of 4 MHz. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a low-power, less accurate oscillator and is used as a source for LFCLK, to generate clocks for peripheral operation in Deep-Sleep mode. Its clock frequency is 40 kHz with ±60 percent accuracy.

An external clock source ranging from 1 MHz to 48 MHz can be used to generate the clock derivatives for the functional blocks instead of the IMO.

The WCO is a 32-kHz watch crystal oscillator. It is used to dynamically trim the IMO to an accuracy of ±1 percent to enable precision timing applications.

1.5.2 Power System

The device operates with a single external supply in the range 1.71 V to 5.5 V. It provides multiple power supply domains – V_{DDD} to power digital section, and V_{DDA} for noise isolation of analog section. V_{DDD} and V_{DDA} should be shorted externally.

The device has two low-power modes – Sleep and Deep-Sleep – in addition to the default Active mode. In Active mode, the CPU runs with all the logic powered. In Sleep mode, the CPU is powered off with all other peripherals functional. In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention; the main system clock is OFF while the low-frequency clock is ON and the low-frequency peripherals are in operation.

Multiple internal regulators are available in the system to support power supply schemes in different power modes.

1.5.3 GPIO

Every GPIO has the following characteristics:

- Eight drive strength modes
- Individual control of input and output disables
- Hold mode for latching previous state
- Selectable slew rates
- Interrupt generation – edge triggered

In addition, the device has a Smart I/O block that provides the ability to perform Boolean functions on the port I/Os. The Smart I/O block is available in all device power modes, including low-power modes.

The pins are organized in a port that is 8-bit wide. A high-speed I/O matrix is used to multiplex between various signals that may connect to an I/O pin. Pin locations for fixed-function peripherals are also fixed.

1.5.4 Watchdog Timers

The PSoC device has one 16-bit watchdog timer, which is capable of automatically resetting the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality.

In addition to this, two 16-bit and one 32-bit up-counting timers are available for general-purpose use.

1.6 Fixed-Function Digital

1.6.1 Timer/Counter/PWM Block

The Timer/Counter/PWM block consists of eight 16-bit counter with user-programmable period length. The TCPWM block has a capture register, period register, and compare register. The block supports complementary, dead-band programmable outputs. It also has a kill input to force outputs to a predetermined state. Other features of the block include center-aligned PWM, clock prescaling, pseudo random PWM, and quadrature decoding.

1.6.2 Serial Communication Blocks

The device has three SCBs. Each SCB can implement a serial communication interface as I^2C, UART, local interconnect network (LIN) slave, or SPI.

The features of each SCB include:

- Standard I^2C multi-master and slave function
- Standard SPI master and slave function with Motorola, Texas Instruments, and National (MicroWire) mode
- Standard UART transmitter and receiver function with SmartCard reader (ISO7816), IrDA protocol, and LIN
- Standard LIN slave with LIN v1.3 and LIN v2.1/2.2 specification compliance
- EZ function mode support with 32-byte buffer

1.7 Analog System

1.7.1 SAR ADC

The PSoC Analog Coprocessor has a configurable 12-bit 1-Msps SAR ADC. The ADC provides three internal voltage references (V_{DDA}, $V_{DDA}/2$, and V_{REF}) and an external reference through a GPIO pin. V_{REF} come from the PRB. The SAR hardware sequencer is available, which scans multiple channels without CPU intervention. An optional configurable analog filter is available on one channel of ADC.

1.7.2 Continuous Time Block (CTB)

CTB provides operational amplifiers for use in continuous-time signal chains. Each CTB contains two opamps, a switch matrix for input/output routing, and the resistor array for each opamp gain setting. The device has two such CTB blocks. These blocks work in Active, Sleep, and Deep-Sleep modes of the device.

CTBs are used to implement amplifier architectures such as non-inverting amplifier and differential amplifier with configurable gain. It can also be used as a comparator with configurable threshold. Opamp input and output can be brought to the pins for external connections making it possible to create designs such as active filters.

1.7.3 Universal Analog Block (UAB)

UAB is a programmable switched capacitor block. It consists of a pair of opamps, buffers, comparators, and switched capacitor arrays forming two identical networks. UAB is used to implement analog functions such as programmable gain amplifier, mixer, delta-sigma modulator, DAC, and multi-pole filter. It also contains decimators, which can be used to implement a 14-bit delta-sigma ADC. These blocks can work in Active and Sleep modes of the device.

1.7.4 Low-Power Comparators

The PSoC Analog Coprocessor has a pair of low-power comparators, which can operate in all device power modes. This functionality allows the CPU and other system blocks to be disabled while retaining the ability to monitor external voltage levels during low-power modes. Two input voltages can both come from pins, or one from an internal signal through the AMUXBUS.

1.7.5 Programmable Reference Block

The programmable reference block (PRB) generates four different reference voltages for use with the analog blocks – CTB, UAB, and SAR ADC. It uses either the internal bandgap reference voltage of 1.2 V or the V_{DDA} to generate the reference voltages.

1.8 Special Function Peripherals

1.8.1 LCD Segment Drive

The PSoC Analog Coprocessor has an LCD controller, which can drive up to eight commons and every GPIO can be configured to drive common or segment. It uses full digital methods (digital correlation and PWM) to drive the LCD segments, and does not require generation of internal LCD voltages.

1.8.2 CapSense

PSoC Analog Coprocessor devices have the CapSense feature, which allows you to use the capacitive properties of your fingers to toggle buttons and sliders. CapSense functionality is supported on all GPIO pins, in self-capacitance and mutual-capacitance modes, through a CapSense Sigma-Delta (CSD) block. The CSD also provides waterproofing capability. The CSD block can also be used to implement a 10-bit single slope ADC.

1.8.2.1 IDACs and Comparator

The CapSense block has two IDACs and a comparator with an adjustable reference, which can be used for general purposes, if CapSense is not used.

1.9 Program and Debug

The device supports programming and debugging features of the device via the on-chip SWD interface. The PSoC Creator IDE provides fully integrated programming and debugging support. The SWD interface is also fully compatible with industry standard third-party tools.

1.10 Device Feature Summary

Table 1-1 shows the PSoC Analog Coprocessor device

Table 1-1. PSoC Analog Coprocessor Device Summary

Feature	PSoC Analog Coprocessor
Maximum CPU Frequency	48 MHz
Flash	32 KB
SRAM	4 KB
GPIOs (max)	38
Smart I/O	1 Port
CapSense	Available
LCD Driver	Available
Timer, Counter, PWM (TCPWM)	8
16-bit Timer	2
32-bit Timer	1
Serial Communication Block (SCB)	3
IDAC (part of CapSense)	2
Opamp (CTB)	4
Opamp (UAB)	2
Low-Power Comparator (LPCOMP)	2
SAR ADC	12-bit, 1 Msps
Voltage DAC (VDAC)	13-bit
Programmable Voltage Reference (PVref)	4 Channels
Watch Crystal Oscillator (WCO)	Available
Power Modes	Active, Sleep, and Deep-Sleep

summary.

2. Getting Started

2.1 Support

Free support for PSoC® Analog Coprocessor products is available online at www.cypress.com/products/PSoC-Analog-Coprocessor. Resources include training seminars, discussion forums, application notes, PSoC consultants, CRM technical support email, knowledge base, and application support engineers.

For application assistance, visit www.cypress.com/support/ or call 1-800-541-4736.

2.2 Product Upgrades

Cypress provides scheduled upgrades and version enhancements for PSoC Creator free of charge. Upgrades are available from your distributor on DVD-ROM; you can also download them directly from www.cypress.com/psoccreator. Critical updates to system documentation are also provided in the Documentation section.

2.3 Development Kits

The Cypress Online Store contains development kits, C compilers, and the accessories you need to successfully develop PSoC projects. Visit the Cypress Online Store website at www.cypress.com/cypress-store. Under **Products**, click **Programmable System-on-Chip** to view a list of available items. Development kits are also available from Digi-Key, Avnet, Arrow, and Future.

2.4 Application Notes

Refer to application note *AN211293 - Getting Started with PSoC Analog Coprocessor* for additional information on the PSoC Analog Coprocessor capabilities and to quickly create a simple application using PSoC Creator and PSoC development kits.

Refer to application note *AN211294 - AFE Implementation Using PSoC Analog Coprocessor* for the details on interfacing PSoC Analog Coprocessor with different analog sensors.

3. Document Construction

This document includes the following sections:

3.1 Major Sections

For ease of use, information is organized into sections and chapters that are divided according to device functionality.

- Section – Presents the top-level architecture, how to get started, and conventions and overview information of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These are the detailed implementation and use information for some aspect of the integrated circuit.
- Glossary – Defines the specialized terminology used in this technical reference manual (TRM). Glossary terms are presented in bold, italic font throughout.
- Registers Technical Reference Manual – Supplies all device register details summarized in the technical reference manual. This is an additional document.

3.2 Documentation Conventions

This document uses only four distinguishing font types, besides those found in the headings.

- The first is the use of *italics* when referencing a document title or file name.
- The second is the use of ***bold italics*** when referencing a term described in the Glossary of this document.
- The third is the use of Times New Roman font, distinguishing equation examples.
- The fourth is the use of `Courier New` font, distinguishing code examples.

3.2.1 Register Conventions

Register conventions are detailed in the *PSoC Analog Coprocessor Family Registers TRM*.

3.2.2 Numeric Naming

Hexadecimal numbers are represented with all letters in uppercase with an appended lowercase ‘h’ (for example, ‘14h’ or ‘3Ah’) and *hexadecimal* numbers may also be represented by a ‘0x’ prefix, the *C* coding convention. Binary numbers have an appended lowercase ‘b’ (for example, 01010100b’ or ‘01000011b’). Numbers not indicated by an ‘h’ or ‘b’ are *decimal*.

3.2.3 Units of Measure

This table lists the units of measure used in this document.

Table 3-1. Units of Measure

Abbreviation	Unit of Measure
bps	bits per second
°C	degrees Celsius
dB	decibels
fF	femtofarads
Hz	Hertz
k	kilo, 1000
K	kilo, 2^10
KB	1024 bytes, or approximately one thousand bytes
Kbit	1024 bits
kHz	kilohertz (32.000)
kΩ	kilohms
MHz	megahertz
MΩ	megaohms
µA	microamperes
µF	microfarads
µs	microseconds
µV	microvolts
µVrms	microvolts root-mean-square
mA	milliamperes
ms	milliseconds
mV	millivolts
nA	nanoamperes
ns	nanoseconds
nV	nanovolts
Ω	ohms
pF	picofarads
pp	peak-to-peak
ppm	parts per million
SPS	samples per second
σ	sigma: one standard deviation
V	volts

3.2.4 Acronyms

This table lists the acronyms used in this document

Table 3-2. Acronyms

Acronym	Definition
ABUS	analog output bus
AC	alternating current
ADC	analog-to-digital converter
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an ARM data transfer bus
API	application programming interface
APOR	analog power-on reset
BC	broadcast clock
BOD	brownout detect
BOM	bill of materials
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CAN	controller area network
CI	carry in
CMP	compare
CO	carry out
COM	LCD common signal
CPU	central processing unit
CRC	cyclic redundancy check
CSD	CapSense sigma delta
CT	continuous time
CTB	continuous time block
CTBm	continuous time block mini
DAC	digital-to-analog converter
DAP	debug access port
DC	direct current
DI	digital or data input
DMA	direct memory access
DMIPS	Dhrystone million instructions per second
DNL	differential nonlinearity
DO	digital or data output
DSI	digital signal interface
DSM	deep-sleep mode
DW	data wire
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
EMIF	external memory interface
FB	feedback
FIFO	first in first out
FSR	full scale range
GPIO	general purpose I/O
HCI	host-controller interface
HFCLK	high-frequency clock
HSIOM	high-speed I/O matrix
I^2C	inter-integrated circuit
IDE	integrated development environment
ILO	internal low-speed oscillator
ITO	indium tin oxide
IMO	internal main oscillator
INL	integral nonlinearity
I/O	input/output

Table 3-2. Acronyms *(continued)*

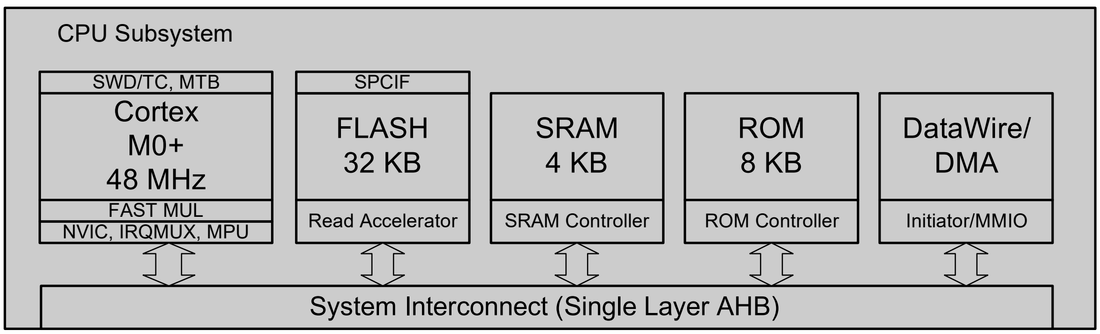
Acronym	Definition
IOR	I/O read
IOW	I/O write
IRES	initial power on reset
IRA	interrupt request acknowledge
IRQ	interrupt request
ISR	interrupt service routine
IVR	interrupt vector read
LCD	liquid crystal display
LFCLK	low-frequency clock
LPCOMP	low-power comparator
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
MISO	master-in-slave-out
MMIO	memory mapped input/output
MOSI	master-out-slave-in
MPU	memory protection unit
MSb	most significant bit
MSB	most significant byte
MSP	main stack pointer
NMI	non-maskable interrupt
NVIC	nested vectored interrupt controller
PC	program counter
PCB	printed circuit board
PCH	program counter high
PCL	program counter low
PD	power down
PGA	programmable gain amplifier
PM	power management
PMA	PSoC memory arbiter
POR	power-on reset
PPOR	precision power-on reset
PRS	pseudo random sequence
PSoC®	Programmable System-on-Chip
PSP	process stack pointer
PSRR	power supply rejection ratio
PSSDC	power system sleep duty cycle
PWM	pulse width modulator
RAM	random-access memory
RETI	return from interrupt
RF	radio frequency
ROM	read only memory
RMS	root mean square
RW	read/write
SAR	successive approximation register
SEG	LCD segment signal
SC	switched capacitor
SCB	serial communication block
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SYSCLK	system clock
SWD	single wire debug
TC	terminal count
TCPWM	timer, counter, PWM
TD	transaction descriptors
TIA	trans-impedance amplifier
UAB	universal analog block
UART	universal asynchronous receiver/transmitter
UDB	universal digital block
USB	universal serial bus
USBIO	USB I/O
VTOR	vector table offset register
WCO	watch crystal oscillator
WDT	watchdog timer
WDR	watchdog reset
XRES	external reset
XRES_N	external reset, active low

Section B: CPU System

This section encompasses the following chapters:

Top Level Architecture

CPU System Block Diagram



4. Cortex-M0+ CPU

The PSoC® Analog Coprocessor ARM Cortex-M0+ core is a 32-bit CPU optimized for low-power operation. It has an efficient two-stage pipeline, a fixed 4-GB memory map, and supports the ARMv6-M Thumb instruction set. The Cortex-M0+ also features a single-cycle 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

This section gives an overview of the Cortex-M0+ processor. For more details, see the ARM Cortex-M0+ user guide or technical reference manual, both available at www.arm.com.

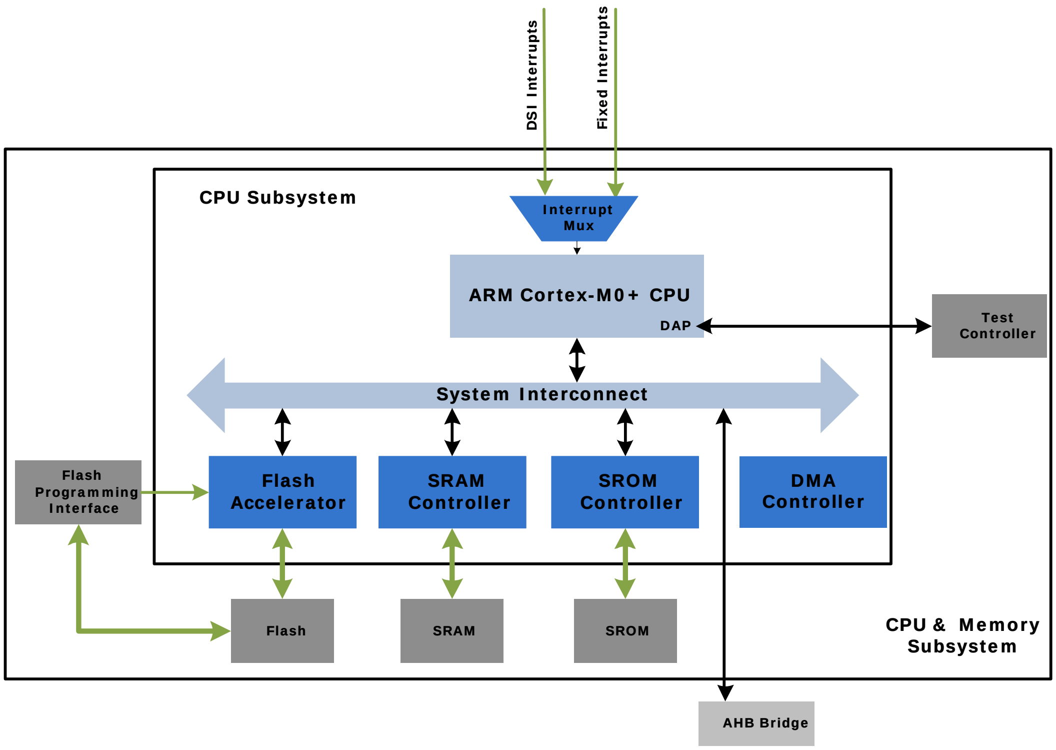
4.1 Features

The PSoC Cortex-M0+ has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Supports the Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Implements design time configurable Memory Protection Unit (MPU)
- Supports unprivileged and privileged mode execution
- Supports optional Vector Table Offset Register (VTOR)
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

4.2 Block Diagram

Figure 4-1. CPU Subsystem Block Diagram



4.3 How It Works

The Cortex-M0+ is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. It supports most 16-bit instructions in the Thumb instruction set and some 32-bit instructions in the Thumb-2 instruction set.

The processor supports two operating modes (see Operating Modes on page 31). It has a single-cycle 32-bit multiplication instruction.

4.4 Address Map

The ARM Cortex-M0+ has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in Table 4-1. Note that code can be executed from the code and SRAM regions.

Table 4-1. Cortex-M0+ Address Map

Address Range	Name	Use
0x00000000 - 0x1FFFFFFF	Code	Program code region. You can also place data here. Includes the exception vector table, which starts at address 0.
0x20000000 - 0x3FFFFFFF	SRAM	Data region. You can also execute code from this region.
0x40000000 - 0x5FFFFFFF	Peripheral	All peripheral registers. You cannot execute code from this region.
0x60000000 - 0xDFFFFFFF		Not used.
0xE0000000 - 0xE00FFFFF	PPB	Peripheral registers within the CPU core.
0xE0100000 - 0xFFFFFFFF	Device	PSoC implementation-specific.

4.5 Registers

The Cortex-M0+ has sixteen 32-bit registers, as Table 4-2 shows:

- R0 to R12 – General-purpose registers. R0 to R7 can be accessed by all instructions; the other registers can be accessed by a subset of the instructions.
- R13 – Stack pointer (SP). There are two stack pointers, with only one available at a time. In thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP).
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

Table 4-2. Cortex-M0+ Registers

Name	Type[a]	Reset Value	Description
R0-R12	RW	Undefined	R0-R12 are 32-bit general-purpose registers for data operations.
MSP (R13) PSP (R13)	RW	[0x00000000]	The stack pointer (SP) is register R13. In thread mode, bit[1] of the CONTROL register indicates which 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 0x00000000.
LR (R14)	RW	Undefined	The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004]	The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.
PSR	RW	Undefined	The program status register (PSR) combines: Application Program Status Register (APSR). Execution Program Status Register (EPSR). Interrupt Program Status Register (IPSR).
APSR	RW	Undefined	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	[0x00000004].0	On reset, EPSR is loaded with the value bit[0] of the register [0x00000004].
IPSR	RO	0	The IPSR contains the exception number of the current ISR.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL register controls the stack used when the processor is in thread mode.

a. Describes access type during program execution in thread mode and handler mode. Debug access can differ.

Table 4-3 shows how the PSR bits are assigned.

Table 4-3. Cortex-M0+ PSR Bit Assignments

Bit	PSR Register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSR	V	Overflow flag

Table 4-3. Cortex-M0+ PSR Bit Assignments

Bit	PSR Register	Name	Usage
27 – 25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23 – 6	–	–	Reserved
5 – 0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = NMI 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 47 = 40

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the Interrupts chapter on page 47 for a list of exceptions.

4.6 Operating Modes

The Cortex-M0+ processor supports two operating modes:

- Thread Mode – used by all normal applications. In this mode, the MSP or PSP can be used. The CONTROL register bit 1 determines which stack pointer is used:
  - 0 = MSP is the current stack pointer
  - 1 = PSP is the current stack pointer
- Handler Mode – used to execute exception handlers. The MSP is always used.

In thread mode, use the MSR instruction to set the stack pointer bit in the CONTROL register. When changing the stack pointer, use an ISB instruction immediately after the MSR instruction. This action ensures that instructions after the ISB execute using the new stack pointer.

In handler mode, explicit writes to the CONTROL register are ignored, because the MSP is always used. The exception entry and return mechanisms automatically update the CONTROL register.

4.7 Instruction Set

The Cortex-M0+ implements a version of the Thumb instruction set, as Table 4-4 shows. For details, see the Cortex-M0+ Generic User Guide.

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. Many instructions are unable to use, or have restrictions on using, the PC or SP for the operands or destination register.

Table 4-4. Thumb Instruction Set

Mnemonic	Brief Description
ADCS	Add with carry
ADD{S}[a]	Add
ADR	PC-relative address to register
ANDS	Bit wise AND
ASRS	Arithmetic shift right
B{cc}	Branch {conditionally}
BICS	Bit clear
BKPT	Breakpoint
BL	Branch with link
BLX	Branch indirect with link
BX	Branch indirect
CMN	Compare negative
CMP	Compare
CPSID	Change processor state, disable interrupts
CPSIE	Change processor state, enable interrupts
DMB	Data memory barrier
DSB	Data synchronization barrier

Table 4-4. Thumb Instruction Set

Mnemonic	Brief Description
EORS	Exclusive OR
ISB	Instruction synchronization barrier
LDM	Load multiple registers, increment after
LDR	Load register from PC-relative address
LDRB	Load register with word
LDRH	Load register with half-word
LDRSB	Load register with signed byte
LDRSH	Load register with signed half-word
LSLS	Logical shift left
LSRS	Logical shift right
MOV{S}[a]	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result
MVNS	Bit wise NOT
NOP	No operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-reverse word
REV16	Byte-reverse packed half-words
REVSH	Byte-reverse signed half-word
RORS	Rotate right
RSBS	Reverse subtract
SBCS	Subtract with carry
SEV	Send event
STM	Store multiple registers, increment after
STR	Store register as word
STRB	Store register as byte
STRH	Store register as half-word
SUB{S}[a]	Subtract
SVC	Supervisor call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND-based test
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait for event
WFI	Wait for interrupt

a. The ‘S’ qualifier causes the ADD, SUB, or MOV instructions to update APSR condition flags.

4.7.1 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 half-word-aligned address is used for a half-word access. Byte accesses are always aligned.

No support is provided for unaligned accesses on the Cortex-M0+ processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

4.7.2 Memory Endianness

The Cortex-M0+ uses the little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

4.8 Systick Timer

The Systick timer is integrated with the NVIC and generates the SYSTICK interrupt. This interrupt can be used for task management in a real-time system. The timer has a reload register with 24 bits available to use as a countdown value. The Systick timer uses either the Cortex-M0+ internal clock or the low-frequency clock (LF_CLK) as the source.

4.9 Debug

PSoC contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

5. DMA Controller Modes

The DMA controller provides DataWire (DW) and Direct Memory Access (DMA) functionality. The DMA controller has the following features:

- Supports eight DMA channels
- Four levels of priority for each channel
- Byte, half-word (2 bytes), and word (4 bytes) transfers
- Three modes of operation supported for each channel
- Configurable interrupt generation
- Output trigger on completion of transfer
- Transfer sizes up to 65,536 data elements

The DMA controller supports three operation modes. These operational modes are different in how the DMA controller operates on a single trigger signal. These operating modes allow the user to implement different operation scenarios for the DMA. The operation modes are

- Mode 0: Single data element per trigger
- Mode 1: All data elements per trigger
- Mode 2: All data elements per trigger and automatically trigger chained descriptor

The data transfer specifics, such as source and destination address locations and the size of the transfer, are specified by a descriptor structure. Each channel has an independent descriptor structure.

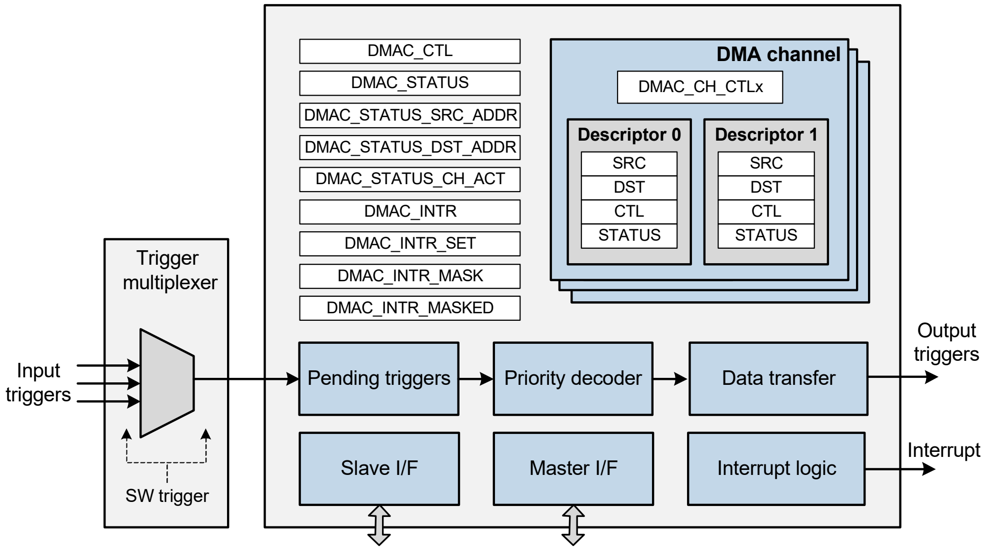
The DMA controller provides Active/Sleep functionality and is not available in the Deep-Sleep power mode.

5.1 Block Diagram Description

The DMA transfers data to and from memory, peripherals, and registers. These transfers occur independent of the CPU. The DMA can transfer up to 65,536 data elements in one transfer. These data elements can be 8-bit, 16-bit, or 32-bit wide. The DMA starts each transaction through an external trigger that can come from a DMA channel (including itself), another DMA channel, a peripheral, or the CPU. The DMA is best used to offload data transfer tasks from the CPU.

Figure 5-1 gives an overview of the DMA controller at a block level.

Figure 5-1. DMA Controller Block Diagram



Every DMA channel has two descriptors, which are responsible for configuring parameters specific to the transfer, such as source address, destination address, and data width. The transfer initiation in the DMA channel is on a trigger event. The trigger signals can come from different peripherals in the device, including the DMA itself.

The DMA controller has two bus interfaces, the master interface and the slave interface. Master I/F is an AHB-Lite bus master, which allows the DMA controller to initiate AHB-Lite data transfers to the source and destination locations. The DMA is the bus master in the master interface. This is the interface through which all DMA transfers are accomplished.

The DMA configuration registers and descriptors are accessed and reconfigured through the slave interface. Slave I/F is an AHB-Lite bus slave, which allows the PSoC main CPU to access the DMA controller's control/status registers and to access the descriptor structure. CPU is generally the master for this bus.

The receipt of a trigger activates a state machine in the DMA controller that goes through a trigger prioritization and processing and then initiates a data transfer according to the descriptor setting. When a transfer is complete, an output trigger is generated, which can be used as trigger condition or event for starting another function.

The DMA controller also has an interrupt logic block. Only one interrupt line is available from the DMA controller to interrupt the CPU. Individual DMA descriptors can be configured so that they activate this interrupt line on completion of the transfer.

5.1.1 Trigger Sources and Multiplexing

Every DMA channel has an input and output trigger associated with it. The input trigger can come from any peripheral, CPU, or a DMA channel itself. The input trigger is used to trigger a DMA transfer, as defined by the 5.2.4 Transfer Mode. A 'logic high', on the trigger input will trigger the DMA channel. The minimum width of this 'logic high' is two system clock cycles. The deactivation setting configures the nature of trigger deactivation.

The output trigger signals the completion of a transfer. This signal can be used as a trigger to a DMA channel or as a digital signal to the digital interconnect. The trigger input can come from different sources and is routed through a 5.1.1.1 Trigger Multiplexer.

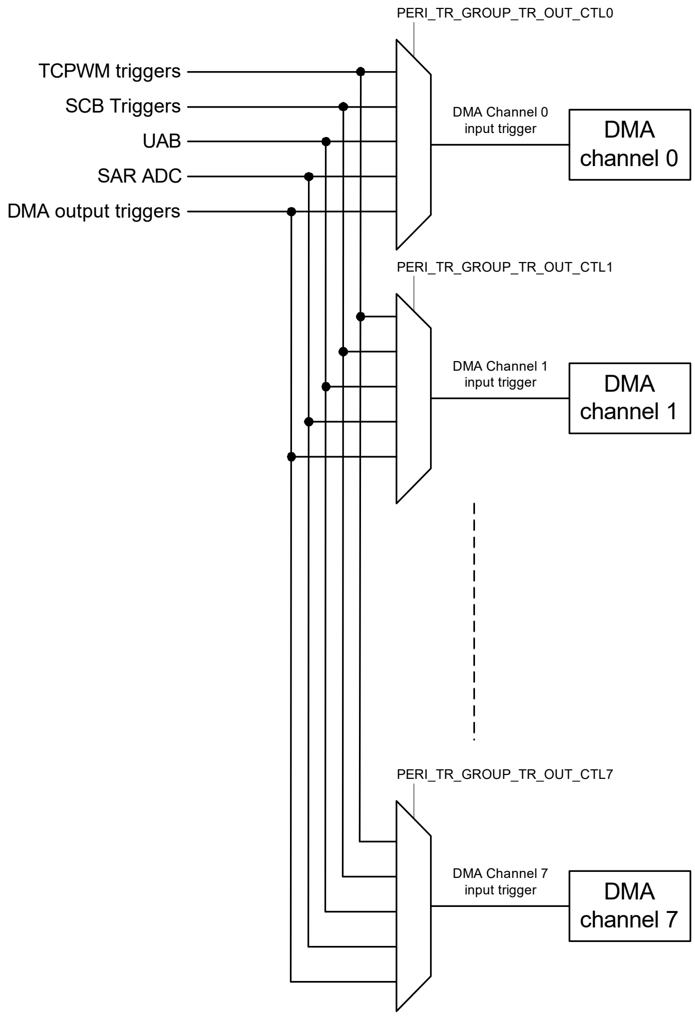
5.1.1.1 Trigger Multiplexer

The DMA channels can have trigger inputs from different peripheral sources in the PSoC. This is routed to the individual DMA channel trigger inputs through the trigger multiplexer.

In the DMA trigger, multiplexers are organized in trigger groups. Each trigger group is composed of multiple multiplexers feeding into the individual DMA channel trigger inputs.

The PSoC Analog Coprocessor implements a single trigger group (Trigger group 0), which provides trigger inputs to the DMA. The trigger input options can come from TCPWM, SAR ADC, SCB, UAB, and DMA output triggers. Figure 5-2 shows the trigger multiplexer implementation.

Figure 5-2. Trigger Multiplexer Implementation



The trigger source for individual DMA channels is selected in the PERI_TR_GROUP_TR_OUT_CTLx[5:0] register. Table 5-1 provides the trigger multiplexers.

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
x[5:0]	Trigger Source
0	Software trigger
1	TCPWM 0 overflow
2	TCPWM 1 overflow
3	TCPWM 2 overflow
4	TCPWM 3 overflow
5	TCPWM 4 overflow
6	TCPWM 5 overflow
7	TCPWM 6 overflow
8	TCPWM 7 overflow

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
9	TCPWM 0 compare match
10	TCPWM 1 compare match
11	TCPWM 2 compare match
12	TCPWM 3 compare match
13	TCPWM 4 compare match
14	TCPWM 5 compare match
15	TCPWM 6 compare match
16	TCPWM 7 compare match
17	TCPWM 0 underflow
18	TCPWM 1 underflow
19	TCPWM 2 underflow
20	TCPWM 3 underflow
21	TCPWM 4 underflow

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
22	TCPWM 5 underflow
23	TCPWM 6 underflow
24	TCPWM 7 underflow
25	SCB 0 TX request
26	SCB 0 RX request
27	SCB 1 TX request
28	SCB 1 RX request
29	SCB 2 TX request
30	SCB 2 RX request
31	DMA Channel 0 trigger out
32	DMA Channel 1 trigger out
33	DMA Channel 2 trigger out
34	DMA Channel 3 trigger out
35	DMA Channel 4 trigger out
36	DMA Channel 5 trigger out
37	DMA Channel 6 trigger out
38	DMA Channel 7 trigger out
39	SAR ADC sample done
40	UAB valid signal 0
41	UAB valid signal 1
42	UAB trigger out 0
43	UAB trigger out 1
44	UAB-DAC Interrupt 0
45	UAB-DAC Interrupt 1
46	Decimator Interrupt 0
47	Decimator Interrupt 1

5.1.1.2 Creating Software Triggers

Every DMA channel has a trigger input and output trigger associated with it. This trigger input can come from any trigger group, as described in “Trigger Multiplexer” on page 34. A software trigger for the DMA channel is implemented using the trigger input option 0 in the trigger multiplexer settings. When PERI_TR_GROUP_TR_OUT_CTLx [5:0] is zero, the DMA trigger is configured for a software trigger. The DMA channel is then triggered using the PERI_TR_CTL register.

5.1.2 Pending Triggers

When a DMA channel is already operational and a trigger event is encountered, the DMA channel corresponding to the trigger is put into a pending state. Pending triggers keep track of activated triggers by locally storing them in pending bits. This is essential, because multiple channel triggers may be activated simultaneously, whereas only one channel can be served by the data transfer engine at a time. This block enables the use of both level-sensitive and pulse-sensitive triggers.

The pending triggers are registered in the status register (DMAC_STATUS_CH_ACT).

5.1.3 Output Triggers

Each channel has an output trigger. This trigger is high for two system clock cycles. The trigger is generated on the completion of a data transfer. At the system level, these output triggers can be connected to the trigger multiplexer component. This connection allows for a DMA controller output trigger to be connected to a DMA controller input trigger. In other words, the completion of a transfer in one channel can activate another channel or even reactivate the same channel.

5.1.4 Channel Prioritization

When there are multiple channels with active triggers, the channel priority is used to determine which channel gets the access to the data transfer engine. The priorities are set for each channel using the PRIO field of the channel control register (DMAC_CH_CTL), with ‘0’ representing the highest priority and ‘3’ representing the lowest priority. Priority decoding uses the channel priority to determine the highest priority activated channel. If multiple activated channels have the same highest priority, the channel with the lowest index ‘i’, is considered the highest priority activated channel.

5.1.5 Data Transfer Engine

The data transfer engine is responsible for the data transfer from a source location to a destination location. When idle, the data transfer engine is ready to accept the highest priority activated channel. The configuration of the data transfer is specified by the descriptor. The data transfer engine implements a state machine, which has the following states.

- State 0 - Default State: This is the idle state of the DMA controller, where it waits for a trigger condition to initiate transfer.
- State 1 - Load Descriptor: When a trigger condition is encountered and priority is resolved, the data transfer engine enters the load descriptor state. In this state, the active descriptor (SRC, DST, and CTL) is loaded into the DMA controller to initiate the transfer. The DMAC_STATUS, DMAC_STATUS_SRC_ADDR and DMAC_STATUS_DST_ADDR, and STATUS_CH_ACT will also reflect the currently active status.
- State 2 - Loading data from source: The data transfer engine uses the master I/F to load data from the source location.
- State 3 - Storing data at destination: The data transfer engine uses the master I/F to store data to the destination location.

Depending on the Transfer mode, State 2 and 3 may be performed multiple times.

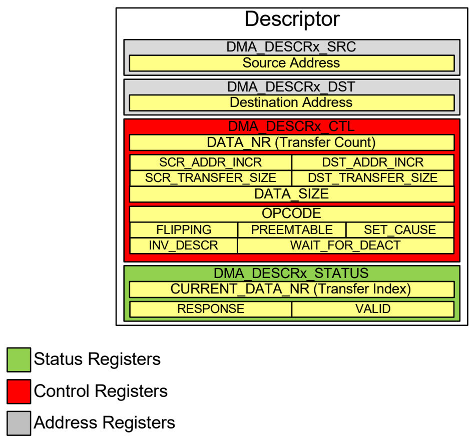
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two cycles of the trigger activation. If it was set to 'wait indefinitely', the DMA controller will remain in this state until the trigger signal has gone low.
- State 6 - Storing Descriptor Response: In this phase, the data transfer according to the descriptor is completed and an interrupt may be generated if it was configured to do so. The Response field in DMAC_DESCR_PING_STATUS or DMAC_DESCR_PONG_STATUS is also populated and the state transitions to State 0.

5.2 Descriptors

The data transfer between a source and a destination in a channel is configured using a descriptor. Each channel in the DMA has two descriptors named PING and PONG descriptors (also called Descriptor 0 and Descriptor 1 in this document). A descriptor is a set of four 32-bit registers that contain the configuration for the transfer in the associated channel.

Figure 5-3 shows the structure of a descriptor.

Figure 5-3. Descriptor Structure



5.2.1 Address Configuration

Figure 5-4 demonstrates the use of the descriptor settings for the address configuration of a transfer.

Source and Destination Address: The Source and Destination addresses are set in the respective registers in the descriptor. These set the base addresses for the source and destination location for the transfer. In case the descriptor is configured to transfer a single element, this field holds the source/destination address of the data element. If the descriptor is configured to transfer multiple elements with source address or destination address or both in an incremental mode, this field will hold the address of the first element that is transferred.

Data Number (DATA_NR): This is a transfer count parameter. DATA_NR is a 16-bit number, which determines the number of elements to be transferred before a descriptor is defined as completed. In a typical use case, this setting is the buffer size of a transfer.

Source Address Increment (SCR_ADDR_INC): This is a bit setting in the control register, which determines if a source address is incremented between each data element transfer. This feature is enabled when the source of the data is a buffer and each transfer element needs to be fetched from subsequent locations in the memory. In this case, the Source Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the SCR_TRANSFER_SIZE setting described in 5.2.2 Transfer Size on page 38.

Destination Address Increment (DST_ADDR_INC): This is a bit setting in the control register, which determines if a destination address is incremented between each element transfer. This feature is enabled when the destination of the data is a buffer and each transfer element needs to be transferred to subsequent locations in the memory. In this case, the Destination Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the DST_TRANSFER_SIZE setting described in 5.2.2 Transfer Size on page 38.

Invalidate Descriptor (INV_DESCR): When this bit is set, the descriptor transfers all data elements and clears the descriptor's VALID bit, making it invalid. This feature affects the VALID bit in the DMA_DESCRx_STATUS register. This setting is used in cases where the user expects the descriptor to get invalidated after its transfer is complete. The descriptor can be made valid again in firmware by setting the VALID bit in the descriptor's STATUS register.

Preemptable (PREEMPTABLE): If disabled, the current transfer as defined by Operational mode is allowed to complete undisturbed. If enabled, the current transfer as defined by Operation Mode can be preempted/interrupted by a DMA channel of higher priority. When this channel is preempted, it is set as pending and will run the next time its priority is the highest.

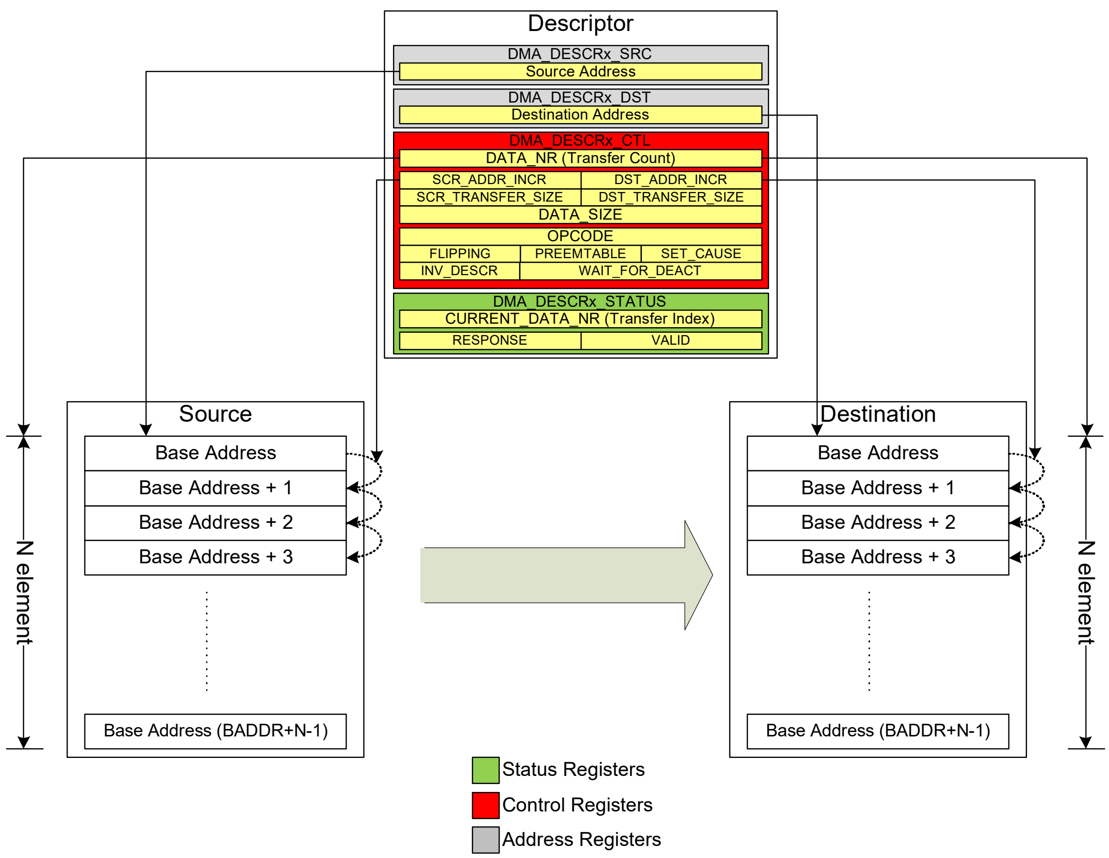
Setting Interrupt Cause (SET_CAUSE): When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

Trigger Type (WAIT_FOR_DEACT): When the DMA transfer based on the descriptor is completed, the data transfer engine checks the state of trigger deactivation. This is corresponding to State 5 of the data transfer engine. See 5.1.5 Data Transfer Engine on page 36. The type of DMA input trigger will determine when the trigger signal is considered deactivated. The DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger state is deactivated. This field is used to synchronize the controller's data transfer(s) with the agent that generated the trigger.

This field is ONLY used on completion of a descriptor execution and has four settings:

- 0 - Pulse Trigger: Do not wait for deactivation.
- 1 - Level-sensitive waits four SYSCLK cycles: The DMA trigger is deactivated after the level trigger signal is detected for four cycles.
- 2 - Level-sensitive waits eight SYSCLK cycles: The DMA transfer is initiated after the level trigger signal is detected for eight cycles.
- 3 - Pulse trigger waits indefinitely for deactivation. The DMA transfer is initiated after the trigger signal deactivates.

Figure 5-4. DMA Transfer: Address Configuration



5.2.2 Transfer Size

The transfer word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter

(DATA_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes, set by SCR_TRANSFER_SIZE and DST_TRANSFER_SIZE, can have a value of either the DATA_SIZE or 32 bit. DATA_SIZE can be set to a 32-bit, 16-bit, or 8-bit setting.

The data width of most peripheral registers is 4 bytes (32 bit); therefore, SCR_TRANSFER_SIZE or DST_TRANSFER_SIZE should typically be set to 32 bit when DMA is using a peripheral as its source or destination. The source and destination transfer size for the DMA component must match the addressable width of the source and destination, regardless of the width of data that needs to be moved. The DATA_SIZE parameter will correspond to the width of the actual data. For example, if a 16-bit PWM is used as a destination for DMA data, the DST_TRANSFER_SIZE must be set to 32 bit to match the width of the PWM register, because the peripheral register width for the TCPWM block (and most peripherals) is always 32 bit. However, in this example the DATA_SIZE for the destination may still be set to 16 bit because the 16-bit PWM only uses 2 bytes of data. SRAM and flash are 8-bit, 16-bit, or 32-bit addressable and can use any source and destination transfer sizes to match the needs of the application.

Table 5-2 summarizes the possible combinations of the transfer size settings and its description.

Table 5-2. Transfer Size Settings

DATA_SIZE	SCR_TRANSFER_SIZE	DST_TRANSFER_SIZE	Typical Usage	Description
8-bit	8-bit	8-bit	Memory to Memory	No data manipulation
8-bit	32-bit	8-bit	Peripheral to Memory	Higher 24 bits from the source dropped
8-bit	8-bit	32-bit	Memory to Peripheral	Higher 24 bits zero padded at destination
8-bit	32-bit	32-bit	Peripheral to Peripheral	Higher 24 bits from the source dropped and higher 24 bits zero padded at destination
16-bit	16-bit	16-bit	Memory to Memory	No data manipulation
16-bit	32-bit	16-bit	Peripheral to Memory	Higher 16 bits from the source dropped
16-bit	16-bit	32-bit	Memory to Peripheral	Higher 16 bits zero padded at destination
16-bit	32-bit	32-bit	Peripheral to Peripheral	Higher 16 bits from the source dropped and higher 16-bit zero padded at destination
32-bit	32-bit	32-bit	Peripheral to Peripheral	No data manipulation

5.2.3 Descriptor Chaining

Every channel has a PING and PONG descriptor, which can have a distinct setting for the associated transfer. The active descriptor is set by the PING_PONG bit in the individual channel control register (DMAC_CH_CTL). The functionality of the PING and PONG descriptors is to create a link list of descriptors. This helps create a transition from one transfer configuration to another without CPU intervention. In addition, the two descriptors mean that the CPU is free to modify the PING register when PONG register is active and vice versa.

The FLIPPING bit in a descriptor, when enabled, links it to its PING/PONG counterpart. This field is used in conjunction with the OPCODE 2 transfer mode. Therefore, when the FLIPPING bit is enabled in a PING descriptor, configured for OPCODE 2, the channel automatically executes the PONG descriptor at the end of the PING descriptor. In case the configuration is for an OPCODE 0 or OPCODE 1, a new trigger is required to start the PONG descriptor.

The use of PING PONG has more relevance in the context of transfer modes.

5.2.4 Transfer Mode

The operation of a channel during the execution of a descriptor is defined by the OPCODE settings. Three OPCODEs are possible for each channel of the DMA controller.

5.2.4.1 Single Data Element Per Trigger (OPCODE 0)

This mode is achieved when an OPCODE of 0 is configured. DMA transfers a single data element from a source location to a destination location on each trigger signal. This functionality can be used in conjunction with other settings in the descriptor such as Source and Destination increment.

Figure 5-5 shows a typical use case of this transfer. Here a UART receive (RX) register is the source and the destination is a peripheral register such as an SPI transmit (TX) register. The trigger is from the DMA request signal of the UART. When the trigger is received, the transfer engine will load data from the UART RX register and store the lower eight bits to the SPI TX register. Successive triggers will result in the same behavior because the descriptor will be rerun.

Note how the source and destination data widths are assigned as 32 bit. This is because all accesses to peripheral registers in PSoC must be 32 bit. Because the valid data width is only eight bits, the DATA_SIZE is maintained as eight bit.

Figure 5-5. OPCODE 0: Simple DMA Transfer from Peripheral to Peripheral

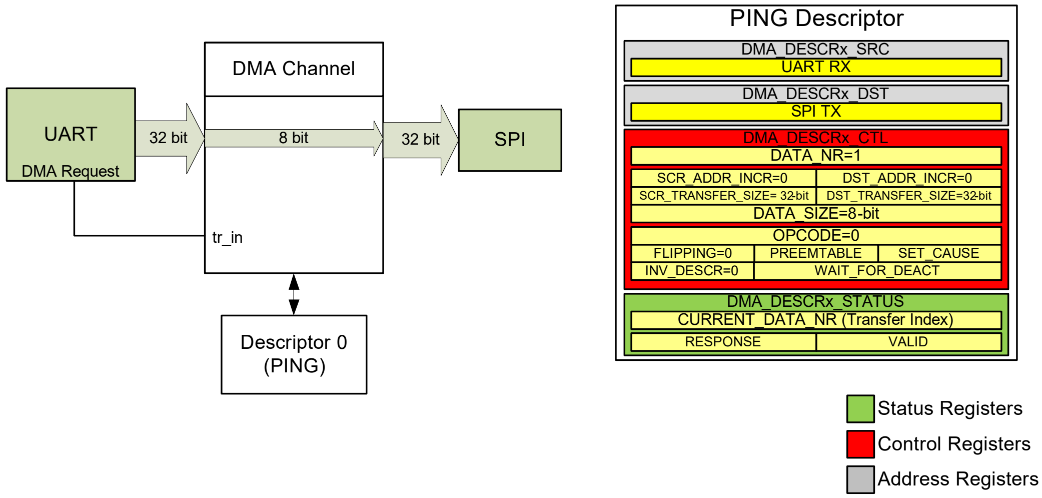
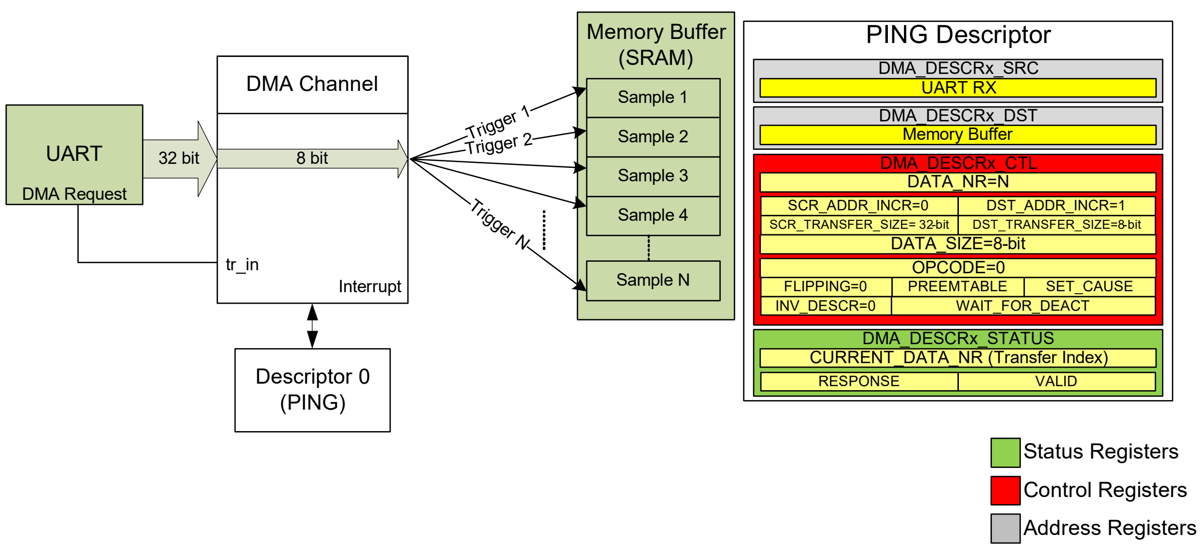


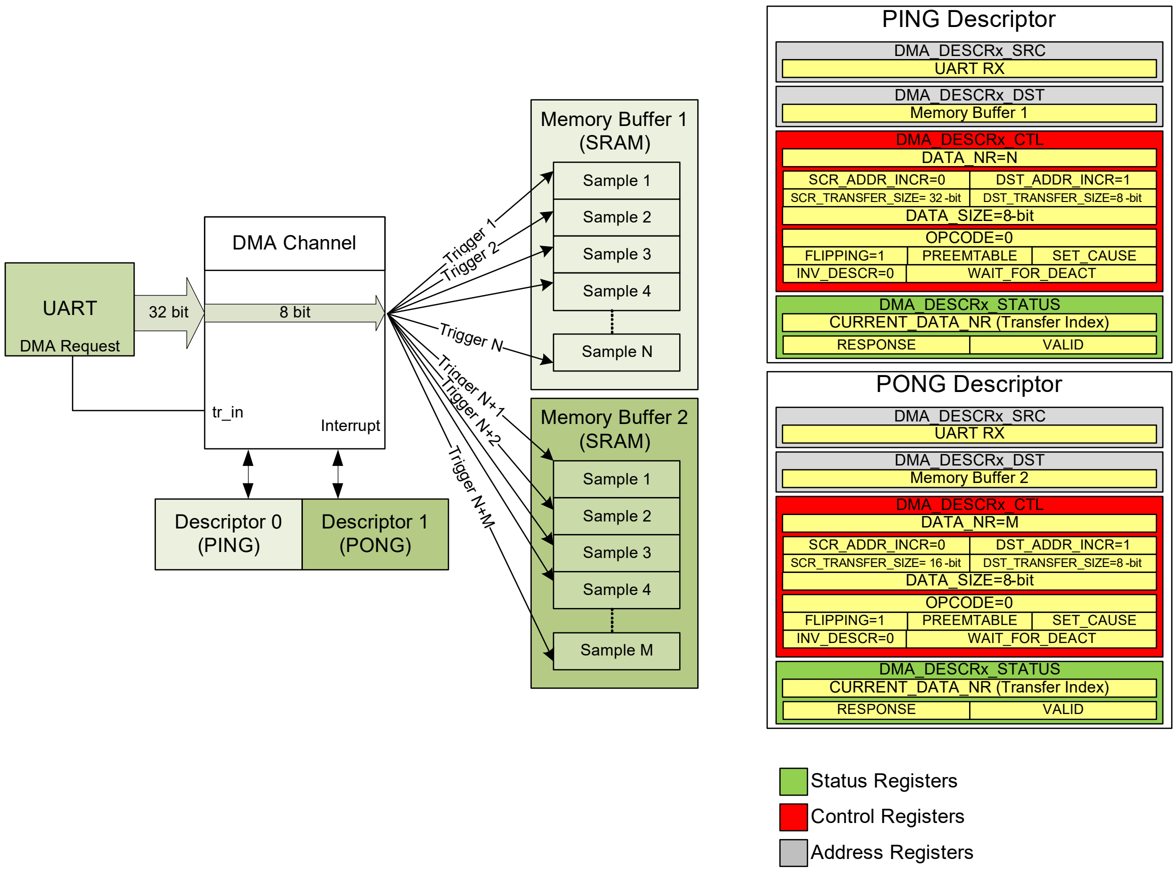
Figure 5-6 describes another use case where the data transfer is between the UART RX register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is a UART. When the trigger is received, the transfer engine will load data from the UART RX register and store to the Memory Buffer, Sample 1 memory location. Subsequent triggers will continue to store the UART data into consecutive locations from Sample 1, until the PING descriptor buffer size (DATA_NR field) is filled.

Figure 5-6. OPCODE 0: Transfer with Destination Address Increment Feature



A similar use case is shown in Figure 5-7. This demonstrates the use of the PING and PONG descriptors. On completion of the PING descriptor, the controller will flip to execute the PONG descriptor. Thus, two buffer transfers are achieved in sequence. However, note that the transfers are still done at one element transfer for every trigger.

Figure 5-7. DMA Transfer Using Flipping Feature

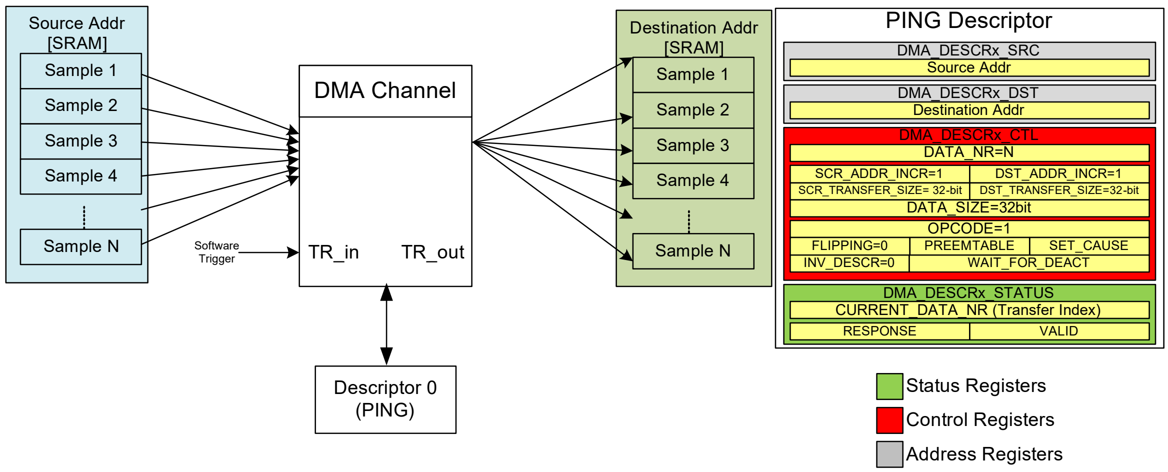


5.2.4.2 *Entire Descriptor Per Trigger (OPCODE 1)*

In this mode of operation, the DMA transfers multiple data elements from a source location to a destination location in one trigger. In OPCODE 1, the controller executes the entire descriptor in a single trigger. This type of functionality is useful in memory-to-memory buffer transfers. When the trigger condition is encountered, the transfer is continued until the descriptor is completed.

Figure 5-8 shows an OPCODE 1 transfer, which transfers the entire contents of the source buffer into the destination buffer. The entire transfer is part of a single PING descriptor and is completed on a single trigger.

Figure 5-8. DMA Transfer Example with OPCODE 1

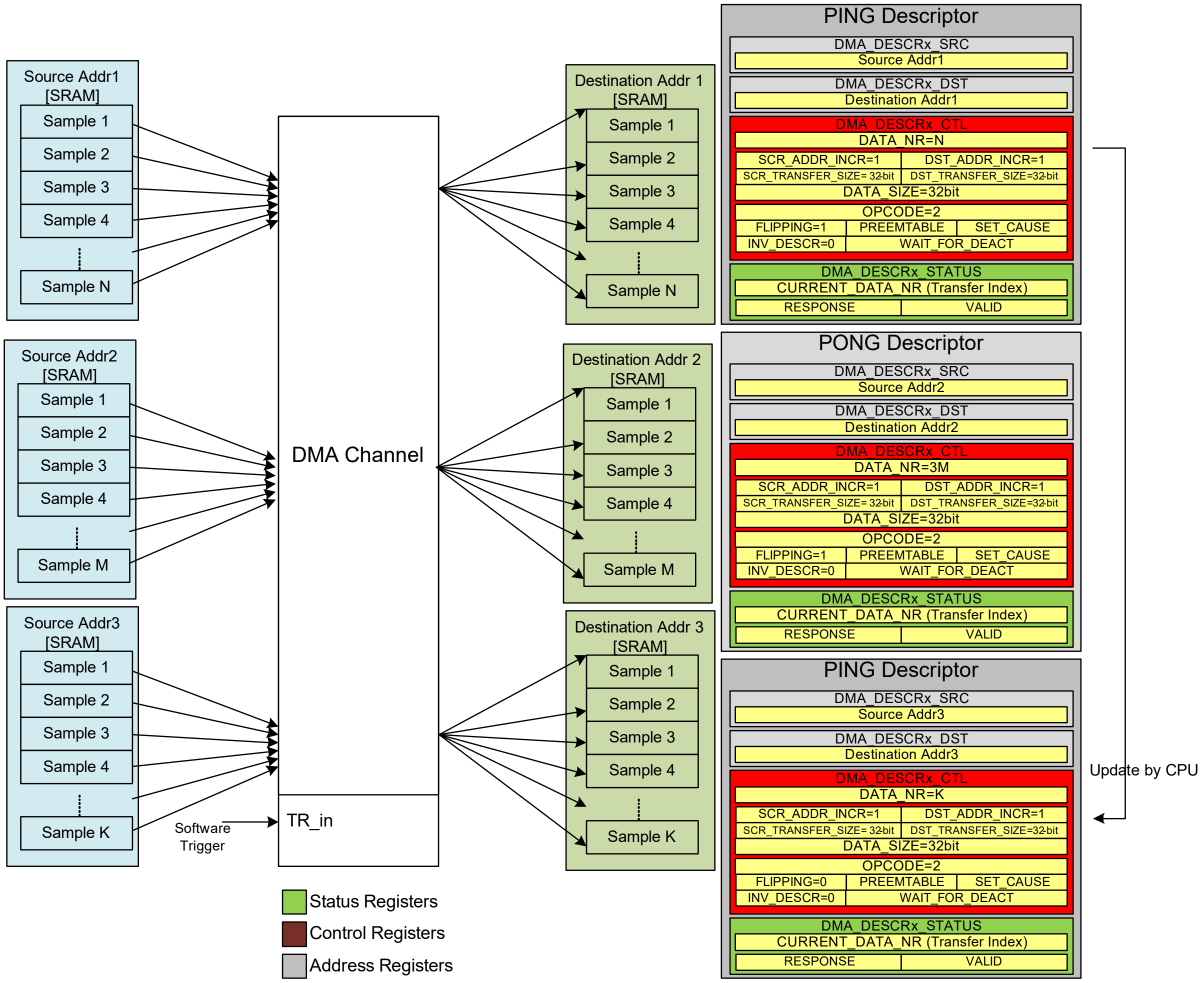


5.2.4.3 Entire Descriptor Chain Per Trigger (OPCODE 2)

OPCODE 2 is always used in conjunction with the FLIPPING field. When OPCODE 2 is used with FLIPPING enabled in a PING descriptor, a single trigger can execute a PING descriptor and automatically flip to the PONG descriptor and execute that too. If the PONG descriptor is also provided with an OPCODE 2, then the cycling between PING and PONG will continue until one of the descriptors are invalidated or changed by the CPU.

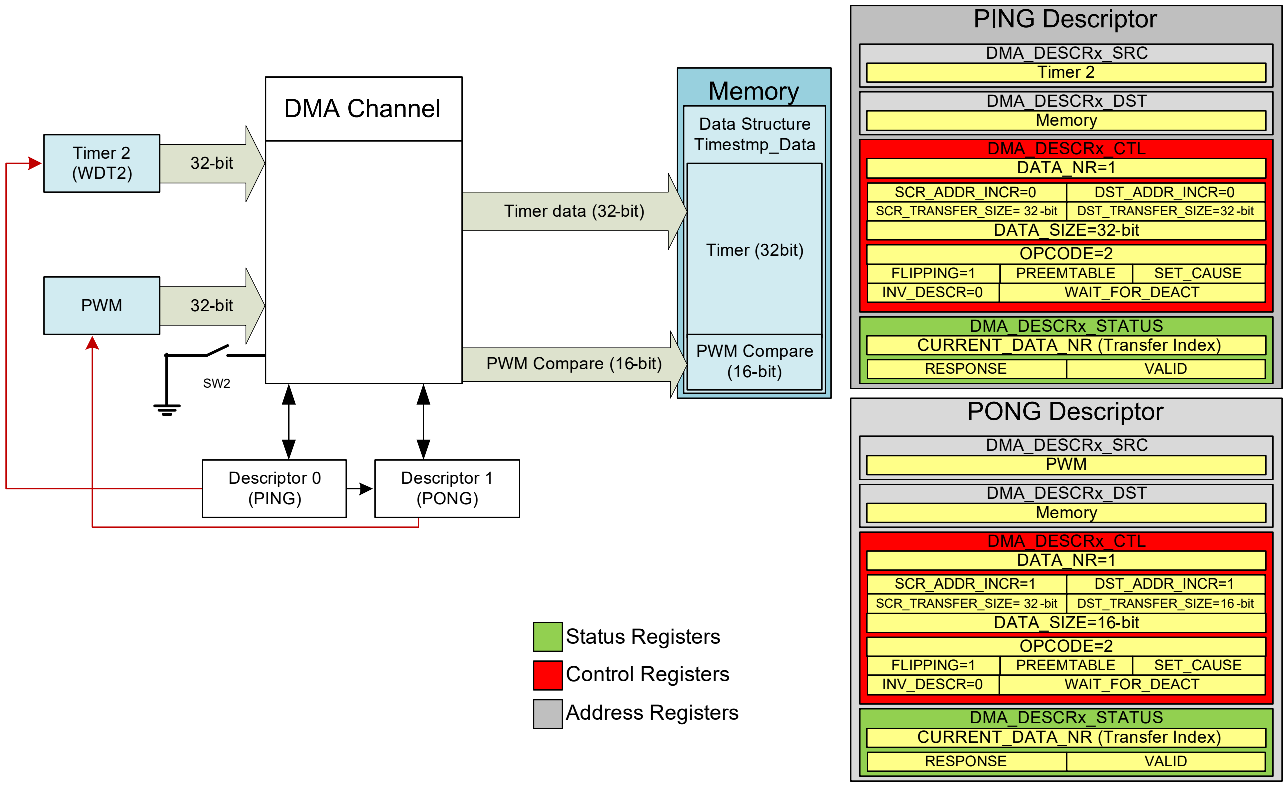
Figure 5-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

Figure 5-9. DMA Transfer Example with OPCODE 2



The OPCODE 2 transfer mode can be customized to implement distinct use cases. Figure 5-10 illustrates one such use case. Here, the source data can come from two different locations which are not consecutive memory. The destination is a data structure that is in consecutive memory locations. One source is the Timer 2, which holds a timing data and the other source is a PWM compare register. Both the data is stored in consecutive locations in memory.

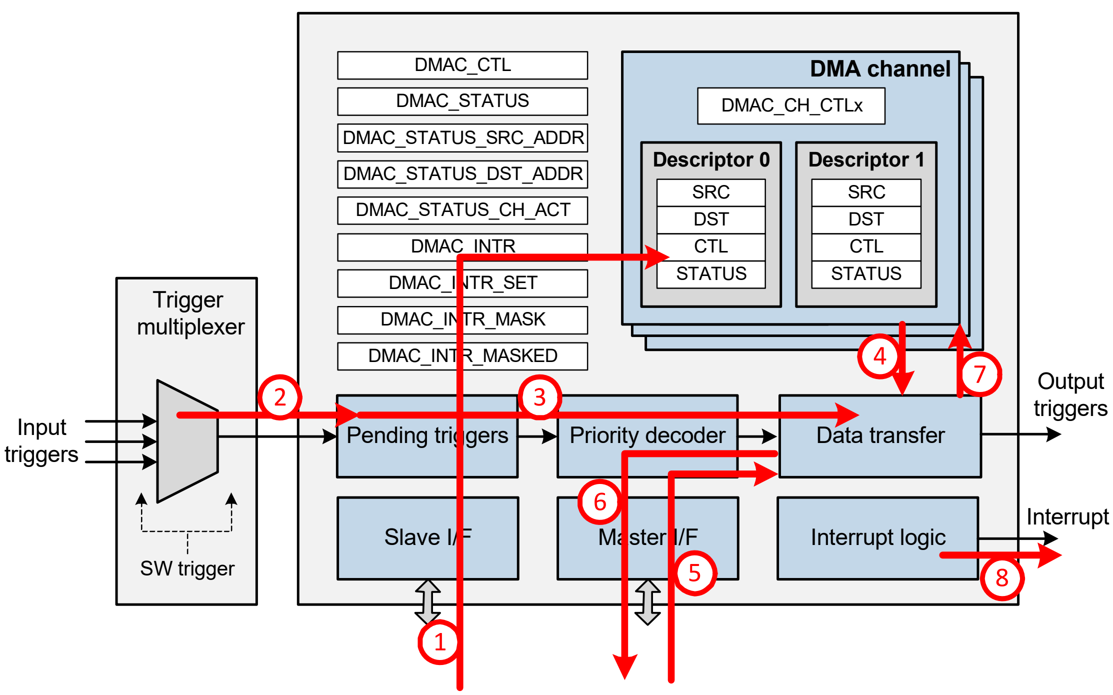
Figure 5-10. OPCODE 2: Multiple Sources to Memory



5.3 Operation and Timing

Figure 5-11 shows the DMA controller design with a trigger, data, or interrupt flow superimposed on it.

Figure 5-11. Operational Flow



The flow exemplifies the steps that are involved in a DMA controller data transfer:

1. The main CPU programs the descriptor structure for a specific channel. It also programs the DMA register that selects a specific system trigger for the channel.
2. The channel's system trigger is activated.
3. Priority decoding determines the highest priority activated channel.
4. The data transfer engine accepts the activated channel and uses the channel identifier to load the channel's descriptor structure. The descriptor structure specifies the channel's data transfers.
5. The data transfer engine uses the master I/F to load data from the source location.
6. The data transfer engine uses the master I/F to store data to the destination location. In a single element (opcode 0) transfer, steps 5 and 6 are performed once. In a multiple element descriptor (opcode 1 or 2) transfer, steps 5 and 6 may be performed multiple times in sequence to implement multiple data element transfers.
7. The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor SRAM.
8. If all the data transfers as specified by a descriptor channel structure have completed, an interrupt may be generated (this is a programmable option).

The DMA controller data transfer steps can be classified as either: initialization, concurrent, or sequential steps:

- Initialization: This includes step 1, which programs the descriptor structures. This step is done for each descriptor structure. It is performed by the main CPU and is NOT initiated by an activated channel trigger.
- Concurrent: This includes steps 2 and 3. These steps are performed in parallel for each channel.
- Sequential: This includes steps 4 through 8. These steps are performed sequentially for each activated channel. As a result, the DMA controller throughput is determined by the time it takes to perform these steps. This time consists of two parts: the time spent by the controller (to load and store the descriptor) and the time spent on the bus infrastructure. The latter time is dependent on the latency of the bus (determined by arbiter and bridge components) and the target memories/peripherals.

When transferring single data elements, it takes 12 clock cycles to complete one full transfer under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to complete a transfer in this mode is:

No of cycles = 12 + LOAD wait states + STORE wait states

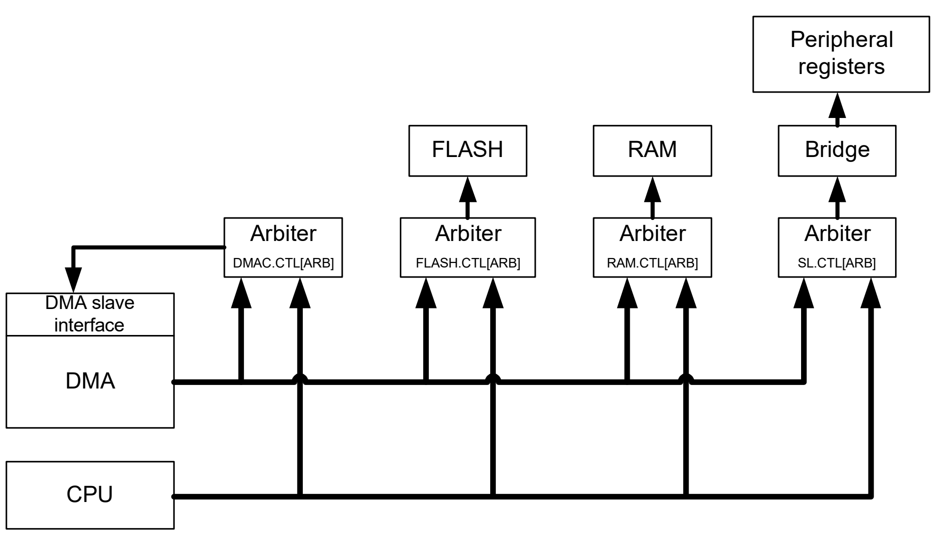
When transferring entire descriptors or chaining descriptor chains, 12 clock cycles are needed for the first data element. Subsequent elements need three cycles. This is also under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to transfer ‘N’ elements is:

No of cycles = (12 + LOAD wait states + STORE wait states) + (N–1)*(3 + LOAD wait states + STORE wait states)

5.4 Arbitration

The AHB bus of the device has two masters: the CPU and the DMA controller. All peripherals and memory connect to the bus through slave interfaces. There are dedicated slave interfaces for flash memory and RAM with their own arbiters. The peripheral registers all connect to a single slave interface through a bridge into a dedicated arbiter. The DMA controller's slave interface, which is used to access the DMA controller's control registers, all connect through another slave interface. Figure 5-12 illustrates this architecture.

Figure 5-12. PSoC 4 Bus Architecture

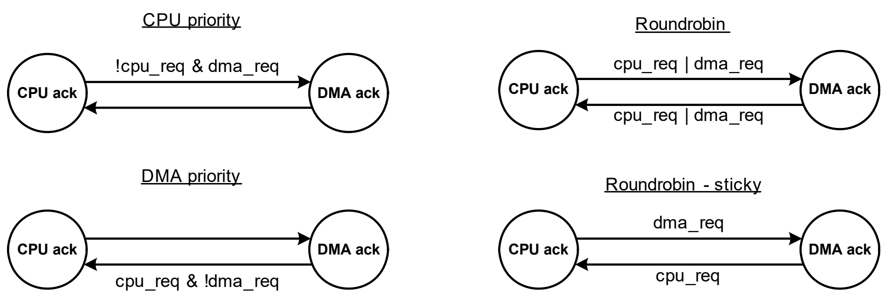


The arbitration policy for each slave can be one of the following:

- CPU priority: CPU always has the priority on arbitration. DMA access is allowed only when there are no CPU requests.
- DMA priority: DMA always has the priority on arbitration. CPU access is allowed only when there are no DMA requests.
- Round-robin: The arbitration priority keeps switching between DMA and CPU for every request. The arbitration priority switches for every request – CPU or DMA.
- Round-robin sticky: This mode is similar to the round robin, but the priority switches only when there has been a request from lower priority master. For example, if the current priority was CPU and there was a request made by the DMA, the priority switches to DMA for the next request. If there was no request from DMA, CPU holds the current priority.

The arbitration models are illustrated using the following diagrams.

Figure 5-13. Arbitration Models



5.5 Register List

Register Name	Comments	Features
DMAC_CTL	Block control	Enable bit for the DMA controller.
DMAC_STATUS	Block status	Provides status information of the DMA controller.
DMAC_STATUS_SRC_ADDR	Current source address	Provides details of the source address currently being loaded.
DMAC_STATUS_DST_ADDR	Current destination address	Provides details of the destination address currently being loaded.
DMAC_STATUS_CH_ACT	Channel activation status	Software reads this field to get information on all actively pending channels (either in pending or in the data transfer engine).
DMAC_CH_CTLx	Channel control register	Provides channel enable, PING/PONG and priority settings for Channel x.
DMAC_DESCRx_PING_SRC	PING source address	Base address of source location for Channel x.
DMAC_DESCRx_PING_DST	PING destination address	Base address of destination location for Channel x.
DMAC_DESCRx_PING_CTL	PING control word	All control settings for the PING descriptor.
DMAC_DESCRx_PING_STATUS	PING status word	Validity, response, and real time Data_NR index status.
DMAC_DESCRx_PONG_SRC	PONG source address	Base address of source location for Channel x.
DMAC_DESCRx_PONG_DST	PONG destination address	Base address of destination location for Channel x.
DMAC_DESCRx_PONG_CTL	PONG control word	All control settings for the PONG descriptor.
DMAC_DESCRx_PONG_STATUS	PONG status word	Validity, response, and real time Data_NR index status.
DMAC_INTR	Interrupt register	
DMAC_INTR_SET	Interrupt set register	When read, this register reflects the interrupt request register.
DMAC_INTR_MASK	Interrupt mask	Mask for corresponding field in INTR register.
DMAC_INTR_MASKED	Interrupt masked register	When read, this register reflects a bit-wise and between the interrupt request and mask registers. This register allows the software to read the status of all mask-enabled interrupt causes with a single load operation, rather than two load operations: one for the interrupt causes and one for the masks. This simplifies firmware development.

6. Interrupts

The ARM Cortex-M0+ (CM0+) CPU in PSoC® Analog Coprocessor supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or interrupt service routine (ISR) being executed by the CPU. The device provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

6.1 Features

PSoC supports the following interrupt features:

- Supports 25 interrupts
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

6.2 How It Works

Figure 6-1. PSoC Analog Coprocessor Interrupts Block Diagram

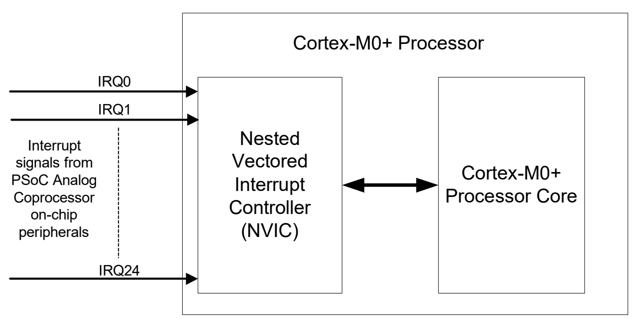


Figure 6-1 shows the interaction between interrupt signals and the Cortex-M0+ CPU. PSoC has 25 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in Figure 6-1 because they are part of CM0+ core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

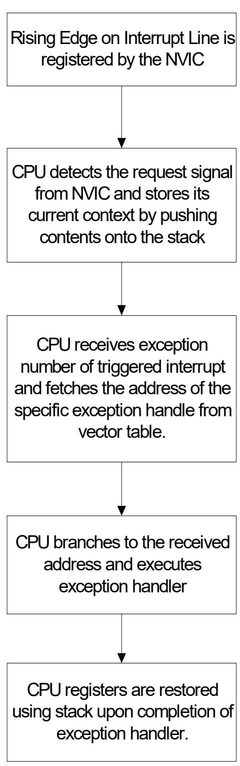
6.3 Interrupts and Exceptions - Operation

6.3.1 Interrupt/Exception Handling

The following sequence of events occurs when an interrupt or exception event is triggered:

1. Assuming that all the interrupt signals are initially low (idle or inactive state) and the processor is executing the main code, a rising edge on any one of the interrupt lines is registered by the NVIC. The interrupt line is now in a pending state waiting to be serviced by the CPU.
2. On detecting the interrupt request signal from the NVIC, the CPU stores its current context by pushing the contents of the CPU registers onto the stack.
3. The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions have a unique exception number, as given in Table 6-1. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table.
4. The CPU then branches to this address and executes the exception handler that follows.
5. Upon completion of the exception handler, the CPU registers are restored to their original state using stack pop operations; the CPU resumes the main code execution.

Figure 6-2. Interrupt Handling When Triggered



When the NVIC receives an interrupt request while another interrupt is being serviced or receives multiple interrupt requests at the same time, it evaluates the priority of all these interrupts, sending the exception number of the highest priority interrupt to the CPU. Thus, a higher priority interrupt can block the execution of a lower priority ISR at any time.

Exceptions are handled in the same way that interrupts are handled. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

6.3.2 Level and Pulse Interrupts

NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IRQ24). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 6-3. Level Interrupts

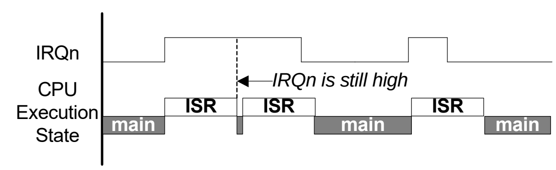


Figure 6-4. Pulse Interrupts

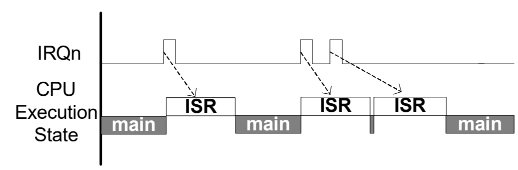


Figure 6-3 and Figure 6-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

1. On a rising edge event of the interrupt signal, the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the interrupt requests have not yet been serviced by the CPU.
2. The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
3. When the ISR is being executed by the CPU, one or more rising edges of the interrupt signal are logged as a single pending request. The pending interrupt is serviced again after the current ISR execution is complete (see Figure 6-4 for pulse interrupts).
4. If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. Figure 6-3 illustrates this for level triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

6.3.3 Exception Vector Table

The exception vector table (Table 6-1), stores the entry point addresses for all exception handlers. The CPU fetches the appropriate address based on the exception number.

Table 6-1. Exception Vector Table

Exception Number	Exception	Exception Priority	Vector Address
–	Initial Stack Pointer Value	Not applicable (NA)	Base_Address - 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM)
1	Reset	–3, the highest priority	Base_Address + 0x04
2	Non Maskable Interrupt (NMI)	–2	Base_Address + 0x08
3	HardFault	–1	Base_Address + 0x0C
4-10	Reserved	NA	Base_Address + 0x10 to Base_Address + 0x28
11	Supervisory Call (SVCall)	Configurable (0 - 3)	Base_Address + 0x2C
12-13	Reserved	NA	Base_Address + 0x30 to Base_Address + 0x34
14	PendSupervisory (PendSV)	Configurable (0 - 3)	Base_Address + 0x38
15	System Timer (SysTick)	Configurable (0 - 3)	Base_Address + 0x3C
16	External Interrupt(IRQ0)	Configurable (0 - 3)	Base_Address + 0x40
…	…	Configurable (0 - 3)	…
40	External Interrupt(IRQ24)	Configurable (0 - 3)	Base_Address + 0xA0

In Table 6-1, the first word (4 bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. In PSoC, the vector table can be configured to be located either in flash memory (base address of 0x00000000) or SRAM (base address of 0x20000000). This configuration is done by writing to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. When the VECT_IN_RAM bit field is ‘1’, CPU fetches exception handler addresses from the SRAM vector table location. When this bit field is ‘0’ (reset state), the vector table in flash memory is used for exception address fetches. You must set the VECT_IN_RAM bit field as part of the device boot code to configure the vector table to be in SRAM. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

Reads of flash addresses 0x00000000 and 0x00000004 are redirected to the first eight bytes of SROM to fetch the stack pointer and reset vectors, unless the NO_RST_OVR bit of the CPUSS_SYSREQ register is set. To allow flash read from addresses 0x00000000 and 0x00000004, the NO_RST_OVR bit should be set to ‘1’. The stack pointer vector holds the address that the stack pointer is loaded with on reset. The reset vector holds the address of the boot sequence. This mapping is done to use the default addresses for the stack pointer and reset vector from SROM when the device reset is released. For reset, boot code in SROM is executed first and then the CPU jumps to address 0x00000004 in flash to execute the handler in flash. The reset exception address in the SRAM vector table is never used.

Also, when the SYSREQ bit of the CPUSS_SYSREQ register is set, reads of flash address 0x00000008 are redirected to SROM to fetch the NMI vector address instead of from flash. Reset CPUSS_SYSREQ to read the flash at address 0x00000008.

The exception sources (exception numbers 1 to 15) are explained in 6.4 Exception Sources. The exceptions marked as Reserved in Table 6-1 are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 40) are explained in 6.5 Interrupt Sources.

6.4 Exception Sources

This section explains the different exception sources listed in Table 6-1 (exception numbers 1 to 15).

6.4.1 Reset Exception

Device reset is treated as an exception in PSoC. It is always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of supervisory read-only memory (SROM). The boot code and other data in SROM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in Table 6-1) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in the SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted.

6.4.2 Non-Maskable Interrupt (NMI) Exception

Non-maskable interrupt (NMI) is the highest priority exception other than reset. It is always enabled with a fixed priority of –2. There are two ways to trigger an NMI exception in the device:

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** An NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0P_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

6.4.3 HardFault Exception

HardFault is an always-enabled exception that occurs because of an error during normal or exception processing. HardFault has a fixed priority of –1, meaning it has higher priority than any exception with configurable priority. HardFault exception is a catch-all exception for different types of fault conditions, which include executing an undefined instruction and accessing an invalid memory addresses. The CM0+ CPU does not provide fault status information to the HardFault exception handler, but it does permit the handler to perform an exception return and continue execution in cases where software has the ability to recover from the fault situation.

6.4.4 Supervisor Call (SVCall) Exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0+ in PSoC uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the Chip Operational Modes chapter on page 87 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in the device. The application developer must define the SVCall exception handler according to the end application requirements.

The priority of a SVCall exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_11[31:30] of the System Handler Priority Register 2 (SHPR2). When the SVC instruction is executed, the SVCall exception enters the pending state and waits to be serviced by the CPU. The SVCALLPENDED bit in the System Handler Control and State Register (SHCSR) can be used to check or modify the pending status of the SVCall exception.

6.4.5 PendSV Exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, CM0P_ICSR. On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSVCLR bit in the Interrupt Control State Register, CM0P_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_14[23:22] of the System Handler Priority Register 3 (CM0P_SHPR3). See the ARMv6-M Architecture Reference Manual for more details.

6.4.6 SysTick Exception

CM0+ CPU in PSoC supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0P_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_15[31:30] of the System Handler Priority Register 3 (SHPR3). The SysTick exception can always be generated in software at any instant by writing a one to the PENDSTSETb bit in the Interrupt Control State Register, CM0P_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a one to the PENDSTCLR bit in the Interrupt Control State Register, CM0P_ICSR.

6.5 Interrupt Sources

PSoC supports 25 interrupts (IRQ0 to IRQ24 or exception numbers 16 – 40) from peripherals. The source of each interrupt is listed in . PSoC provides flexible sourcing options for each interrupt line. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM and serial communication block. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously.

See the I/O System chapter on page 57 for details on GPIO interrupts.

Table 6-2. List of PSoC Analog Coprocessor Interrupt Sources

Interrupt	Cortex-M0+ Exception No.	Interrupt Source
NMI	2	SYS_REQ
IRQ0	16	GPIO Interrupt - Port 0
IRQ1	17	GPIO Interrupt - Port 1
IRQ2	18	GPIO Interrupt - Port 2
IRQ3	19	GPIO Interrupt - Port 3
IRQ4	20	GPIO Interrupt - All Port
IRQ5	21	LPCOMP (low-power comparator)
IRQ6	22	CTB (Continuous Time Block) - All CTBs
IRQ7	23	WDT (Watchdog timer)
IRQ8	24	SCB0 (Serial Communication Block 0)
IRQ9	25	SCB1 (Serial Communication Block 1)
IRQ10	26	SCB2 (Serial Communication Block 2)
IRQ11	27	WCO
IRQ12	28	DMA Interrupt
IRQ13	29	SPCIF Interrupt
IRQ14	30	CSD (CapSense)
IRQ15	31	SAR ADC
IRQ16	32	UAB(Universal Analog Block) -all UABs
IRQ17	33	TCPWM0 (Timer/Counter/PWM 0)
IRQ18	34	TCPWM1 (Timer/Counter/PWM 1)
IRQ19	35	TCPWM2 (Timer/Counter/PWM 2)
IRQ20	36	TCPWM3 (Timer/Counter/PWM 3)
IRQ21	37	TCPWM4 (Timer/Counter/PWM 4)
IRQ22	38	TCPWM5 (Timer/Counter/PWM 5)
IRQ23	39	TCPWM6 (Timer/Counter/PWM 6)
IRQ24	40	TCPWM7 (Timer/Counter/PWM 7)

6.6 Exception Priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. PSoC provides flexibility in choosing priority values for different exceptions. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In PSoC, lower priority numbers represent higher priorities. This means that the Reset, NMI, and HardFault exceptions have the highest priorities. The other exceptions can be assigned a configurable priority level between 0 and 3.

PSoC supports nested exceptions in which a higher priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as active exception. The

CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0+ CPU in PSoC allows nesting of up to four exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1 to 15 are explained in "Exception Sources" on page 49.

The priority of the 25 interrupts (IRQ0 to IRQ24) can be configured by writing to the Interrupt Priority registers (CM0P_IPR). This is a group of 32-bit registers with each register storing the priority values of four interrupts, as given in Table 6-3. The other bit fields in the register are not used.

Table 6-3. Interrupt Priority Register Bit Definitions

Bits	Name	Description
7:6	PRI_N0	Priority of interrupt number N.
15:14	PRI_N1	Priority of interrupt number N+1.
23:22	PRI_N2	Priority of interrupt number N+2.
31:30	PRI_N3	Priority of interrupt number N+3.

6.7 Enabling and Disabling Interrupts

The NVIC provides registers to individually enable and disable the 25 interrupts in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that interrupt line. The Interrupt Set-Enable Register (CM0P_ISER) and the Interrupt Clear-Enable Register (CM0P_ICER) are used to enable and disable the interrupts respectively. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 6-4 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-4. Interrupt Enable/Disable Registers

Register	Operation	Bit Value	Comment
Interrupt Set Enable Register (CM0P_ISER)	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
Interrupt Clear Enable Register (CM0P_ICER)	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The CM0P_ISER and CM0P_ICER registers are applicable only for interrupts IRQ0 to IRQ24. These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in "Exception Sources" on page 49.

The PRIMASK register in Cortex-M0+ (CM0+) CPU can be used as a global exception enable register to mask all the configurable priority exceptions irrespective of whether they are enabled. Configurable priority exceptions include all the exceptions except Reset, NMI, and HardFault listed in Table 6-1. They can be configured to a priority level between 0 and 3, 0 being the highest priority and 3 being the lowest priority. When the PM bit (bit 0) in the PRIMASK register is set, none of the configurable priority exceptions can be serviced by the CPU, though they can be in the pending state waiting to be serviced by the CPU after the PM bit is cleared.

6.8 Exception States

Each exception can be in one of the following states.

Table 6-5. Exception States

Exception State	Meaning
Inactive	The exception is not active or pending. Either the exception is disabled or the enabled exception has not been triggered.
Pending	The exception request is received by the CPU/NVIC and the exception is waiting to be serviced by the CPU.
Active	An exception that is being serviced by the CPU but whose exception handler execution is not yet complete. A high-priority exception can interrupt the execution of lower priority exception. In this case, both the exceptions are in the active state.
Active and Pending	The exception is serviced by the processor and there is a pending request from the same source during its exception handler execution.

The Interrupt Control State Register (CM0P_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0P_ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0P_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0P_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ24) is in a pending state.

6.8.1 Pending Exceptions

When a peripheral generates an interrupt request signal to the NVIC or an exception event occurs, the corresponding exception enters the pending state. When the CPU starts executing the corresponding exception handler routine, the exception is changed from the pending state to the active state.

The NVIC allows software pending of the 25 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending register (CM0P_ISPR) and the Interrupt Clear-Pending register (CM0P_ICPR) are used to set and clear the pending status of the interrupt lines. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt.

Table 6-6 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-6. Interrupt Set Pending/Clear Pending Registers

Register	Operation	Bit Value	Comment
Interrupt Set-Pending Register (CM0P_ISPR)	Write	1	To put an interrupt to pending state
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending
Interrupt Clear-Pending Register (CM0P_ICPR)	Write	1	To clear a pending interrupt
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending

Setting the pending bit when the same bit is already set results in only one execution of the ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0P_ISER register.

Note that the CM0P_ISPR and CM0P_ICPR registers are used only for the 25 peripheral interrupts (exception numbers 16–40). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in “Exception Sources” on page 49.

6.9 Stack Usage for Exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its general-purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the Program and Status Register (PSR), ReturnAddress, Link Register (LR or R14), R12, R3, R2, R1, and R0. Cortex-M0+ has two stack pointers - MSP and PSP. Only one of the stack pointers can be active at a time. When in thread mode, the Active Stack Pointer bit in the Control register is used to define the current active stack pointer. When in handler mode, the MSP is always used as the stack pointer. The stack pointer in Cortex-M0+ always grows downwards and points to the address that has the last pushed data.

When the CPU is in thread mode and an exception request comes, the CPU uses the stack pointer defined in the control register to store the general-purpose register contents. After the stack push operations, the CPU enters handler mode to execute the exception handler. When another higher priority exception occurs while executing the current exception, the MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the Cortex-M0+ CPU chapter on page 28 for details.

The Cortex-M0+ uses two techniques, tail chaining and late arrival, to reduce latency in servicing exceptions. These techniques are not visible to the external user and are part of the internal processor architecture. For information on tail chaining and late arrival mechanism, visit the ARM Infocenter.

6.10 Interrupts and Low-Power Modes

PSoC allows device wakeup from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the device to enter Active mode when one or more wakeup sources generate an interrupt signal. After entering Active mode, the ISR of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0+ CPU, triggers the transition into Sleep and Deep-Sleep modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 88. Chip low-power modes have two categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available only in the Active and Deep-Sleep modes (watchdog timer interrupt,)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

6.11 Exceptions – Initialization and Configuration

This section covers the different steps involved in initializing and configuring exceptions in PSoC.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required – either in flash memory or SRAM. This configuration is done by writing either a ‘1’ (SRAM vector table) or ‘0’ (flash vector table) to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. This register write is done as part of device initialization code.

   It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents. PSoC Creator IDE uses the vector table in SRAM by default.

2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   b. Define the exception handler function and write the address of the function to the exception vector table. Table 6-1 gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
   c. Set up the exception priority, as explained in “Exception Priority” on page 51.
   d. Enable the exception, as explained in “Enabling and Disabling Interrupts” on page 52.

6.12 Registers

Table 6-7. List of Registers

Register Name	Description
CM0P_ISER	Interrupt Set-Enable Register
CM0P_ICER	Interrupt Clear Enable Register
CM0P_ISPR	Interrupt Set-Pending Register
CM0P_ICPR	Interrupt Clear-Pending Register
CM0P_IPR	Interrupt Priority Registers
CM0P_ICSR	Interrupt Control State Register
CM0P_AIRCR	Application Interrupt and Reset Control Register
CM0P_SCR	System Control Register
CM0P_CCR	Configuration and Control Register
CM0P_SHPR2	System Handler Priority Register 2
CM0P_SHPR3	System Handler Priority Register 3
CM0P_SHCSR	System Handler Control and State Register
CM0P_SYST_CSR	Systick Control and Status Register
CPUSS_CONFIG	CPU Subsystem Configuration Register
CPUSS_SYSREQ	System Request Register

6.13 Associated Documents

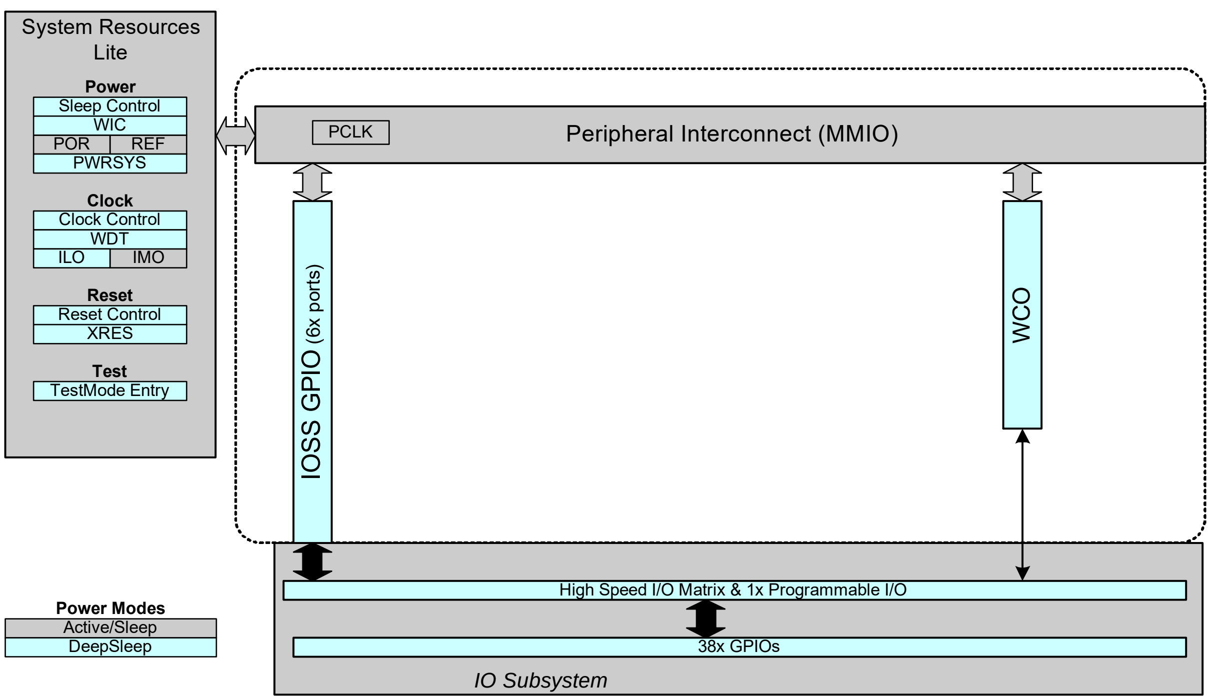
- ARMv6-M Architecture Reference Manual – This document explains the ARM Cortex-M0+ architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

Section C: System Resources Subsystem (SRSS)

This section encompasses the following chapters:

Top Level Architecture

System-Wide Resources Block Diagram



7. I/O System

This chapter explains the PSoC® Analog Coprocessor I/O system, its features, architecture, operating modes, and interrupts. The GPIO pins in the PSoC Analog Coprocessor are grouped into ports; a port can have a maximum of eight GPIOs. The PSoC Analog Coprocessor has a maximum of 38 GPIOs arranged in six ports.

7.1 Features

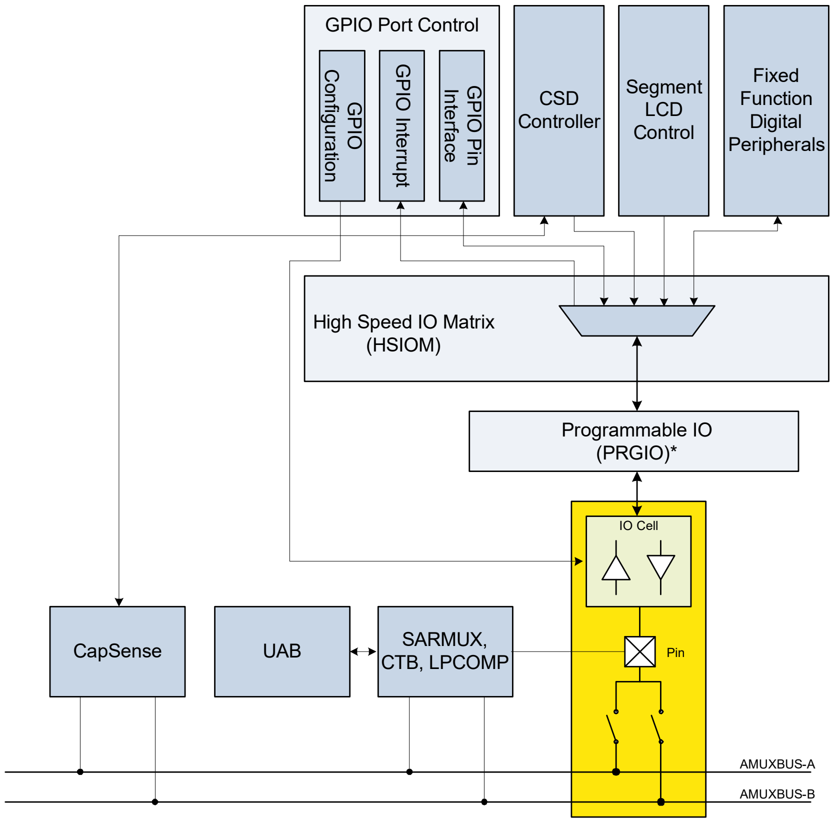
The PSoC Analog Coprocessor GPIOs have these features:

- Analog and digital input and output capabilities
- Eight drive strength modes
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Hold mode for latching previous state (used for retaining I/O state in Deep-Sleep mode)
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Smart I/O block provides the ability to perform Boolean functions in the I/O signal path
- CapSense support
- Segment LCD drive support
- Two analog mux buses (AMUXBUS-A and AMUXBUS-B) that can be used to multiplex analog signals

7.2 GPIO Interface Overview

PSoC Analog Coprocessor is equipped with analog and digital peripherals. Figure 7-1 shows an overview of the routing between the peripherals and pins.

Figure 7-1. GPIO Interface Overview



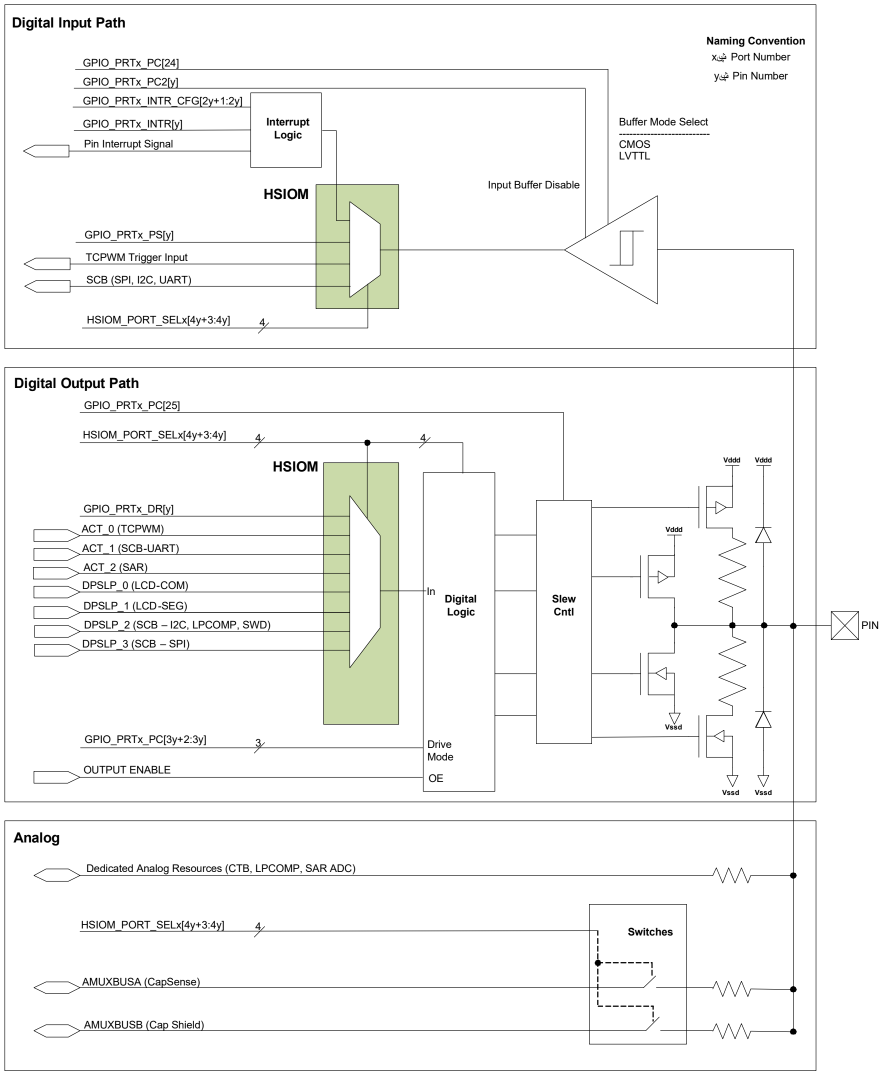
* not available on all the GPIOs

GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high input impedance and a driver for the digital output signals. The digital peripherals connect to the I/O cells via the high-speed I/O matrix (HSIOM). HSIOM contains multiplexers to connect between a peripheral selected by the user and the pin. Some port pins have a Smart I/O block between the HSIOM and the pins. The Smart I/O block enables logical operations on the pin signal. The analog peripheral and analog mux bus connections are done in the GPIO cell directly. The CapSense block is connected to the GPIO pins through the AMUX buses.

7.3 I/O Cell Architecture

Figure 7-2 shows the I/O cell architecture. It comprises of an input buffer and an output driver. This architecture is present in every GPIO cell. It connects to the HSIOM multiplexers/Smart I/O block for the digital input and the output signal.

Figure 7-2. GPIO Block Diagram



7.3.1 Digital Input Buffer

The digital input buffer provides a high-impedance buffer for the external digital input. The buffer is enabled and disabled by the INP_DIS bit of the Port Configuration Register 2 (GPIO_PRTx_PC2, where x is the port number). The buffer is configurable for the following modes:

- CMOS
- LVTTL

These buffer modes are selected by the PORT_VTRIP_SEL bit (GPIO_PRTx_PC[24])of the Port Configuration register.

Table 7-1. Input Buffer Modes

PORT_VTRIP_SEL	Input Buffer Mode
0b	CMOS
1b	LVTTL

The threshold values for each mode can be obtained from the device datasheet. The output of the input buffer is connected to the HSIOM for routing to the selected peripherals. Writing to the HSIOM port select register (HSIOM_PORT_SELx) selects the peripheral. The digital input peripherals in the HSIOM, shown in Figure 7-2, are pin dependent. See the device datasheet to know the functions available for each pin.

7.3.2 Digital Output Driver

Pins are driven by the digital output driver. It consists of circuitry to implement different drive modes and slew rate control for the digital output signals. The peripheral connects to the digital output driver through the HSIOM; a particular peripheral is selected by writing to the HSIOM port select register (HSIOM_PORT_SELx).

In the PSoC Analog Coprocessor I/Os are driven with V_{DDD} supply. Each GPIO pin has ESD diodes to clamp the pin voltage to the V_{DDD} source. Ensure that the voltage at the pin does not exceed the I/O supply voltage V_{DDD} and drop below V_{SSD}. For the absolute maximum and minimum GPIO voltage, see the device datasheet. The digital output driver can be enabled and disabled using the DSI signal from the peripheral or data register (GPIO_PRTx_DR) associated with the output pin. See 7.4 High-Speed I/O Matrix to know about the peripheral source selection for the data and to enable or disable control source selection.

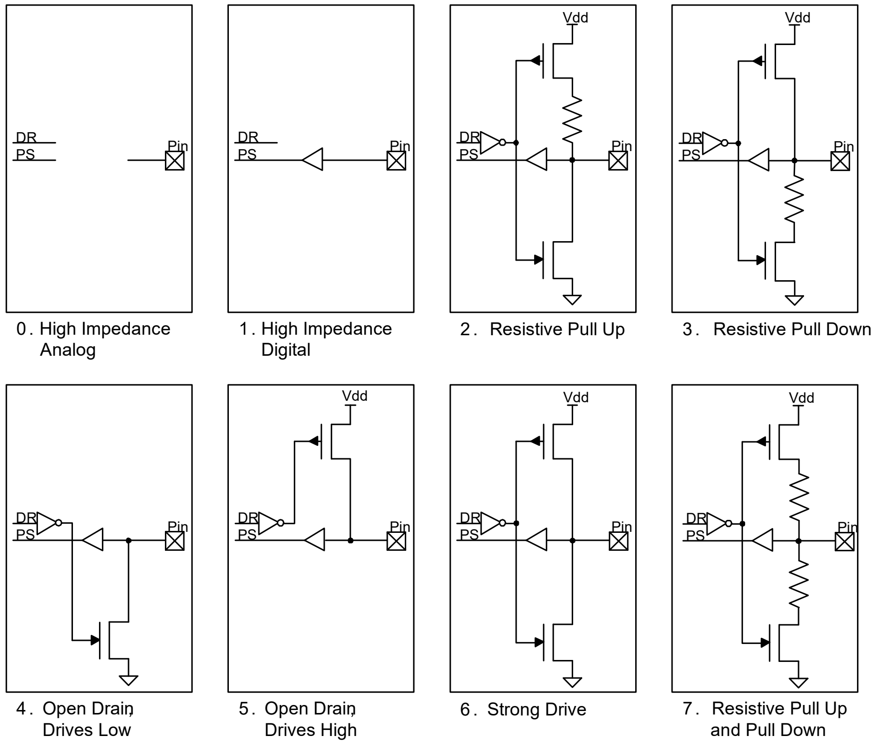
7.3.2.1 Drive Modes

Each I/O is individually configurable into one of eight drive modes using the Port Configuration register, GPIO_PRTx_PC. Table 7-2 lists the drive modes. Figure 7-2 is a simplified output driver diagram that shows the pin view based on each of the eight drive modes.

Table 7-2. Drive Mode Settings

GPIO_PRTx_PC ('x' denotes port number and 'y' denotes pin number)				
Bits	**Drive Mode**	**Value**	**Data = 1**	**Data = 0**
3y+2: 3y	SEL'y'	Selects Drive Mode for Pin 'y' (0 ≤ y ≤ 7)		
	High-Impedance Analog	0	High Z	High Z
	High-impedance Digital	1	High Z	High Z
	Resistive Pull Up	2	Weak 1	Strong 0
	Resistive Pull Down	3	Strong 1	Weak 0
	Open Drain, Drives Low	4	High Z	Strong 0
	Open Drain, Drives High	5	Strong 1	High Z
	Strong Drive	6	Strong 1	Strong 0
	Resistive Pull Up and Down	7	Weak 1	Weak 0

Figure 7-3. I/O Drive Mode Block Diagram



- High-Impedance Analog

High-impedance analog mode is the default reset state; both output driver and digital input buffer are turned off. This state prevents an external voltage from causing a current to flow into the digital input buffer. This drive mode is recommended for pins that are floating or that support an analog voltage. High-impedance analog pins cannot be used for digital inputs. Reading the pin state register returns a 0x00 regardless of the data register value. To achieve the lowest device current in low-power modes, unused GPIOs must be configured to the high-impedance analog mode.

- High-Impedance Digital

High-impedance digital mode is the standard high-impedance (High Z) state recommended for digital inputs. In this state, the input buffer is enabled for digital input signals.

- Resistive Pull-Up or Resistive Pull-Down

Resistive modes provide a series resistance in one of the data states and strong drive in the other. Pins can be used for either digital input or digital output in these modes. If resistive pull-up is required, a ‘1’ must be written to that pin’s Data Register bit. If resistive pull-down is required, a ‘0’ must be written to that pin’s Data Register. Interfacing mechanical switches is a common application of these drive modes. The resistive modes are also used to interface PSoC with open drain drive lines. Resistive pull-up is used when input is open drain low and resistive pull-down is used when input is open drain high.

- Open Drain Drives High and Open Drain Drives Low

Open drain modes provide high impedance in one of the data states and strong drive in the other. The pins can be used as digital input or output in these modes. Therefore, these modes are widely used in bi-directional digital communication. Open drain drive high mode is used when signal is externally pulled down and open drain drive low is used when signal is externally pulled high. A common application for open drain drives low mode is driving I^2C bus signal lines.

- Strong Drive

The strong drive mode is the standard digital output mode for pins; it provides a strong CMOS output drive in both high and low states. Strong drive mode pins must not be used as inputs under normal circumstances. This mode is often used for digital output signals or to drive external transistors.

- Resistive Pull-Up and Resistive Pull-Down

In the resistive pull-up and resistive pull-down mode, the GPIO will have a series resistance in both logic 1 and logic 0 output states. The high data state is pulled up while the low data state is pulled down. This mode is used when the bus is driven by other signals that may cause shorts.

7.3.2.2 Slew Rate Control

GPIO pins have fast and slow output slew rate options in strong drive mode; this is configured using PORT_SLOW bit of the Port Configuration register (GPIO_PRTx_PC[25]). Slew rate is individually configurable for each port. This bit is cleared by default and the port works in fast slew mode. This bit can be set if a slow slew rate is required. Slower slew rate results in reduced EMI and crosstalk; hence, the slow option is recommended for low-frequency signals or signals without strict timing constraints.

7.4 High-Speed I/O Matrix

The high-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the peripherals inside the device. As the GPIOs are shared for multiple functions, HSIOM multiplexes the pin and connects to a particular peripheral selected by the user. Note that in PSoC Analog Coprocessor, the Smart I/O block bridges the Port 0 pins to the HSIOM. Other ports connect directly to the HSIOM.The HSIOM_PORT_SELx register is provided to select the peripheral. It is a 32-bit wide register available for each port, with each pin occupying four bits. This register provides up to 16 different options for a pin as listed in Table 7-3.

Table 7-3. PSoC Analog Coprocessor HSIOM Port Settings

HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number)			
Bits	**Name (SEL'y')**	**Value**	**Description (Selects pin 'y' source ($0 \le y \le 7$))**
4y+3 : 4y	DR	0	Pin is regular firmware-controlled I/O or connected to dedicated hardware block.
	CSD_SENSE	4	Pin is a CSD sense pin (analog mode).
	CSD_SHIELD	5	Pin is a CSD shield pin (analog mode).
	AMUXA	6	Pin is connected to AMUXBUS-A.
	AMUXB	7	Pin is connected to AMUXBUS-B. This mode is also used for CSD I/O charging. When CSD I/O charging is enabled in CSD_CONTROL, the digital I/O driver is connected to csd_charge signal (the pin is still connected to AMUXBUS-B).
	ACTIVE_0	8	Pin-specific Active source #0 (TCPWM Output).
	ACTIVE_1	9	Pin-specific Active source #1 (SCB-UART).
	ACTIVE_2	10	Pin-specific Active source #2 (UAB).
	ACTIVE_3	11	Pin-specific Active source #3 (TCPWM Input).
	DEEP_SLEEP_0	12	Pin-specific Deep-Sleep source #0 (LCD - COM).
	DEEP_SLEEP_1	13	Pin-specific Deep-Sleep source #1 (LCD - SEG).
	DEEP_SLEEP_2	14	Pin-specific Deep-Sleep source #2 (SCB-I^2C, SWD, LPCOMP).
	DEEP_SLEEP_3	15	Pin-specific Deep-Sleep source #3 (SCB-SPI).

Note The Active and Deep-Sleep sources are pin dependent. See the “Pinouts” section of the device datasheet for more details on the features supported by each pin.

7.5 Smart I/O

The Smart I/O block adds programmable logic to an I/O port. This programmable logic integrates board-level Boolean logic functionality such as AND, OR, and XOR into the port. The Smart I/O block has these features:

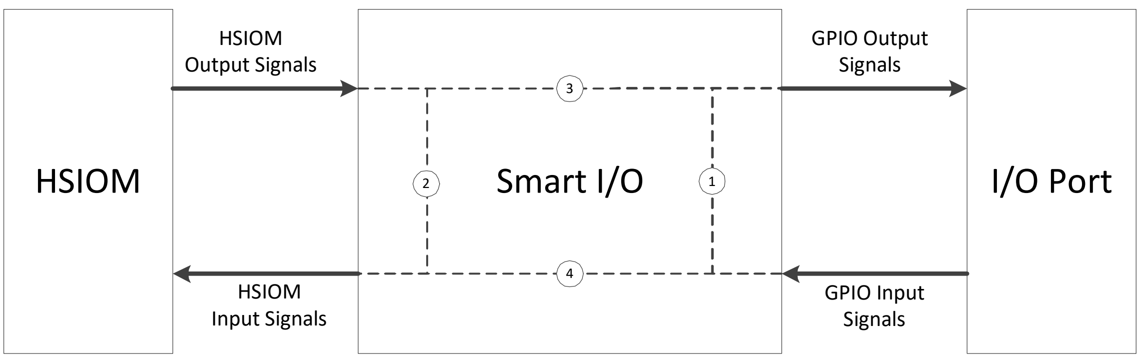
- Integrate board-level Boolean logic functionality into a port
- Ability to preprocess HSIOM input signals from the GPIO port pins
- Ability to post-process HSIOM output signals to the GPIO port pins
- Support in all device power modes
- Integrate closely to the I/O pads, providing shortest signal paths with programmability

PSoC Analog Coprocessor supports SmartIO on Port 0.

7.5.1 Overview

The Smart I/O block is positioned in the signal path between the HSIOM and the I/O port. The HSIOM multiplexes the output signals from fixed-function peripherals and CPU to a specific port pin and vice-versa. The Smart I/O block is placed on this signal path, acting as a bridge that can process signals from port pins and HSIOM, as shown in Figure 7-4.

Figure 7-4. Smart I/O Interface



The signal paths supported through the Smart I/O block as shown in Figure 7-4 are as follows:

1. Implement self-contained logic functions that directly operate on port I/O signals
2. Implement self-contained logic functions that operate on HSIOM signals or a combination of both
3. Operate on and modify HSIOM output signals and route the modified signals to port I/O signals
4. Operate on and modify port I/O signals and route the modified signals to HSIOM input signals

The following sections discuss the Smart I/O block components, routing, and configuration in detail. In these sections, GPIO signals (io_data) refer to the input/output signals from the I/O port; device or chip (chip_data) signals refer to the input/output signals from HSIOM.

7.5.2 Block Components

The internal logic of the Smart I/O includes these components:

- Clock/reset component
- Synchronizers
- LUT3 components
- Data unit component

7.5.2.1 Clock and Reset

The clock and reset component selects the Smart I/O block's clock (clk_block) and reset signal (rst_block_n). A single clock and reset signal is used for all components in the block. The clock and reset sources are determined by the CLOCK_SRC[4:0] bit field of the PRGIO_PRTx_CTL register. The selected clock is used for the synchronous logic in the block components, which includes the I/O input synchronizers, LUT, and data unit components. The selected reset is used to asynchronously reset the synchronous logic in the LUT and data unit components.

Note that the selected clock (clk_block) for the block's synchronous logic is not phase-aligned with other synchronous logic in the device, operating on the same clock. Therefore, communication between Smart I/O and other synchronous logic should be treated as asynchronous.

The following clock sources are available for selection:

- GPIO input signals "io_data_in[7:0]". These clock sources have no associated reset.
- HSIOM output signals "chip_data[7:0]". These clock sources have no associated reset.

- The Smart I/O clock (clk_prgio). This is derived from the system clock (clk_sys) using a peripheral clock divider. See the Clocking System chapter on page 76 for details on peripheral clock dividers. This clock is only available in Active and Sleep power modes. The clock can have one out of two associated resets: rst_sys_act_n and rst_sys_dpslp_n. These resets determine in which system power modes the block synchronous state is reset; for example, rst_sys_act_n is intended for Smart I/O synchronous functionality in the Active power mode and reset is activated in the Deep-Sleep power mode.
- The low-frequency (40 kHz) system clock (clk_lf). This clock is available in Deep-Sleep power mode. This clock has an associated reset, rst_lf_dpslp_n.

When the block is enabled, the selected clock (clk_block) and associated reset (rst_block_n) are released to the fabric components. When the fabric is disabled, no clock is released to the fabric components and the reset is activated (the LUT and data unit components are set to the reset value of ‘0’).

The I/O input synchronizers introduce a delay of two clk_block cycles (when synchronizers are enabled). As a result, in the first two cycles, the block may be exposed to stale data from the synchronizer output. Hence, during the first two clock cycles, the reset is activated and the block is in bypass mode.

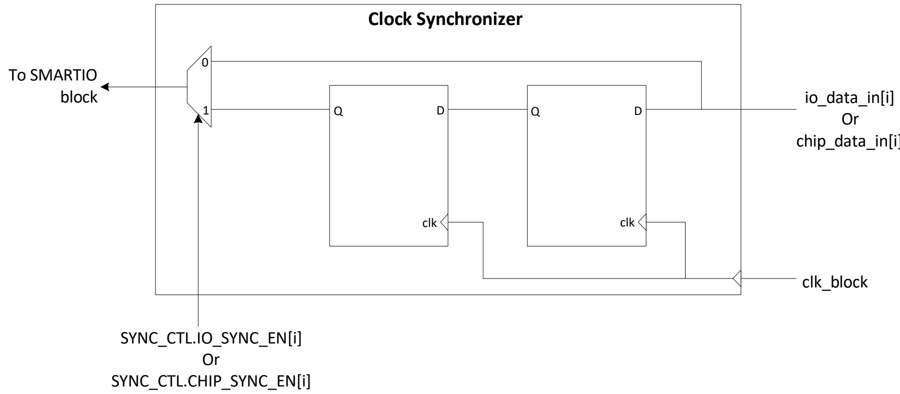
Table 7-4. Clock and Reset Register Control

Register[BIT_POS]	Bit Name	Description
PRGIO_PRT0_CTL[12:8]	CLK_SRC[4:0]	Clock (clk_block)/reset (rst_block_n) source selection: "0": io_data[0]/'1' ... "7": io_data[7]/'1' "8": chip_data[0]/'1' ... "15": chip_data[7]/'1' "16": clk_prgio/rst_sys_act_n; asserts reset in any power mode other than Active; that is, Smart I/O is active only in Active power mode with clock from the peripheral divider. "17": clk_prgio/rst_sys_dpslp_n. Smart I/O is active in all power modes with clock from the peripheral divider. However, the clock will not be active in Deep-Sleep power mode. "19": clk_lf/rst_lf_dpslp_n. Smart I/O is active in all power modes with clock from ILO. "20"-"30": Clock source is a constant '0'. Any of these clock sources should be selected when the IP is disabled to ensure low power consumption. "31": clk_sys/'1'. This selection is NOT intended for "clk_sys" operation. However, for asynchronous operation, three "clk_sys" cycles after enabling the IP, the IP is fully functional (reset is de-activated). To be used for asynchronous (clockless) block functionality.

7.5.2.2 Synchronizer

Each GPIO input signal and device input signal (HSIOM input) can be used either asynchronously or synchronously. To use the signals synchronously, a double flip-flop synchronizer, as shown in Figure 7-5, is placed on both these signal paths to synchronize the signal to the Smart I/O clock (clk_block). The synchronization for each pin/input is enabled or disabled by setting or clearing the IO_SYNC_EN[i] bit field for GPIO input signal and CHIP_SYNC_EN[i] for HSIOM signal in the PRGIO_PRT0_SYNC_CTL register, where ‘i’ is the pin number.

Figure 7-5. Smart I/O Clock Synchronizer



7.5.2.3 LUT3

Each Smart I/O block contains eight lookup table (LUT3) components. The LUT3 component consists of a three-input LUT and a flip-flop. Each LUT3 block takes three input signals and generates an output based on the configuration set in the PRGIO_PRTx_LUT_CTLy register (y denotes the LUT3 number). For each LUT3, the configuration is determined by an 8-bit lookup vector LUT[7:0] and a 2-bit opcode OPC[1:0] in the PRGIO_PRTx_LUT_CTLy register. The 8-bit vector is used as a lookup table for the three input signals. The 2-bit opcode determines the usage of the flip-flop. The LUT3 configuration for different opcode is shown in Figure 7-6.

PRGIO_PRTx_LUT_SELy registers select the three input signals (tr0_in, tr1_in and tr2_in) going into each LUT3. The input can come from the following sources:

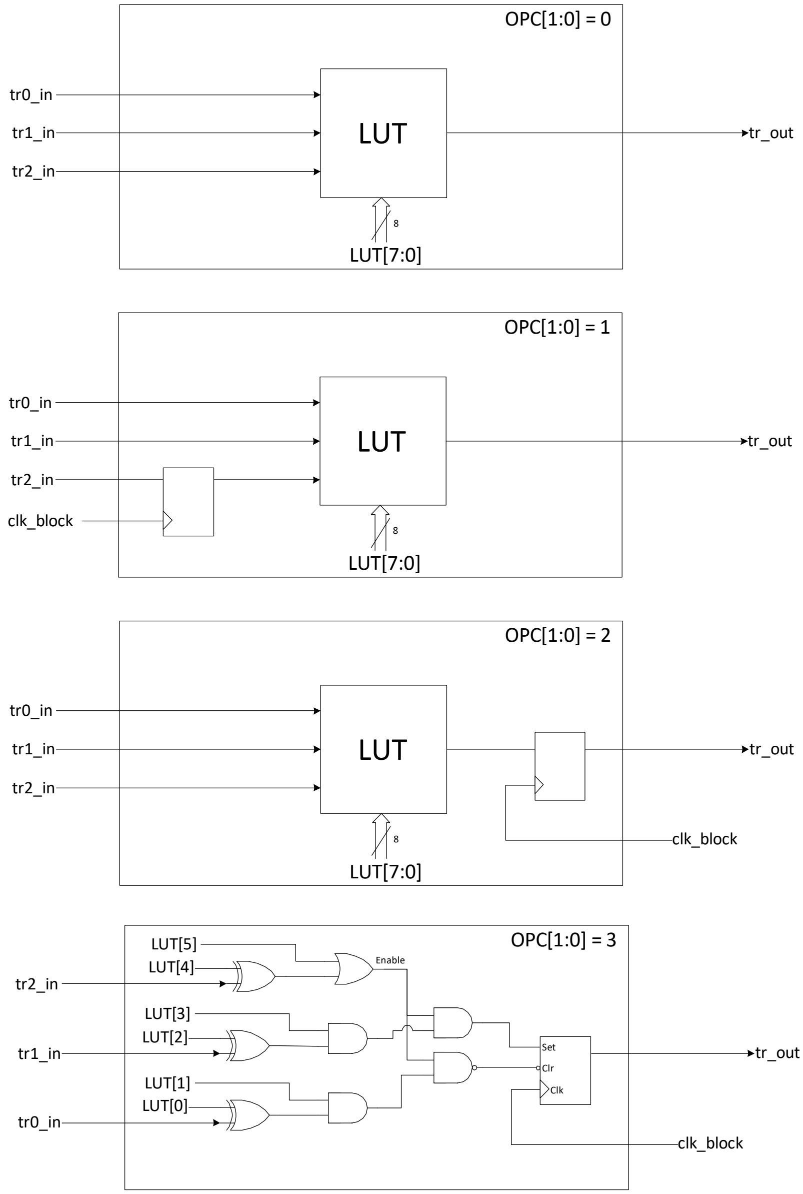
- Data unit output
- Other LUT3 output signals (tr_out)
- HSIOM output signals (chip_data[7:0])
- GPIO input signals (io_data[7:0])

LUT_TR0_SEL[3:0] bits of the PRGIO_PRTx_LUT_SELy register selects the tr0_in signal for the y^{th} LUT3. Similarly, LUT_TR1_SEL[3:0] bits and LUT_TR2_SEL[3:0] bits select the tr1_in and tr2_in signals respectively. See Table 7-5 for details.

Table 7-5. LUT3 Register Control

Register[BIT_POS]	Bit Name	Description
PRGIO_PRTx_LUT_CTLy[7:0]	LUT[7:0]	LUT configuration. Depending on the LUT opcode (LUT_OPC), the internal state, and the LUT input signals tr0_in, tr1_in, and tr2_in, the LUT configuration is used to determine the LUT output signal and the next sequential state.
PRGIO_PRTx_LUT_CTLy[9:8]	LUT_OPC[1:0]	LUT opcode specifies the LUT operation as illustrated in Figure 7-6.
PRGIO_PRTx_LUT_SELy[3:0]	LUT_TR0_SEL[3:0]	LUT input signal "tr0_in" source selection: "0": Data unit output "1": LUT 1 output "2": LUT 2 output "3": LUT 3 output "4": LUT 4 output "5": LUT 5 output "6": LUT 6 output "7": LUT 7 output "8": chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7) "9": chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7) "10": chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7) "11": chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7) "12": io_data[0] (for LUTs 0, 1, 2, 3); io_data[4] (for LUTs 4, 5, 6, 7) "13": io_data[1] (for LUTs 0, 1, 2, 3); io_data[5] (for LUTs 4, 5, 6, 7) "14": io_data[2] (for LUTs 0, 1, 2, 3); io_data[6] (for LUTs 4, 5, 6, 7) "15": io_data[3] (for LUTs 0, 1, 2, 3); io_data[7] (for LUTs 4, 5, 6, 7)
PRGIO_PRTx_LUT_SELy[11:8]	LUT_TR1_SEL[3:0]	LUT input signal "tr1_in" source selection: "0": LUT 0 output "1": LUT 1 output "2": LUT 2 output "3": LUT 3 output "4": LUT 4 output "5": LUT 5 output "6": LUT 6 output "7": LUT 7 output "8": chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7) "9": chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7) "10": chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7) "11": chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7) "12": io_data[0] (for LUTs 0, 1, 2, 3); io_data[4] (for LUTs 4, 5, 6, 7) "13": io_data[1] (for LUTs 0, 1, 2, 3); io_data[5] (for LUTs 4, 5, 6, 7) "14": io_data[2] (for LUTs 0, 1, 2, 3); io_data[6] (for LUTs 4, 5, 6, 7) "15": io_data[3] (for LUTs 0, 1, 2, 3); io_data[7] (for LUTs 4, 5, 6, 7)
PRGIO_PRTx_LUT_SELy[19:16]	LUT_TR2_SEL[3:0]	LUT input signal "tr2_in" source selection. Encoding is the same as for LUT_TR1_SEL.

Figure 7-6. Smart I/O LUT3 Configuration



7.5.2.4 Data Unit

Each Smart I/O block includes a data unit (DU) component. The data unit consists of a simple 8-bit datapath. It is capable of performing simple increment, decrement, increment/decrement, shift, and AND/OR operations. The operation performed by the DU is selected using a 4-bit opcode DU_OPC[3:0] bit field in the PRGIO_PRTx_DU_CTL register.

The data unit component supports up to three input trigger signals (tr0_in, tr1_in, tr2_in) similar to the LUT3 component. These signals are used to initiate an operation defined by the DU opcode. In addition, the data unit also includes two 8-bit input data (data0_in[7:0] and data1_in[7:0]) that are used to initialize the 8-bit internal state (data[7:0]) or to provide a reference. The input to these 8-bit data can come from these sources:

- Constant ‘0x00’
- io_data_in[7:0]
- chip_data_in[7:0]
- DATA[7:0] bit field of PRGIO_PRTx_DATA register

The trigger signals are selected using the DU_TRx_SEL[3:0] bit field of the PRGIO_PRTx_DU_SEL register. The DUT_DATAx_SEL[1:0] bits of the PRGIO_PRTx_DU_SEL register selects the 8-bit input data source. The size of the DU (number of bits used by the datapath) is defined by the DU_SIZE[2:0] bits of the PRGIO_PRTx_DU_CTL register. See Table 7-6 for register control details.

Table 7-6. Data Unit Register Control

Register[BIT_POS]	Bit Name	Description
PRGIO_PRTx_DU_CTL[2:0]	DU_SIZE[2:0]	Size/width of the data unit (in bits) is DU_SIZE+1. For example, if DU_SIZE is 7, the width is 8 bits.
PRGIO_PRTx_DU_CTL[11:8]	DU_OPC[3:0]	Data unit opcode specifies the data unit operation: "1": INCR "2": DECR "3": INCR_WRAP "4": DECR_WRAP "5": INCR_DECR "6": INCR_DECR_WRAP "7": ROR "8": SHR "9": AND_OR "10": SHR_MAJ3 "11": SHR_EQL Otherwise: Undefined.
PRGIO_PRTx_DU_SEL[3:0]	DU_TR0_SEl[3:0]	Data unit input signal "tr0_in" source selection: "0": Constant '0'. "1": Constant '1'. "2": Data unit output. "10-3": LUT 7 - 0 outputs. Otherwise: Undefined.
PRGIO_PRTx_DU_SEL[11:8]	DU_TR1_SEl[3:0]	Data unit input signal "tr1_in" source selection. Encoding same as DU_TR0_SEL
PRGIO_PRTx_DU_SEL[19:16]	DU_TR2_SEl[3:0]	Data unit input signal "tr2_in" source selection. Encoding same as DU_TR0_SEL
PRGIO_PRTx_DU_SEL[25:24]	DU_DATA0_SEL[1:0]	Data unit input data "data0_in" source selection: "0": 0x00 "1": chip_data[7:0]. "2": io_data[7:0]. "3": PRGIO_PRTx_DATA.DATA[7:0] register field.
PRGIO_PRTx_DU_SEL[29:28]	DU_DATA1_SEL[1:0]	Data unit input data "data1_in" source selection. Encoding same as DU_DATA0_SEL.
PRGIO_PRTx_DATA[7:0]	DATA[7:0]	Data unit input data source.

The data unit generates a single output trigger signal (“tr_out”). The internal state (du_data[7:0]) is captured in flip-flops and requires clk_block.

The following pseudo code describes the various datapath operations supported by the DU opcode. Note that “Comb” describes the combinatorial functionality – that is, functionalities that operate independent of previous output states. “Reg” describes the registered functionality – that is, functionalities that operate on inputs and previous output states (registered using flip-flops).

```
// The following is shared by all operations.
mask = (2 ^ (DU_SIZE+1) - 1)
data_eql_data1_in = (data & mask) == (data1_in & mask));
data_eql_0        = (data & mask) == 0);
data_incr         = (data + 1) & mask;
data_decr         = (data - 1) & mask;
data0_masked      = data_in0 & mask;
```

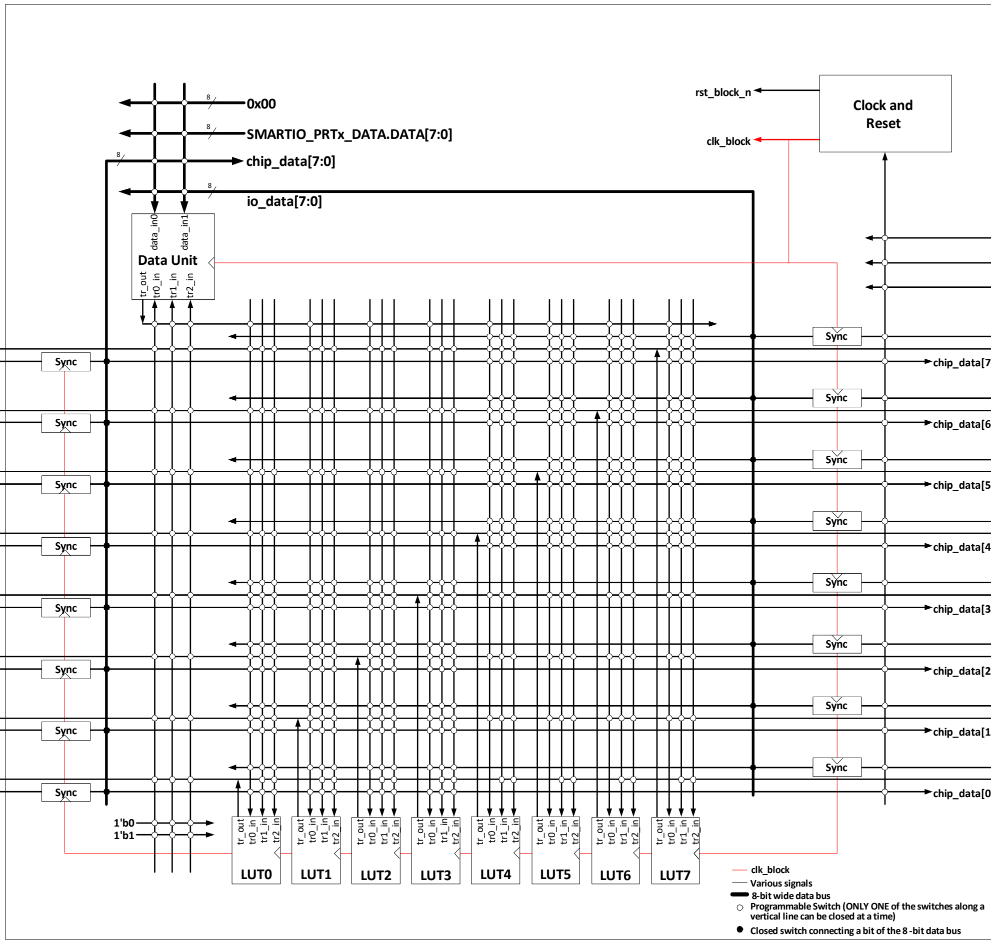
```
// INCR operation: increments data by 1 from an initial value (data0) until it reaches a
// final value (data1).
Comb:tr_out = data_eql_data1_in;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked; //tr0_in is reload signal - loads masked data0
                                             // into data
      else if (tr1_in) data <= data_eql_data1_in ? data : data_incr; //increment data until
                                                                     // it equals data1

// INCR_WRAP operation: operates similar to INCR but instead of stopping at data1, it wraps
// around to data0.
Comb:tr_out = data_eql_data1_in;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked;
      else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;

// DECR operation: decrements data from an initial value (data0) until it reaches 0.
Comb:tr_out = data_eql_0;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked;
      else if (tr1_in) data <= data_eql_0        ? data : data_decr;

// DECR_WRAP operation: works similar to DECR. Instead of stopping at 0, it wraps around to
// data0.
Comb:tr_out = data_eql_0;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked;
      else if (tr1_in) data <= data_eql_0        ? data0_masked: data_decr;

// INCR_DECR operation: combination of INCR and DECR. Depending on trigger signals it either
// starts incrementing or decrementing. Increment stops at data1 and decrement stops at 0.
Comb:tr_out = data_eql_data1_in | data_eql_0;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked; // Increment operation takes precedence over
                                             // decrement when both signal are available
      else if (tr1_in) data <= data_eql_data1_in ? data : data_incr;
      else if (tr2_in) data <= data_eql_0  ? data : data_decr;

// INCR_DECR_WRAP operation: same functionality as INCR_DECR with wrap around to data0 on
// touching the limits.
Comb:tr_out = data_eql_data1_in | data_eql_0;
Reg:  data <= data;
      if (tr0_in)      data <= data0_masked;
      else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;
      else if (tr2_in) data <= data_eql_0  ? data0_masked : data_decr;

// ROR operation: rotates data right and LSB is sent out. The data for rotation is taken from
// data0.
Comb:tr_out = data[0];
Reg:  data <= data;
      if (tr0_in)      data          <= data0_masked;
      else if (tr1_in) {
                       data          <= {0, data[7:1]} & mask; //Shift right operation
                       data[du_size] <= data[0]; //Move the data[0] (LSB) to MSB
      }
```

```
// SHR operation: performs shift register operation. Initial data (data0) is shifted out and
// data on tr2_in is shifted in.
Comb:tr_out = data[0];
Reg:  data <= data;
      if (tr0_in)      data          <= data0_masked;
      else if (tr1_in) {
                       data          <= {0, data[7:1]} & mask; //Shift right operation
                       data[du_size] <= tr2_in; //tr2_in Shift in operation
      }

// SHR_MAJ3 operation: performs the same functionality as SHR. Instead of sending out the
// shifted out value, it sends out a '1' if in the last three samples/shifted-out values
// (data[0]), the signal high in at least two samples. otherwise, sends a '0'. This function
// sends out the majority of the last three samples.
Comb:tr_out =   (data == 0x03)
               | (data == 0x05)
               | (data == 0x06)
               | (data == 0x07);
Reg:  data <= data;
      if (tr0_in)      data          <= data0_masked;
      else if (tr1_in) {
                       data          <= {0, data[7:1]} & mask;
                       data[du_size] <= tr2_in;
      }

// SHR_EQL operation: performs the same operation as SHR. Instead of shift-out, the output is
// a comparison result (data0 == data1).
Comb:tr_out = data_eql_data1_in;
Reg:  data <= data;
      if      (tr0_in) data          <= data0_masked;
      else if (tr1_in) {
                       data          <= {0, data[7:1]} & mask;
                       data[du_size] <= tr2_in;
      }

// AND_OR operation: ANDs data1 and data0 along with mask; then, ORs all the bits of the
// ANDed output.
Comb:tr_out = | (data & data1_in & mask);
Reg:  data <= data;
      if (tr0_in) data <= data0_masked;
```

7.5.3 Routing

The Smart I/O block includes many switches that are used to route the signals in and out of the block and also between various components present inside the block. The routing switches are handled through the PRTGIO_PRTx_LUT_SELy and PRGIO_PRTx_DU_SEL registers. Refer to the Registers TRM for details. The Smart I/O internal routing is shown in Figure 7-7. In the figure, note that LUT7 to LUT4 operate on io_data/chip_data[7] to io_data/chip_data[4] whereas LUT3 to LUT0 operate on io_data/chip_data[3] to io_data/chip_data[0].

Figure 7-7. Smart I/O Routing



7.5.4 Operation

The Smart I/O block should be configured and operated as follows:

1. Before enabling the block, all the components should be configured and the routing should be selected, as explained in "Block Components" on page 63.
2. In addition to configuring the components and routing, some block level settings need to be configured correctly for desired operation.
   a. Bypass control: The Smart I/O path can be bypassed for a particular GPIO signal by setting the BYPASS[i] bit field in the PRGIO_PRTx_CTL register. When bit 'i' is set in the BYPASS[7:0] bit field, the ith GPIO signal is bypassed to the HSIOM signal path directly – Smart I/O logic will not be present in that signal path. This is useful when the Smart I/O functionality is required only on select I/Os.
   b. Pipelined trigger mode: The LUT3 input multiplexers and the LUT3 component itself do not include any combinatorial loops. Similarly, the data unit also does not include any combinatorial loops. However, when one LUT3 interacts with the other or to the data unit, inadvertent combinatorial loops are possible. To overcome this limitation, the PIPELINE_EN bit field of the PRGIO_PRTx_CTL register is used. When set, all the outputs (LUT3 and data unit) are

registered (flopped) before branching out to other components. The output will be unflopped when the PIPELINE_EN bit is cleared.

3. After the Smart I/O block is configured for the desired functionality, the block can be enabled by setting the ENABLED bit field of the PRGIO_PRTx_CTL register. If disabled, the Smart I/O block is put in bypass mode, where the GPIO signals are directly controlled by the HSIOM signals and vice-versa. The Smart I/O block must be configured; that is, all register settings must be updated before enabling the block to prevent glitches during register updates.

Table 7-7. Smart I/O Block Controls

Register [BIT_POS]	Bit Name	Description
PRGIO_PRTx_CTL[25]	PIPELINE_EN	Enable for pipeline register: '0': Disabled (register is bypassed). '1': Enabled
PRGIO_PRTx_CTL[31]	ENABLED	Enable Smart I/O. Should only be set to '1' when the Smart I/O is completely configured: '0': Disabled (signals are bypassed; behavior as if BYPASS[7:0] is 0xFF). When disabled, the block (data unit and LUTs) reset is activated. If the block is disabled: - The PIPELINE_EN register field should be set to '1', to ensure low power consumption. - The CLOCK_SRC register field should be set to 20 to 30 (clock is constant '0'), to ensure low power consumption. '1': Enabled. When enabled, it takes three "clk_block" clock cycles until the block reset is de-activated and the block becomes fully functional. This action ensures that the I/O pins' input synchronizer states are flushed when the block is fully functional.
PRGIO_PRTx_CTL[7:0]	BYPASS[7:0]	Bypass of the Smart I/O, one bit for each I/O pin: BYPASS[i] is for I/O pin i. When ENABLED is '1', this field is used. When ENABLED is '0', this field is not used and Smart I/O is always bypassed. '0': No bypass (Smart I/O is present in the signal path) '1': Bypass (Smart I/O is absent in the signal path)

7.6 I/O State on Power Up

During power up all the GPIOs are in high-impedance analog state and the input buffers are disabled. During run time, GPIOs can be configured by writing to the associated registers. Note that the pins supporting debug access port (DAP) connections (SWD lines) are always enabled as SWD lines during power up. However, the DAP connection can be disabled or reconfigured for general-purpose use through HSIOM. However, this reconfiguration takes place only after the device boots and start executing code.

7.7 Behavior in Low-Power Modes

shows the status of GPIOs in low-power modes.

Table 7-8. GPIO in Low-Power Modes

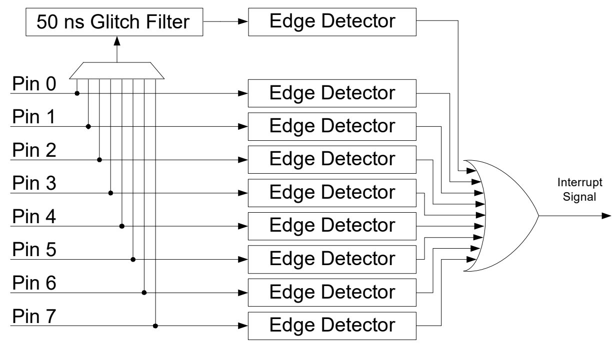
Low-Power Mode	Status
Sleep	■ GPIOs are active and can be driven by peripherals such as CapSense, CTB, SAR ADC, TCPWM, SCBs, and low-power comparators, which can work in sleep mode. ■ Input buffers are active; thus an interrupt on any I/O can be used to wake up the CPU. ■ AMUXBUS connections are available.
Deep-Sleep	■ GPIO output pin states are latched and remain in the frozen state, except the I^2C and SPI pins. SCB (I^2C and SPI) block can work in the deep-sleep mode and can wake up the CPU on address match or SPI slave select event. The low-power comparator can receive signals from its dedicated pins and can wake up the CPU. CTB is also functional in this mode with dedicated pins. ■ Input buffers are also active in this mode; pin interrupts are functional. ■ AMUXBUS connections are not available.

7.8 Interrupt

All the port pins in the device have the capability to generate interrupts. As shown in Figure 7-2, the pin signal is routed to the interrupt controller through the GPIO Edge Detect block.

Figure 7-8 shows the GPIO Edge Detect block architecture.

Figure 7-8. GPIO Edge Detect Block Architecture



An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, and both edges without reconfiguration. The edge detector is configured by writing into the EDGE_SEL bits of the Port Interrupt Configuration register, GPIO_PRTx_INTR_CFG, as shown in Table 7-9.

Table 7-9. Edge Detector Configuration

EDGE_SEL	Configuration
00	Interrupt is disabled
01	Interrupt on Rising Edge
10	Interrupt on Falling Edge
11	Interrupt on Both Edges

Besides the pins, edge detector is also present at the glitch filter output. This filter can be used on one of the pins of a port. The pin is selected by writing to the FLT_SEL field of the GPIO_PRTx_INTR_CFG register as shown in Table 7-10.

Table 7-10. Glitch filter Input Selection

FLT_SEL	Selected Pin
000	Pin 0 is selected
001	Pin 1 is selected
010	Pin 2 is selected
011	Pin 3 is selected
100	Pin 4 is selected
101	Pin 5 is selected
110	Pin 6 is selected
111	Pin 7 is selected

The edge detector outputs of a port are ORed together and then routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port. On a pin interrupt, it is required to know which pin caused an interrupt. This is done by reading the Port Interrupt Status register, GPIO_PRTx_INTR. This register not only includes the information on which pin triggered the interrupt, it also includes the pin status; it allows the CPU to read both information in a single read operation. This register has one more important use – to clear the interrupt. Writing ‘1’ to the corresponding status bit clears the pin interrupt. It is important to clear the interrupt status bit; otherwise, the interrupt will occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. Also, note that when the Port Interrupt Control Status register is read when an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code. Table 7-11 shows the Port Interrupt Status register bit fields.

Table 7-11. Port Interrupt Status Register

GPIO_PRTx_INTR	Description
0000b to 0111b	Interrupt status on pin 0 to pin 7. Writing ‘1’ to the corresponding bit clears the interrupt
1000b	Interrupt status from the glitch filter
10000b to 10111	Pin 0 to Pin 7 status
11000b	Glitch filter output status

The edge detector block output is routed to the Interrupt Source Multiplexer shown in Figure 6-3 on page 48, which gives an option of Level and Rising Edge detect. If the Level option is selected, an interrupt is triggered repeatedly as long as the Port Interrupt Status register bit is set. If the Rising Edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the Edge Detect block is used.

7.9 Peripheral Connections

7.9.1 Firmware Controlled GPIO

See to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

In addition to the data register, three other registers – GPIO_PRTx_DR_SET, GPIO_PRTx_DR_CLR, and GPIO_PRTx_INV – are provided to set, clear, and invert the output data respectively of a specific pin in a port without affecting other pins. Writing '1' into these registers will set, clear, or invert; writing '0' will have no affect on the pin status.

GPIO_PRTx_PS is the I/O pad register that provides the state of the GPIOs when read. Writes to this register have no effect.

7.9.2 Analog I/O

Analog resources, such as LPCOMP, SARMUX, andCTB, which require low-impedance routing paths have dedicated pins. Dedicated analog pins provide direct connections to specific analog blocks. They help improve performanceand should be given priority over other pins when using these analog resources. See thet device datasheet for details on these dedicated pins.

To configure a GPIO as a dedicated analog I/O, it should be configured in high-impedance analog mode (see Table 7-2) and the respective connection should be enabled in the specific analog resource. This can be done via registers associated with the respective analog resources.

To configure a GPIO as an analog pin connecting to AMUXBUS, it should be configured in high-impedance analog mode and then routed to AMUXBUS using the HSIOM_PORT_SELx register.

7.9.3 LCD Drive

All GPIOs have the capability of driving an LCD common or segment. HSIOM_PORT_SELx registers are used to select the pins for LCD drive. See the LCD Direct Drive chapter on page 217 for details.

7.9.4 CapSense

The pins that support CSD can be configured as CapSense widgets such as buttons, slider elements, touchpad elements, or proximity sensors. CapSense also requires external tank capacitors and shield lines. Table 7-12shows the GPIO and HSIOM settings required for CapSense. See the CapSense chapter on page 229 for more information.

Table 7-12. CapSense Settings

CapSense Pin	GPIO Drive Mode (GPIO_PRTx_PC)	Digital Input Buffer Setting (GPIO_PRTx_PC2)	HSIOM Setting
Sensor	High-Impedance Analog	Disable Buffer	CSD_SENSE
Shield	High-Impedance Analog	Disable Buffer	CSD_SHIELD
CMOD (normal operation)	High-Impedance Analog	Disable Buffer	AMUXBUS A or CSD_COMP
CMOD (GPIO precharge, only available in select GPIO)	High-Impedance Analog	Disable Buffer	AMUXBUS B or CSD_COMP
CSH TANK (GPIO precharge, only available in select GPIO)	High-Impedance Analog	Disable Buffer	AMUXBUS B or CSD_COMP

7.9.5 Serial Communication Block (SCB)

SCB, which can be configured as UART, I^2C, and SPI, has dedicated connections to the pin. See thet device datasheet for details on these dedicated pins. When the UART and SPI mode is used, the SCB controls the digital output buffer drive mode for the input pin to keep the pin in the high-impedance state. That is, the SCB block disables the output buffer at the UART Rx pin and MISO pin when configured as SPI master, and MOSI and select line when configured as SPI slave. This functionality overrides the drive mode settings, which is done using the GPIO_PRTx_PC register.

7.9.6 Timer, Counter, and Pulse Width Modulator (TCPWM) Block

TCPWM has dedicated connections to the pin. See the device datasheet for details on these dedicated pins. Note that when the TCPWM block inputs such as start and stop are taken from the pins, the drive mode can be only high-z digital because the TCPWM block disables the output buffer at the input pins.

7.10 Registers

Table 7-13. I/O Registers

Name	Description
GPIO_PRTx_DR	Port Output Data Register
GPIO_PRTx_DR_SET	Port Output Data Set Register
GPIO_PRTx_DR_CLR	Port Output Data Clear Register
GPIO_PRTx_DR_INV	Port Output Data Inverting Register
GPIO_PRTx_PS	Port Pin State Register - Reads the logical pin state of I/O
GPIO_PRTx_PC	Port Configuration Register - Configures the output drive mode, input threshold, and slew rate
GPIO_PRTx_PC2	Port Secondary Configuration Register - Configures the input buffer of I/O pin
GPIO_PRTx_INTR_CFG	Port Interrupt Configuration Register
GPIO_PRTx_INTR	Port Interrupt Status Register
HSIOM_PORT_SELx	HSIOM Port Selection Register
PRGIO_PRTx_CTL	Smart I/O port control register
PRGIO_PRTx_SYNC_CTL	Smart I/O Synchronization control register
PRGIO_PRTx_LUT_SELy	Smart I/O y^{th} LUT component input selection register
PRGIO_PRTx_LUT_CTLy	Smart I/O y^{th} LUT component control register
PRGIO_PRTx_DU_SEL	Smart I/O data unit input selection register
PRGIO_PRTx_DU_CTL	Smart I/O data unit control register
PRGIO_PRTx_DATA	Smart I/O data unit input data source register

Note The 'x' in the GPIO register name denotes the port number. For example, GPIO_PTR1_DR is the Port 1 output data register. The ‘x’ in the Smart I/O register name denotes the Smart I/O port number. The Smart I/O port number and the actual port number may vary. See 7.5 Smart I/O on page 63 for details.

8. Clocking System

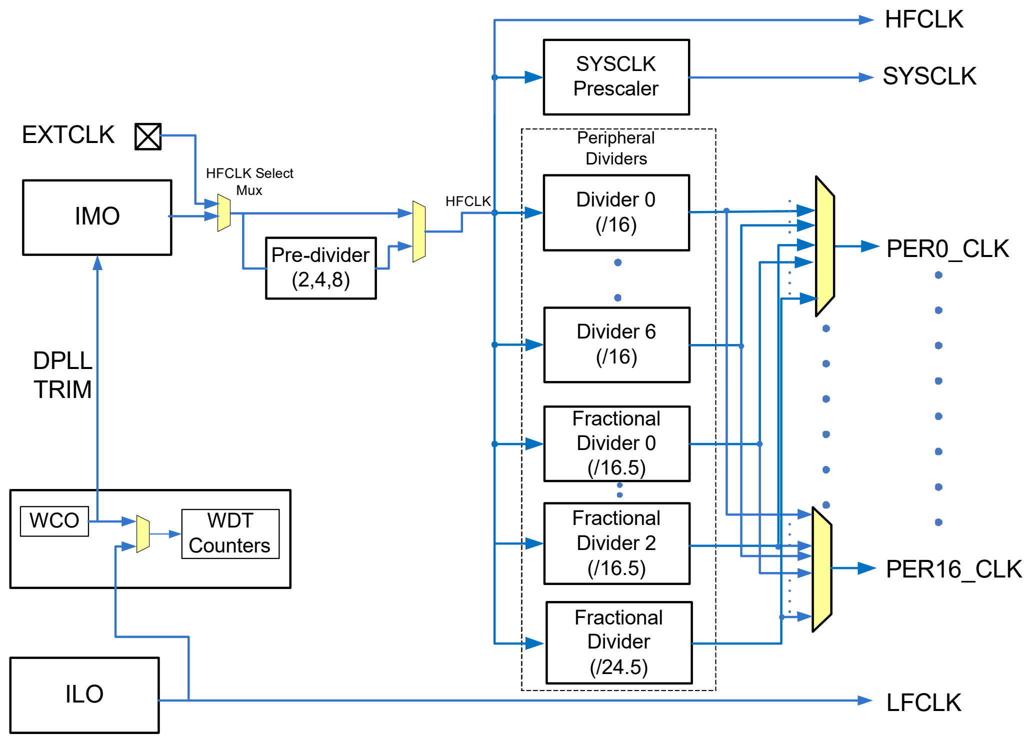
The PSoC® Analog Coprocessor clock system includes these clock resources:

- Two internal clock sources:
  - 3–48 MHz internal main oscillator (IMO) ±2 percent accuracy across all frequencies with trim
  - 40-kHz internal low-speed oscillator (ILO) with ±60 percent accuracy with trim (can be calibrated using the IMO)
- Two external clock sources:
  - External clock (EXTCLK) generated using a signal from an I/O pin
  - External 32-kHz watch crystal oscillator (WCO)
- High-frequency clock (HFCLK) of up to 48 MHz, selected from IMO or external clock
- Low-frequency clock (LFCLK) sourced by ILO or WCO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- Seven16-bit peripheral clock dividers
- Four fractional dividers for accurate clock generation
- Seventeen digital and analog peripheral clocks

8.1 Block Diagram

Figure 8-1 gives a generic view of the clocking system in PSoC Analog Coprocessor devices.

Figure 8-1. Clocking System Block Diagram



The four clock sources in the device are IMO, EXTCLK, WCO, and ILO, as shown in Figure 8-1. The HFCLK mux selects the HFCLK source from the EXTCLK or the IMO. The HFCLK frequency can be a maximum of 48 MHz.

8.2 Clock Sources

8.2.1 Internal Main Oscillator

The internal main oscillator (IMO) is an accurate, high-speed internal (crystal-less) oscillator that is available as the main clock source during Active and Sleep modes. It is the default clock source for the device. Its frequency can be changed in 4-MHz steps between 24 MHz and 48 MHz, with an accuracy of ±2 percent.

The IMO frequency is changed using the CLK_IMO_SELECT register. The default frequency is 24 MHz.

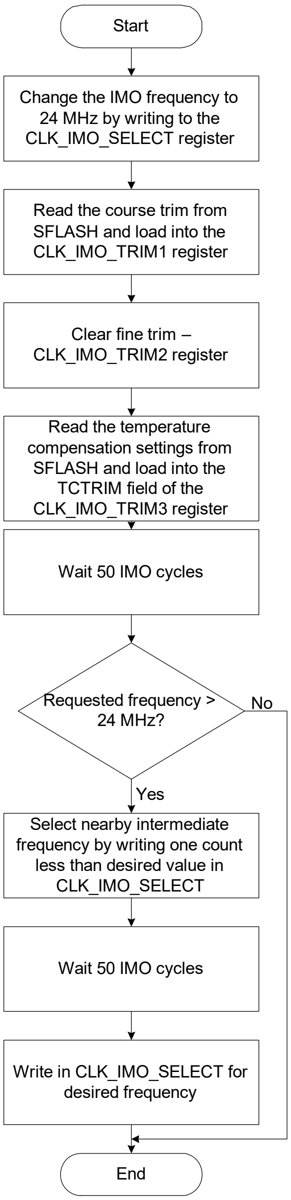
Table 8-1. IMO Frequency

CLK_IMO_SELECT[2:0]	Nominal IMO Frequency
0	24 MHz
1	28 MHz
2	32 MHz
3	36 MHz
4	40 MHz
5	44 MHz
6	48 MHz

To get the accurate IMO frequency, trim registers are provided – CLK_IMO_TRIM1 provides coarse trimming with a step size of 120 kHz, CLK_IMO_TRIM2 is for fine trimming with a step size of 15 kHz, and the TCTRIM field in CLK_IMO_TRIM3 is for temperature compensation. Trim settings are generated during manufacturing for every frequency that can be selected by CLK_IMO_SELECT. These trim settings are stored in SFLASH.

The trim settings are loaded during device startup; however, firmware can load new trim values and change the frequency in run time. Follow the algorithm in Figure 8-2 to change the IMO frequency.

Figure 8-2. Change IMO Frequency



8.2.1.1 Startup Behavior

After reset, the IMO is configured for 24-MHz operation. During the “boot” portion of startup, trim values are read from flash and the IMO is configured to achieve datasheet specified accuracy.

8.2.1.2 Programming Clock

IMO must be set to 48 MHz to program the flash. It is used to drive the charge pumps of flash and for program/erase timing purposes.

8.2.2 Internal Low-speed Oscillator

The internal low-speed oscillator operates with no external components and outputs a stable clock at 40-kHz nominal. The ILO is relatively low power and low accuracy. It can be calibrated periodically using a higher accuracy, high-frequency clock to improve accuracy. The ILO is available in all power modes except Hibernate and Stop modes. The ILO is used as the system low-frequency clock LFCLK in the device. The ILO is a relatively inaccurate (±60 percent overvoltage and temperature) oscillator, which is used to generate low-frequency clocks. If calibrated against the IMO when in operation, the ILO is accurate to ±10 percent for stable temperature and voltage. The ILO is enabled and disabled with register CLK_ILO_CONFIG bit ENABLE.

8.2.3 External Clock (EXTCLK)

The external clock (EXTCLK) is a MHz range clock that can be generated from a signal on a designated PSoC Analog Coprocessor pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is 1–48 MHz. The device always starts up using the IMO and the external clock must be enabled in user mode; so the device cannot be started from a reset, which is clocked by the external clock.

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the I/O System chapter on page 57 for more details.

8.2.4 Watch Crystal Oscillator (WCO)

The PSoC device contains an oscillator to drive a 32.768-kHz watch crystal. Similar to ILO, WCO is also available in all modes. This clock has low power consumption, which makes it ideal for operation in low-power modes such as the Deep-Sleep mode. The WCO is enabled and disabled with the WCO_CONFIG register’s ENABLE bit.

WCO can be forced into low-power mode by setting the WCO_CONFIG[0] bit. Alternatively, the block can be put in the Auto mode where low-power mode transition happens only when the device goes into Deep-Sleep mode. This mode is enabled by setting WCO_CONFIG[1]. Note that the Auto mode will be overridden if the block is forced to low-power mode by setting WCO_CONFIG[0]. During the switching, the WCO output can experience some frequency disturbances. Hence, Auto mode is not suggested for high-accuracy applications such as RTC.

The difference in operation between the normal and low-power mode is the amplifier gain. The low-power mode is expected to have a lower amplifier gain to effectively reduce power. The amplifier gain for the two modes can be set in the WCO_TRIM register.

The IMO supports locking to the WCO. The WCO contains the logic to measure and compare the IMO clock and trim the IMO. The WCO implements a digital phased lock loop scheme to support a clock accuracy of ±1 percent. The IMO trimming logic of the WCO can be enabled by the use of the DPLL_ENABLE bit of the WCO_CONFIG. The user firmware, when using this feature, must make sure that there is a minimum time of 500 ms between the WCO enable and the DPLL_ENABLE events.

8.3 Clock Distribution

PSoC Analog Coprocessor clocks are developed and distributed throughout the device, as shown in Figure 8-1. The distribution configuration options are as follows:

- HFCLK input selection
- LFCLK input selection
- SYSCLK prescaler configuration
- Peripheral divider configuration

8.3.1 .HFCLK Input Selection

HFCLK in PSoC Analog Coprocessor has two input options: IMO and EXTCLK. The HFCLK input is selected using the CLK_SELECT register's HFCLK_SEL bits, as described in Table 8-2.

Table 8-2. HFCLK Input Selection Bits HFCLK_SEL

Name	Description
HFCLK_SEL[2:0]	HFCLK input clock selection 0: IMO. Uses the IMO as the source of the HFCLK 1: EXTCLK. Uses the EXTCLK as the source of the HFCLK 2–7: Reserved. Do not use

Pre-divider is provided for HFCLK to limit the peak current of the device. The divider options are 2, 4, and 8 configured using HFCLK_DIV bits of the CLK_SELECT register. Default divider is 4.

8.3.2 LFCLK Input Selection

Only the ILO can be the source for LFCLK in the PSoC Analog Coprocessor device.

8.3.3 SYSCLK Prescaler Configuration

The SYSCLK Prescaler allows the device to divide the HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by powers of 2 between 2^0 = 1 and 2^7 = 128. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in Table 8-3. The prescaler is initially configured to divide by 1.

Table 8-3. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

Name	Description
SYSCLK_DIV[3:0]	SYSCLK prescaler divide value 0: SYSCLK = HFCLK 1: SYSCLK = HFCLK/2 2: SYSCLK = HFCLK/4 3: SYSCLK = HFCLK/8 4: SYSCLK = HFCLK/16 5: SYSCLK = HFCLK/32 6: SYSCLK = HFCLK/64 7: SYSCLK = HFCLK/128

8.3.4 Peripheral Clock Divider Configuration

PSoC Analog Coprocessor has 11 clock dividers, which include seven 16-bit clock dividers and three 16.5-bit fractional clock dividers, and one 24.5-bit fractional clock divider. Fractional clock dividers allow the clock divisor to include a fraction of 0..31/32. The formula for the output frequency of a fractional divider is Fout = Fin / (INT16_DIV + (FRAC5_DIV/32)). For example, a 16.5-divider with an integer divide value of 2 (INT16_DIV=3, FRAC5_DIV=0), produces signals to generate a 16-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 3 (INT16_DIV=3, FRAC5_DIV=0), produces signals to generate a 12-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 2 (INT16_DIV=3) and a fractional divider of 16 (FRAC5_DIV=16) produces signals to generate a 13.7-MHz clock from a 48-MHz HFCLK. Not all 13.7-MHz clock periods are equal in size; half of them will be 3 HFCLK cycles and half of them will be 2 HFCLK cycles.

Fractional dividers are useful when a high-precision clock is required (for example, for a UART/SPI serial interface). Fractional dividers are not used when a low jitter clock is required, because the clock periods have a jitter of 1 HFCLK cycle.

The 24.5-bit fractional dividers are configured using the PERI_DIV_24_5_CTL0 register. Table 8-4 describes the configuration for these registers.

Table 8-4. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_24_5_CTLx

Bits	Name	Description
0	ENABLE_0	Divider enabled. Hardware sets this field to '1' as a result of an ENABLE command and to '0' as a result of a DISABLE command.
7:3	FRAC5_DIV_0	Fractional division by (FRAC5_DIV/32). Allows for fractional divisions in the range [0, 31/32]. Note that fractional division results in clock jitter as some clock periods may be 1 clk_hf cycle longer than other clock periods.
31:8	INT24_DIV_0	Integer division by (1+INT24_DIV). Allows for integer divisions in the range [1, 16,777,216].

Each divider can be enabled using the PERI_DIV_CMD register. This register acts as the command register for all 16 integer dividers and four fractional dividers. The PERI_DIV_CMD register format is as follows.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Description	Enable	Disable															PA_SEL_TYPE		PA_SEL_DIV						SEL_TYPE		SEL_DIV					

The SEL_TYPE field specifies the type of divider being configured. This field is '1' for the 16-bit integer divider and '2' for the 16.5-bit fractional divider.

The SEL_DIV field specifies the number of the specific divider being configured. For the integer dividers, this number ranges from 0 to 15. For fractional dividers, this field is any value in the range 0 to 3. When SEL_TYPE = 63 and SEL_TYPE = 3, no divider is specified.

The (PA_SEL_TYPE, PA_SEL_DIV) field pair allows a divider to be phase-aligned with another divider. The PA_SEL_DIV specifies the divider which is phase aligned. Any enabled divider can be used as a reference. The PA_SEL_TYPE specifies the type of the divider being phase aligned. When PA_SEL_DIV = 63 and PA_SEL_TYPE = 3, HFCLK is used as a reference.

Consider a 48-MHz HFCLK and a need for a 12-MHz divided clock A and a 8-MHz divided clock B. Clock A uses a 16-bit integer divider 0 and is created by aligning it to HF_CLK ((PA_SEL_TYPE, PA_SEL_DIV) is (3, 63)) and DIV_16_CTL0.INT16_DIV is 3. Clock B uses the integer divider 1 and is created by aligning it to clock A ((PA_SEL_TYPE, PA_SEL_DIV) is (1, 0)) and DIV_16_CTL1.INT16_DIV is 5. This guarantees that clock B is phase-aligned with clock A as the smallest common multiple of the two clock periods is 12 HFCLK cycles, the clocks A and B will be aligned every 12 HFCLK cycles. Note that clock B is phase-aligned to clock A, but still uses HFCLK as a reference clock for its divider value.

Each peripheral block in PSoC has a unique peripheral clock (PERI#_CLK) associated with it. Each of the peripheral clocks have a multiplexed input, which can take the input clock from any of the existing clock dividers.

Table 8-5 shows the mapping of the mux output to the corresponding peripheral blocks (shown in Figure 8-1). Any of the peripheral clock dividers can be mapped to a specific peripheral by using their respective PERI_PCLK_CTLx register.

Table 8-5. Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
0	SCB0
1	SCB1
2	SCB2
3	CSD
4	TCPWM0
5	TCPWM1
6	TCPWM2
7	TCPWM3
8	TCPWM4
9	TCPWM5
10	TCPWM6

Table 8-5. Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
11	TCPWM7
12	SAR
13	UAB (First Half)
14	UAB (Second Half)
15	SmartIO
16	LCD

8.4 Low-Power Mode Operation

The high-frequency clocks including the IMO, EXTCLK, HFCLK, SYSCLK, and peripheral clocks operate only in Active and Sleep modes. The ILO, WCO, and LFCLK operate in all power modes.

8.5 Register List

Table 8-6. Clocking System Register List

Register Name	Description
CLK_IMO_TRIM1	IMO Trim Register - This register contains IMO trim for course correction.
CLK_IMO_TRIM2	IMO Trim Register - This register contains IMO trim for fine correction.
CLK_IMO_TRIM3	IMO Trim Register - This register contains the temperature compensation trim settings for IMO and trim settings to adjust the step size of the course and fine correction of IMO frequency.
PWR_BG_TRIM1	Bandgap Trim Registers - These registers control the trim of the bandgap reference, allowing manipulation of the voltage references in the device.
PWR_BG_TRIM2	
CLK_ILO_CONFIG	ILO Configuration Register - This register controls the ILO configuration.
CLK_IMO_CONFIG	IMO Configuration Register - This register controls the IMO configuration.
CLK_SELECT	Clock Select - This register controls clock tree configuration, selecting different sources for the system clocks.
WCO_CONFIG	WCO Enable. This register enables or disables the external watch crystal oscillator.
PERI_DIV_16_CTLx	Peripheral Clock Divider Control Registers - These registers configure the peripheral clock dividers, setting integer divide value, and enabling or disabling the divider.
PERI_DIV_16_5_CTLx	Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting fractional divide value, and enabling or disabling the divider.
PERI_PCLK_CTLx	Programmable Clock Control Registers - These registers are used to select the input clocks to peripherals.
PERI_DIV_24_5_CTLx	Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting the fractional divide value and enabling or disabling the divider.

9. Power Supply and Monitoring

PSoC® Analog Coprocessor is capable of operating from a 1.71 V to 5.5 V externally supplied voltage. This is supported through one of the two following operating ranges:

- 1.80 V to 5.50 V supply input to the internal regulators
- 1.71 V to 1.89 V[1] direct supply

There are two internal regulators to support the various power modes - Active digital regulator and Deep-Sleep regulator.

9.1 Block Diagram

Figure 9-1. Power System Block Diagram

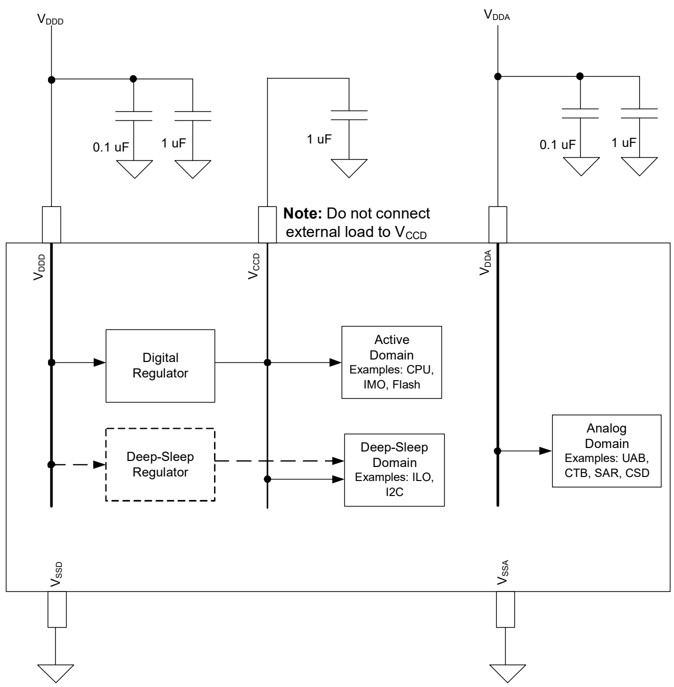


Figure 9-1 shows the power system diagram and all the power supply pins. The system has one regulator in Active mode for the digital circuitry. There is no analog regulator; the analog circuits run directly from the V_{DDA} input. There is a separate regulator for Deep-Sleep mode.

The supply voltage range is 1.71 V to 5.5 V with all functions and circuits operating in that range. The device allows two distinct modes of power supply operation: unregulated external supply and regulated external supply modes.

1. When the system supply is in the range 1.80 V to 1.89 V, both direct supply and internal regulator options can be used. The selection can be made depending on the user's system capability. Note that the supply voltage cannot go above 1.89 V for the direct supply option because it will damage the device. It should not go below 1.80 V for the internal regulator option because the regulator will turn off.

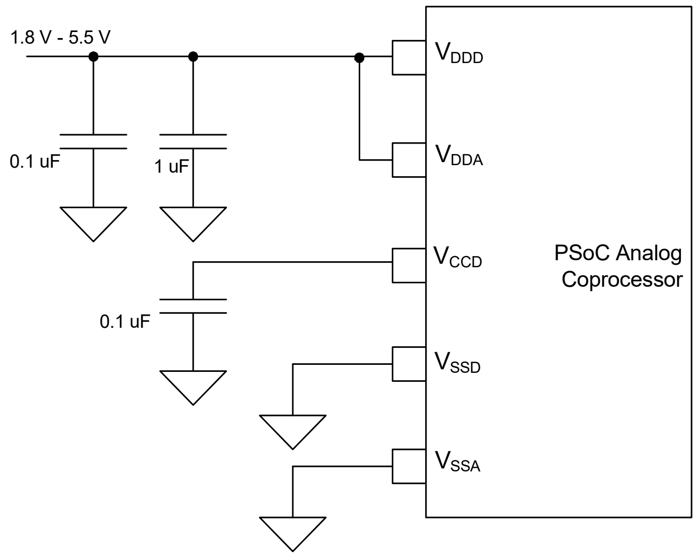
9.2 Power Supply Scenarios

The following diagrams illustrate the different ways in which the device is powered.

9.2.1 Single 1.8 V to 5.5 V Unregulated Supply

If a 1.8-V to 5.5-V supply is to be used as the unregulated power supply input, it should be connected as shown in Figure 9-2.

Figure 9-2. Single Regulated V_{DDD} Supply



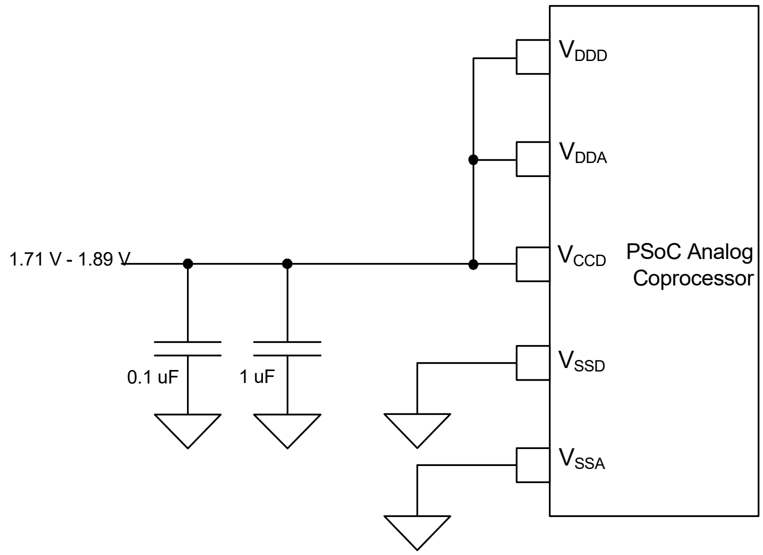
In this mode, the device is powered by an external power supply that can be anywhere in the range of 1.8 V to 5.5 V. This range is also designed for battery-powered operation; for instance, the chip can be powered from a battery system that starts at 3.5 V and works down to 1.8 V. In this mode, the internal regulator supplies the internal logic. The V_{CCD} output must be bypassed to ground via a 0.1 µF external ceramic capacitor.

Bypass capacitors are also required from V_{DDD} to ground; typical practice for systems in this frequency range is to use a bulk capacitor in the 1 µF to 10 µF range in parallel with a smaller ceramic capacitor (0.1 µF, for example). Note that these are simply rules of thumb and that, for critical applications, the PCB layout, lead inductance, and the bypass capacitor parasitic should be simulated to design and obtain optimal bypassing.

9.2.2 Direct 1.71 V to 1.89 V Regulated Supply

In direct supply configuration, V_{CCD} and V_{DDD} are shorted together and connected to a 1.71-V to 1.89-V supply. This regulated supply should be connected to the device, as shown in Figure 9-3.

Figure 9-3. Single Unregulated V_{DDD} Supply



In this mode, V_{CCD} and V_{DDD} pins are shorted together and bypassed. The internal regulator should be disabled in firmware. See 9.3.1.1 Active Digital Regulator on page 85 for details.

9.3 How It Works

The regulators in Figure 9-1 power the various domains of the device. All the core regulators draw their input power from the V_{DDD} pin supply. The analog circuits run directly from the V_{DDA} input.

9.3.1 Regulator Summary

9.3.1.1 Active Digital Regulator

Table 9-1. Regulator Status in Different Power Modes

Mode	Active Digital Regulator	Deep-Sleep Regulator
Deep-Sleep	Off	On
Sleep	On	On
Active	On	On

For external supplies from 1.8 V and 5.5 V, the Active digital regulator provides the main digital logic in Active and Sleep modes. This regulator has its output connected to a pin (V_{CCD}) and requires an external decoupling capacitor (1 µF X5R).

For supplies below 1.8 V, V_{CCD} must be supplied directly. In this case, V_{CCD} and V_{DDD} must be shorted together, as shown in Figure 9-3.

The Active digital regulator can be disabled by setting the EXT_VCCD bit in the PWR_CONTROL register. This action reduces the power consumption in direct supply mode. The Active digital regulator is available only in Active and Sleep power modes.

9.3.1.2 Deep-Sleep Regulator

This regulator supplies the circuits that remain powered in Deep-Sleep mode, such as the ILO, WCO, and SCB (I^2C/SPI), and low-power comparator. The Deep-Sleep regulator is available in all power modes. In Active and Sleep power modes, the main output of this regulator is connected to the output of the Active digital regulator (V_{CCD}).

9.4 Voltage Monitoring

The voltage monitoring system includes power-on-reset (POR) brownout detection (BOD).

9.4.1 Power-On-Reset (POR)

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor V_{CCD} voltage. Typically, the POR circuits are not very accurate with respect to trip-point. POR circuits are used during initial chip power-up and then disabled.

9.4.1.1 Brownout-Detect (BOD)

The BOD circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. BOD circuit monitors the V_{CCD} voltage. The BOD circuit generates a reset if a voltage excursion dips below the minimum V_{CCD} voltage required for safe operation (see the device datasheet for details). The system will not come out of RESET until the supply is detected to be valid again.

To ensure reliable operation of the device, the watchdog timer should be used in all designs. Watchdog timer provides protection against abnormal brownout conditions that may compromise the CPU functionality. See Watchdog Timer chapter on page 92 for more details.

9.5 Register List

Table 9-2. Power Supply and Monitoring Register List

Register Name	Description
PWR_CONTROL	Power Mode Control Register – This register allows configuration of device power modes and regulator activity.

10. Chip Operational Modes

PSoC® Analog Coprocessor is capable of executing firmware in four different modes. These modes dictate execution from different locations in flash and ROM, with different levels of hardware privileges. Only three of these modes are used in end-applications; debug mode is used exclusively to debug designs during firmware development.

PSoC 4Analog Coprocessor operational modes are:

- Boot
- User
- Privileged
- Debug

10.1 Boot

Boot mode is an operational mode where the device is configured by instructions hard-coded in the device SROM. This mode is entered after the end of a reset, provided no debug-acquire sequence is received by the device. Boot mode is a privileged mode; interrupts are disabled in this mode so that the boot firmware can set up the device for operation without being interrupted. During boot mode, hardware trim settings are loaded from flash to guarantee proper operation during power-up. When boot concludes, the device enters user mode and code execution from flash begins. This code in flash may include automatically generated instructions from the PSoC Creator IDE that will further configure the device.

10.2 User

User mode is an operational mode where normal user firmware from flash is executed. User mode cannot execute code from SROM. Firmware execution in this mode includes the automatically generated firmware by the PSoC Creator IDE and the firmware written by the user. The automatically generated firmware can govern both the firmware startup and portions of normal operation. The boot process transfers control to this mode after it has completed its tasks.

10.3 Privileged

Privileged mode is an operational mode, which allows execution of special subroutines that are stored in the device ROM. These subroutines cannot be modified by the user and are used to execute proprietary code that is not meant to be interrupted or observed. Debugging is not allowed in privileged mode.

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see "Performing a System Call" on page 242. Exit from this mode returns the device to user mode.

10.4 Debug

Debug mode is an operational mode that allows observation of the PSoC operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as PSoC Creator and ARM MDK to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the Program and Debug Interface chapter on page 234.

For more details on protection modes, see the Device Security chapter on page 99.

11. Power Modes

The PSoC® Analog Coprocessor provides three power modes, intended to minimize the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep-Sleep

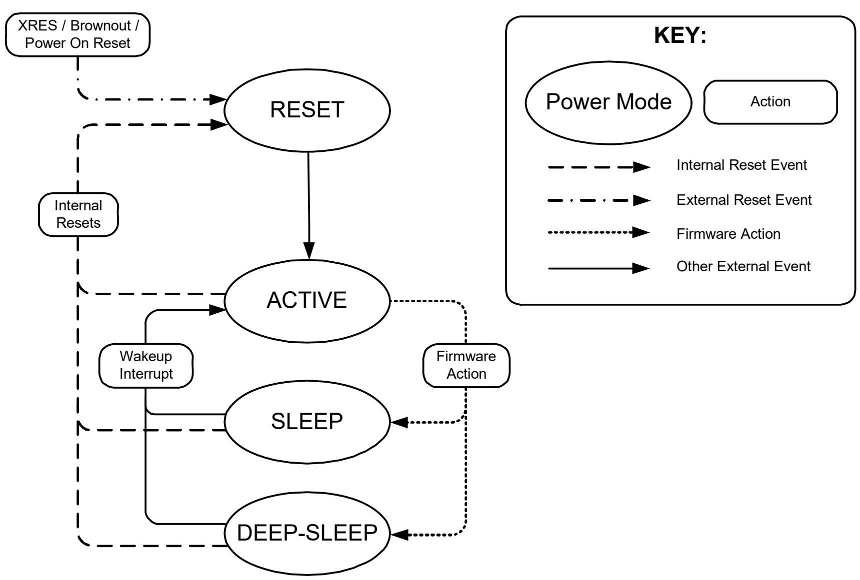
Active, Sleep, and Deep-Sleep are standard ARM-defined power modes, supported by the ARM CPUs.

The power consumption in different power modes is controlled by using the following methods:

- Enabling/disabling peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the PSoC

Figure 11-1 illustrates the various power modes and the possible transitions between them.

Figure 11-1. Power Mode Transitions State Diagram



Note: ARM nomenclature for Deep-Sleep power mode is 'SLEEPDEEP'.

Table 11-1 illustrates the power modes offered by PSoC Analog Coprocessor.

Table 11-1. PSoC Analog Coprocessor Power Modes

Power Mode	Description	Entry Condition	Wakeup Sources	Active Clocks	Wakeup Action	Available Regulators
Active	Primary mode of operation; all peripherals are available (programmable).	Wakeup from other power modes, internal and external resets, brownout, power on reset	Not applicable	All (programmable)	N/A	All regulators are available. The Active digital regulator can be disabled if external regulation is used.
Sleep	CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable).	Manual register write	Any enabled interrupt	All (programmable) except CPU clock	Interrupt	All regulators are available. The Active digital regulator can be disabled if external regulation is used.
Deep-Sleep	All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency clock is available. Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup.	Manual register write	GPIO interrupt, low-power comparator, SCB, watchdog timer	ILO (40 kHz), WCO (32 kHz)	Interrupt	Deep-Sleep regulator

In addition to the wakeup sources mentioned in Table 11-1, external reset (XRES) and brownout reset bring the device to Active mode from any power mode.

11.1 Active Mode

Active mode is the primary power mode of the PSoC device. This mode provides the option to use every possible subsystem/peripheral in the device. In this mode, the CPU is running and all the peripherals are powered. The firmware may be configured to disable specific peripherals that are not in use, to reduce power consumption.

11.2 Sleep Mode

This is a CPU-centric power mode. In this mode, the Cortex-M0+ CPU enters Sleep mode and its clock is disabled. It is a mode that the device should come to very often or as soon as the CPU is idle, to accomplish low power consumption. It is identical to Active mode from a peripheral point of view. Any enabled interrupt can cause wakeup from Sleep mode.

11.3 Deep-Sleep Mode

In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The high-frequency clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (40 kHz) oscillator remains on and low-frequency peripherals continue to operate. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, I^2C slave) continue to operate. Interrupts from low-speed, asynchronous or low-power analog peripherals can cause a wakeup from Deep-Sleep mode. CTB can also operate in this mode with reduced power and bandwidth. For details on power consumption and CTB bandwidth, refer to the device datasheet.

The available wakeup sources are listed in Table 11-3.

11.4 Power Mode Summary

Table 11-3 illustrates the peripherals available in each low-power mode; Table 11-3 illustrates the wakeup sources available in each power mode.

Table 11-2. Available Peripherals

Peripheral	Active	Sleep	Deep-Sleep
CPU	Available	Retention[a]	Retention
SRAM	Available	Retention	Retention
High-speed peripherals	Available	Available	Retention
Low-speed peripherals	Available	Available	Available (optional)
Internal main oscillator (IMO)	Available	Available	Not Available
Internal low-speed oscillator (ILO, 40 kHz)	Available	Available	Available (optional)
Asynchronous peripherals (peripherals that do not run on internal clock)	Available	Available	Available
Power-on-reset, Brownout detection	Available	Available	Available
Analog mux bus connection	Available	Available	Available
GPIO output state	Available	Available	Available

a. The configuration and state of the peripheral is retained. Peripheral continues its operation when the device enters Active mode.

Table 11-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any enabled interrupt source	Interrupt
	CTB	Interrupt
Deep-Sleep	GPIO interrupt	Interrupt
	I2C address match	Interrupt
	Watchdog timer	Interrupt/Reset
	Low-power comparator	Interrupt

Note: In addition to the wakeup sources mentioned in Table 11-3, external reset (XRES) and brownout reset bring the device to Active mode from any power mode. XRES and brownout trigger a full system restart. All the states including frozen GPIOs are lost. In this case, the cause of wakeup is not readable after the device restarts.

11.5 Low-Power Mode Entry and Exit

A Wait For Interrupt (WFI) instruction from the Cortex-M0+ (CM0+) triggers the transitions into Sleep and Deep-Sleep mode. The Cortex-M0+ can delay the transition into a low-power mode until the lowest priority ISR is exited (if the SLEEPONEXIT bit in the CM0 System Control Register is set).

The transition to Sleep and Deep-Sleep modes are controlled by the flags SLEEPDEEP in the CM0 System Control Register (CM0_SCR).

- Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 0.
- Deep-Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 1.

The LPM READY bit in the PWR_CONTROL register shows the status of Deep-Sleep regulator. If the firmware tries to enter Deep-Sleep mode before the regulators are ready, then PSoC Analog Coprocessor goes to Sleep mode first, and when the regulators are ready, the device enters Deep-Sleep mode. This operation is automatically done in hardware.

In Sleep and Deep-Sleep modes, a selection of peripherals are available (see Table 11-3), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode. See the Interrupts chapter on page 47 and the Reset System chapter on page 97 for details.

11.6 Register List

Table 11-4. Power Mode Register List

Register Name	Description
CM0_SCR	System Control - Sets or returns system control data.
PWR_CONTROL	Power Mode Control - Controls the device power mode options and allows observation of current state.

12. Watchdog Timer

The watchdog timer (WDT) is used to automatically reset the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality. The WDT runs from the LFCLK, generated by the ILO. The timer must be serviced periodically in firmware to avoid a reset. Otherwise, the timer will elapse and generate a device reset. The WDT can be used as an interrupt source or a wakeup source in low-power modes.

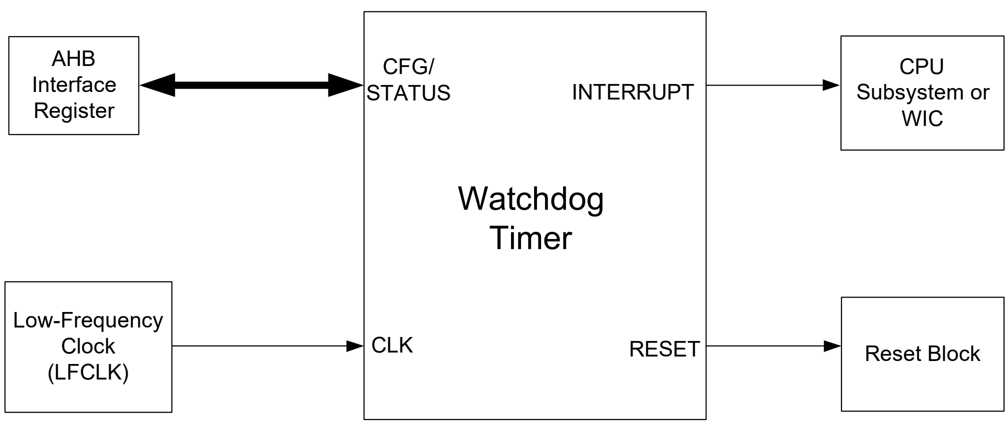
12.1 Features

The WDT has these features:

- System reset generation after a configurable interval
- Periodic interrupt/wake up generation in Active, Sleep, and Deep-Sleep power modes
- Features a 16-bit free-running counter

12.2 Block Diagram

Figure 12-1. Watchdog Timer Block Diagram



12.3 How It Works

The WDT asserts a hardware reset to the device on the third WDT match event, unless it is periodically serviced in firmware. The WDT interrupt has a programmable period of up to 2048 ms. The WDT is a free-running wraparound up-counter with a maximum of 16-bit resolution. The resolution is configurable as explained later in this section.

The WDT_COUNTER register provides the count value of the WDT. The WDT generates an interrupt when the count value in WDT_COUNTER equals the match value stored in the WDT_MATCH register, but it does not reset the count to '0'. Instead, the WDT keeps counting until it overflows (after 0xFFFF when the resolution is set to 16 bits) and rolls back to 0. When the count value again reaches the match value, another interrupt is generated. Note that the match count can be changed when the counter is running.

A bit named WDT_MATCH in the SRSS_INTR register is set whenever the WDT interrupt occurs. This interrupt must be cleared by writing a '1' to the WDT_MATCH bit in SRSS_INTR to reset the watchdog. If the firmware does not reset the WDT for two consecutive interrupts, the third match event will generate a hardware reset.

The IGNORE_BITS in the WDT_MATCH register can be used to reduce the entire WDT counter period. The ignore bits can specify the number of MSBs that need to be discarded. For example, if the IGNORE_BITS value is 3, then the WDT counter becomes a 13-bit counter. For details, see the WDT_COUNTER, WDT_MATCH, and SRSS_INTR registers in the .

When the WDT is used to protect against system crashes, clearing the WDT interrupt bit to reset the watchdog must be done from a portion of the code that is not directly associated with the WDT interrupt. Otherwise, even if the main function of the firmware crashes or is in an endless loop, the WDT interrupt vector can still be intact and feed the WDT periodically.

The safest way to use the WDT against system crashes is to:

- Configure the watchdog reset period such that firmware is able to reset the watchdog at least once during the period, even along the longest firmware delay path.
- Reset the watchdog by clearing the interrupt bit regularly in the main body of the firmware code by writing a '1' to the WDT_MATCH bit in SRSS_INTR register.
- It is not recommended to reset watchdog in the WDT interrupt service routine (ISR), if WDT is being used as a reset source to protect the system against crashes. Hence, it is not recommended to use WDT reset feature and ISR together.

Follow these steps to use WDT as a periodic interrupt generator:

1. Write the desired IGNORE_BITS in the WDT_MATCH register to set the counter resolution.
2. Write the desired match value to the WDT_MATCH register.
3. Clear the WDT_MATCH bit in SRSS_INTR to clear any pending WDT interrupt.
4. Enable the WDT interrupt by setting the WDT_MATCH bit in SRSS_INTR_MASK
5. Enable global WDT interrupt in the CM0_ISER register (See the Interrupts chapter on page 47 for details).
6. In the ISR, clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.

For more details on interrupts, see the Interrupts chapter on page 47.

12.3.1 Enabling and Disabling WDT

The watchdog counter is a free-running counter that cannot be disabled. However, it is possible to disable the watchdog reset by writing a key '0xACED8865' to the WDT_DISABLE_KEY register. Writing any other value to this register will enable the watchdog reset. If the watchdog system reset is disabled, the firmware does not have to periodically reset the watchdog to avoid a system reset. The watchdog counter can still be used as an interrupt source or wakeup source. The only way to stop the counter is to disable the ILO by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The watchdog reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the watchdog reset will automatically enable the ILO.

Note Disabling the WDT reset is not recommended if:

- Protection is required against firmware crashes
- The power supply can produce sudden brownout events that may compromise the CPU functionality

12.3.2 WDT Interrupts and Low-Power Modes

The watchdog counter can send interrupt requests to the CPU in Active power mode and to the WakeUp Interrupt Controller (WIC) in Sleep and Deep-Sleep power modes. It works as follows:

- **Active Mode:** In Active power mode, the WDT can send the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.
- **Sleep or Deep-Sleep Mode:** In this mode, the CPU subsystem is powered down. Therefore, the interrupt request from the WDT is directly sent to the WIC, which will then wake up the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.

For more details on device power modes, see the Power Modes chapter on page 88.

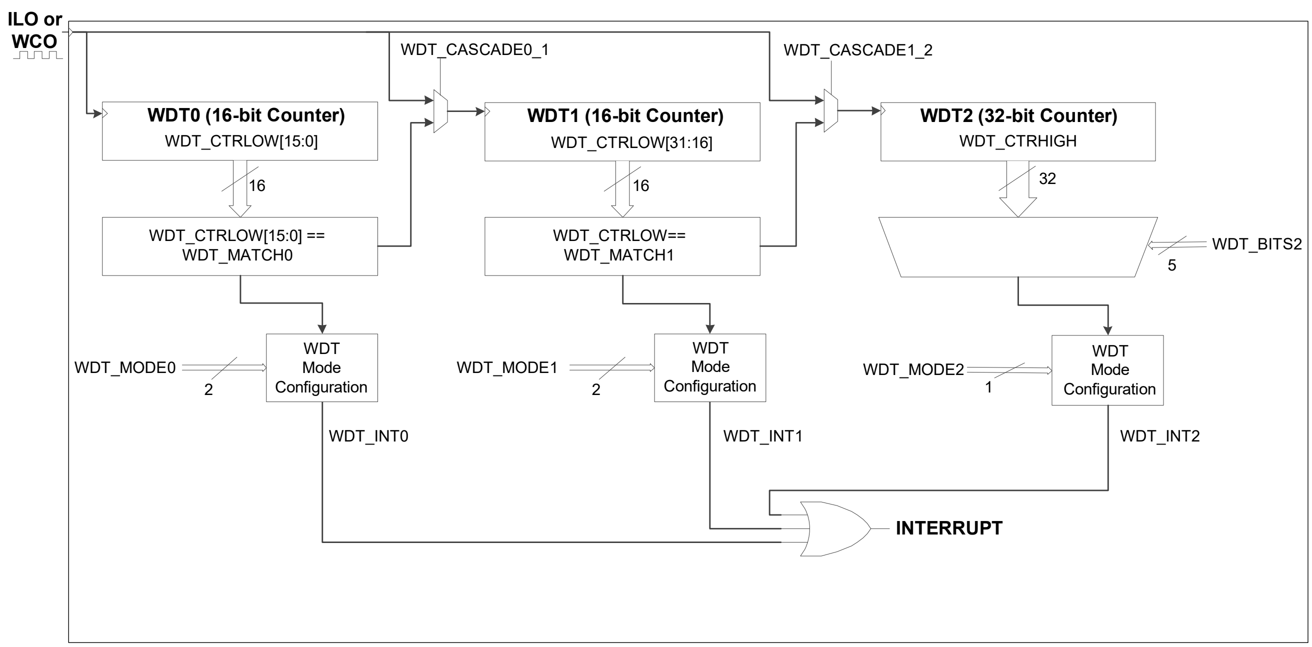
12.3.3 WDT Reset Mode

The RESET_WDT bit in the RES_CAUSE register indicates the reset generated by the WDT. This bit remains set until cleared or until a power-on reset (POR), brownout reset (BOD), or external reset (XRES) occurs. All other resets leave this bit untouched. For more details, see the Reset System chapter on page 97.

12.4 Additional Timers

Besides WDT, there are three additional up-counting timers for general-purpose use – WDT0, WDT1, and WDT2. These three timers are clocked either from ILO or WCO, selected by writing into the WCO_WDT_CLKEN register. These timers can run in Active, Sleep, and Deep-Sleep modes and are capable of generating interrupts.

Figure 12-2. WDT Additional Timers Block Diagram



12.4.1 WDT0 and WDT1

These are 16-bit timers, which can be operated in two configurations:

- Free running
- Clear on match (configurable period)

In the free-running mode, the timer counts throughout the 16-bit range. On reaching 65535 (2^{16}–1), the timer resets to 0 and starts counting again. In the Clear-on-match mode, the match count written in WDT_MATCH0 and WDT_MATCH1 of the WCO_WDT_MATCH register decides the period of WDT0 and WDT1, respectively. When the timer count reaches the match value, the timer resets to 0 and starts counting again. One of these two configurations is selected using WDT_CLEAR0 and WDT_CLEAR1 bits of the WCO_WDT_CONFIG register. The Clear-on-match mode is selected by writing ‘1’ to WDT_CLEARx. Writing ‘0’ to this bit disables the clearing of timer on match count and the free-running mode is configured. Note that changing the match count requires three input clock cycles to come into effect. Before putting the device to deep sleep, ensure delay of at least one input clock cycle after the match count update.

An interrupt can be generated on match or timer overflow by writing into WDT_MODE bits of the WCO_WDT_CONFIG register. On interrupt, the WDT_INTx bit of the WCO_WDT_CONTROL register is set. This bit must be cleared by firmware to allow the next interrupt trigger. Note that the interrupts from all the three timers are ORed to generate a single trigger to the CPU. To identify which timer caused an interrupt, read the WDT_INTx bit.

The timers are enabled by writing ‘1’ to the WDT_ENABLEx bit of the WCO_WDT_CONTROL register. Note that it takes three clock cycles to take effect. It is not recommended to toggle this bit more than once during this time. After enabling the timer, it is not recommended to write to the configuration register (WCO_WDT_CONFIG). The present value of the timers can be read from the WDT_CTRLOW register; it can be reset by writing ‘1’ to the WDT_RESETx bit of the WCO_WDT_CONTROL register.

12.4.2 WDT2

It is similar to WDT0 and WDT1 with following differences:

- WDT2 is a 32-bit up-counting timer
- Supports only free-running configuration with counting range of 0 to (2^{32}–1)
- The interrupt is triggered when one out of 32 bits toggles during counting. The bit position is configured using the 5-bit WDT_BITS2 field of the WCO_WDT_CONFIG register. Setting it to ‘0’ results in an interrupt on every input clock; setting it to ‘1’ results in an interrupt on alternate clocks; setting it to ‘31’ results in an interrupt every 2^{31} clocks.

12.4.3 Cascading

The cascading options are as follows:

- WDT0 and WDT1 timers can be cascaded by writing into WDT_CASCADE0_1 bit of the WCO_WDT_CONFIG register. When cascaded, WDT1 increments after WDT0 reaches its match count.
- WDT1 and WDT2 timers can also be cascaded by writing into WDT_CASCADE1_2 bit of the WCO_WDT_CONFIG register. When cascaded, WDT2 increments after WDT1 reaches its match count.
- All the three timers are cascaded when WDT_CASCADE0_1 and WDT_CASCADE1_2 bits are set.

12.5 Register List

Table 12-1. WDT Registers

Register Name	Description
WDT_DISABLE_KEY	Disables the WDT when 0XACED8865 is written, for any other value WDT works normally
WDT_COUNTER	Provides the count value of the WDT
WDT_MATCH	Stores the match value of the WDT
SRSS_INTR	Services the WDT to avoid reset

Table 12-2. WDT Registers

Register Name	Description
WDT_DISABLE_KEY	Disables the WDT when 0XACED8865 is written; for any other value WDT works normally.
WDT_COUNTER	Provides the count value of the WDT.
WDT_MATCH	Holds the match value of the WDT.
SRSS_INTR	Services the WDT to avoid reset.
WCO_WDT_CTRLOW	Stores the current WDT0 and WDT1 timer value.
WCO_WDT_CTRHIGH	Stores the current WDT2 timer value.
WCO_WDT_MATCH	Holds the match count for WDT0 and WDT1.
WCO_WDT_CONFIG	Configures WDT0, WDT1, and WDT2 – selection of clock source, selection of free running or clear on match, interrupt generation, and cascading.
WCO_WDT_CONTROL	Used for enabling and resetting the timer.
WCO_WDT_CLKEN	Enables the clock (ILO/WCO) to be used with the timer.

13. Reset System

The PSoC® Analog Coprocessor supports several types of resets that guarantee error-free operation during power up and allow the device to reset based on user-supplied external hardware or internal software reset signals. PSoC Analog Coprocessor also contains hardware to enable the detection of certain resets.

The reset system has these sources:

- Power-on reset (POR) to hold the device in reset while the power supply ramps up
- Brownout reset (BOD) to reset the device if the power supply falls below specifications during operation
- Watchdog reset (WRES) to reset the device if firmware execution fails to service the watchdog timer
- Software initiated reset (SRES) to reset the device on demand using firmware
- External reset (XRES) to reset the device using an external electrical signal
- Protection fault reset (PROT_FAULT) to reset the device if unauthorized operating conditions occur

13.1 Reset Sources

The following sections provide a description of the reset sources available in the PSoC Analog Coprocessor.

13.1.1 Power-on Reset

Power-on reset is provided for system reset at power-up. POR holds the device in reset until the supply voltage, V_{DDD}, is according to the datasheet specification. The POR activates automatically at power-up.

POR events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, BOD, or XRES.

13.1.2 Brownout Reset

Brownout reset monitors the chip digital voltage supply V_{CCD} and generates a reset if V_{CCD} is below the minimum logic operating voltage specified in the device datasheet. BOD is available in all power modes.

13.1.3 Watchdog Reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. This feature is enabled by default. It can be disabled by writing '0xACED8865' to the WDT_DISABLE_KEY register.

The RESET_WDT status bit of the RES_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

For more details, see the Watchdog Timer chapter on page 92.

13.1.4 Software Initiated Reset

Software initiated reset (SRES) is a mechanism that allows a software-driven reset. The Cortex-M0+ application interrupt and reset control register (CM0P_AIRCR) forces a device reset when a ‘1’ is written into the SYSRESETREQ bit. CM0P_AIRCR requires a value of A05F written to the top two bytes for writes. Therefore, write A05F0004 for the reset.

The RESET_SOFT status bit of the RES_CAUSE register is set when a software reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.1.5 External Reset

External reset (XRES) is a user-supplied reset that causes immediate system reset when asserted. The XRES pin is ***active low*** – a high voltage on the pin has no effect and a low voltage causes a reset. The pin is pulled high inside the device. XRES is available as a dedicated pin in most of the devices. For detailed pinout, refer to the pinout section of the device datasheet.

The XRES pin holds the device in reset while held active. When the pin is released, the device goes through a normal boot sequence. The logical thresholds for XRES and other electrical characteristics, are listed in the Electrical Specifications section of the device datasheet.

XRES events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, BOD, or XRES.

13.1.6 Protection Fault Reset

Protection fault reset (PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code. For details about privilege code, see “Privileged” on page 87.

The RESET_PROT_FAULT bit of the RES_CAUSE register is set when a protection fault occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.2 Identifying Reset Sources

When the device comes out of reset, it is often useful to know the cause of the most recent or even older resets. This is achieved in the device primarily through the RES_CAUSE register. This register has specific status bits allocated for some of the reset sources. The RES_CAUSE register supports detection of watchdog reset, software reset, and protection fault reset. It does not record the occurrences of POR, BOD, or XRES. The bits are set on the occurrence of the corresponding reset and remain set after the reset, until cleared or a loss of retention, such as a POR reset, external reset, or brownout detect.

If the RES_CAUSE register cannot detect the cause of the reset, then it can be one of the non-recorded and non-retention resets: BOD, POR, XRES. These resets cannot be distinguished using on-chip resources.

13.3 Register List

Table 13-1. Reset System Register List

Register Name	Description
WDT_DISABLE_KEY	Disables the WDT when 0XACED8865 is written, for any other value WDT works normally
CM0P_AIRCR	Cortex-M0+ Application Interrupt and Reset Control Register - This register allows initiation of software resets, among other Cortex-M0+ functions.
RES_CAUSE	Reset Cause Register - This register captures the cause of recent resets.

14. Device Security

PSoC® Analog Coprocessor offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and enabling flash protection provide a high level of security.

The debug circuits are enabled by default and can only be disabled in firmware. If disabled, the only way to re-enable them is to erase the entire device, clear flash protection, and reprogram the device with new firmware that enables debugging. Additionally, all device interfaces can be permanently disabled for applications concerned about phishing attacks due to a maliciously reprogrammed device or attempts to defeat security by starting and interrupting flash programming sequences. Permanently disabling interfaces is not recommended for most applications because the designer cannot access the device. For more information, as well as a discussion on flash row and chip protection, see the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4100S, PSoC 4700S Programming Specifications.

Note Because all programming, debug, and test interfaces are disabled when maximum device security is enabled, PSoC Analog Coprocessor devices with full device security enabled may not be returned for failure analysis.

14.1 Features

The PSoC Analog Coprocessor device security system has the following features:

- User-selectable levels of protection.
- In the most secure case provided, the chip can be “locked” such that it cannot be acquired for test/debug and it cannot enter erase cycles. Interrupting erase cycles is a known way for hackers to leave chips in an undefined state and open to observation.
- CPU execution in a privileged mode by use of the non-maskable interrupt (NMI). When in privileged mode, NMI remains asserted to prevent any inadvertent return from interrupt instructions causing a security leak.

In addition to these, the device offers protection for individual flash row data.

14.2 How It Works

14.2.1 Device Security

The CPU operates in normal user mode or in privileged mode, and the device operates in one of four protection modes: BOOT, OPEN, PROTECTED, and KILL. Each mode provides specific capabilities for the CPU software and debug. You can change the mode by writing to the CPUSS_PROTECTION register.

- **BOOT mode**: The device comes out of reset in BOOT mode. It stays there until its protection state is copied from supervisor flash to the protection control register (CPUSS_PROTECTION). The debug-access port is stalled until this has happened. BOOT is a transitory mode required to set the part to its configured protection state. During BOOT mode, the CPU always operates in privileged mode.
- **OPEN mode**: This is the factory default. The CPU can operate in user mode or privileged mode. In user mode, flash can be programmed and debugger features are supported. In privileged mode, access restrictions are enforced.
- **PROTECTED mode**: The user may change the mode from OPEN to PROTECTED. This mode disables all debug access to user code or memory. In protected mode, only few registers are accessible; debug access to registers to reprogram flash is not available. The mode can be set back to OPEN but only after completely erasing the flash.
- **KILL mode**: The user may change the mode from OPEN to KILL. This mode removes all debug access to user code or memory, and the flash cannot be erased. Access to most registers is still available; debug access to registers to repro-

gram flash is not available. The part cannot be taken out of KILL mode; devices in KILL mode may not be returned for failure analysis.

14.2.2 Flash Security

The PSoC Analog Coprocessor devices include a flexible flash-protection system that controls access to flash memory. This feature is designed to secure proprietary code, but it can also be used to protect against inadvertent writes to the bootloader portion of flash.

Flash memory is organized in rows. You can assign one of two protection levels to each row; see Table 14-1. Flash protection levels can only be changed by performing a complete flash erase.

For more details, see the Nonvolatile Memory Programming chapter on page 241.

Table 14-1. Flash Protection Levels

Protection Setting	Allowed	Not Allowed
Unprotected	External read and write, Internal read and write	–
Full Protection	External read[a] Internal read	External write, Internal write

a. To protect the device from external read operations, you should change the device protection settings to PROTECTED.

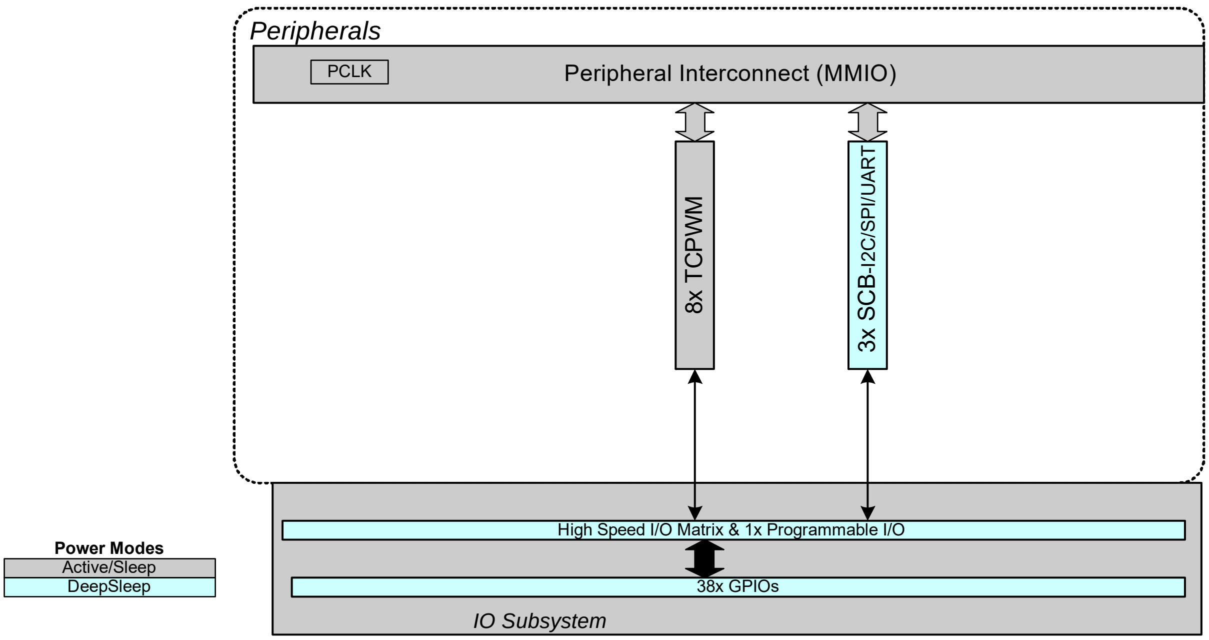
Section D: Digital System

This section encompasses the following chapters:

-
-

Top Level Architecture

Digital System Block Diagram



15. Serial Communications Block (SCB)

The Serial Communications Block (SCB) of PSoC® Analog Coprocessor supports three serial interface protocols: SPI, UART, and I^2C. Only one of the protocols is supported by an SCB at any given time. PSoC devices have three SCBs.

15.1 Features

This block supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, Local Interconnect Network (LIN), and IrDA protocols
- Standard I^2C master and slave functionality
- Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
- EZ mode for SPI and I^2C, which allows for operation without CPU intervention
- Low-power (Deep-Sleep) mode of operation for SPI and I^2C protocols (using external clocking)

Each of the three protocols is explained in the following sections.

15.2 Serial Peripheral Interface (SPI)

The SPI protocol is a synchronous serial interface protocol. Devices operate in either master or slave mode. The master initiates the data transfer. The SCB supports single-master-multiple-slaves topology for SPI. Multiple slaves are supported with individual slave select lines.

You can use the SPI master mode when the PSoC has to communicate with one or more SPI slave devices. The SPI slave mode can be used when the PSoC has to communicate with an SPI master device.

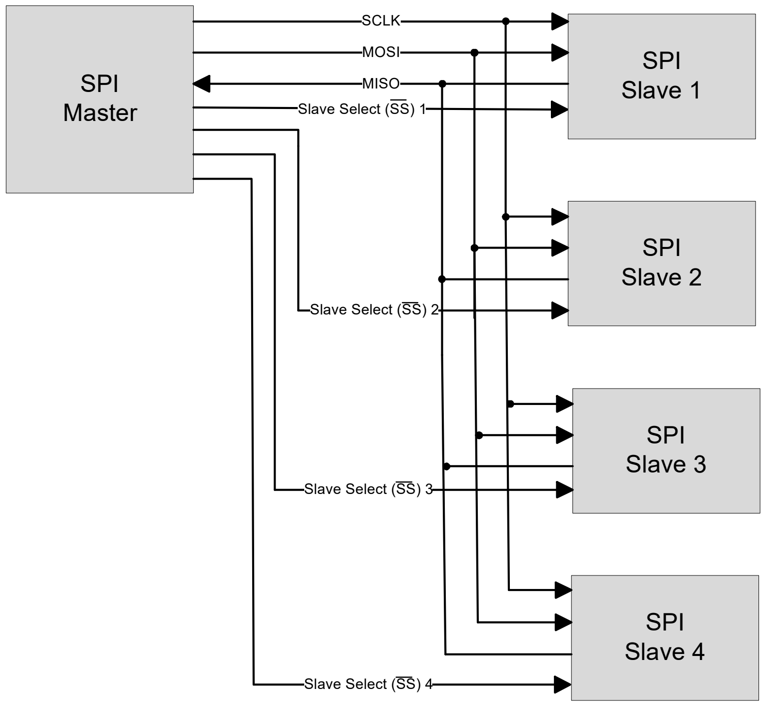
15.2.1 Features

- Supports master and slave functionality
- Supports three types of SPI protocols:
  - Motorola SPI – modes 0, 1, 2, and 3
  - Texas Instruments SPI, with coinciding and preceding data frame indicator for mode 1
  - National Semiconductor (MicroWire) SPI for mode 0
- Supports up to four slave select lines
- Data frame size programmable from 4 bits to 16 bits
- Interrupts or polling CPU interface
- Programmable oversampling
- Supports EZ mode of operation (Easy SPI Protocol)
  - EZSPI mode allows for operation without CPU intervention
- Supports externally clocked slave operation:
  - In this mode, the slave operates in Active, Sleep, and Deep-Sleep system power modes

15.2.2 General Description

Figure 15-1 illustrates an example of SPI master with four slaves.

Figure 15-1. SPI Example



A standard SPI interface consists of four signals as follows.

- SCLK: Serial clock (clock output from the master, input to the slave).
- MOSI: Master-out-slave-in (data output from the master, input to the slave).
- MISO: Master-in-slave-out (data input to the master, output from the slave).
- Slave Select ($\overline{SS}$): Typically an active low signal (output from the master, input to the slave).

A simple SPI data transfer involves the following: the master selects a slave by driving its $\overline{SS}$ line, then it drives data on the MOSI line and a clock on the SCLK line. The slave uses either of the edges of SCLK depending on the configuration to capture the data on the MOSI line; it also drives data on the MISO line, which is captured by the master.

By default, the SPI interface supports a data frame size of eight bits (1 byte). The data frame size can be configured to any value in the range 4 to 16 bits. The serial data can be transmitted either most significant bit (MSb) first or least significant bit (LSB) first.

Three different variants of the SPI protocol are supported by the SCB:

- Motorola SPI: This is the original SPI protocol.
- Texas Instruments SPI: A variation of the original SPI protocol, in which data frames are identified by a pulse on the $\overline{SS}$ line.
- National Semiconductors SPI: A half duplex variation of the original SPI protocol.

15.2.3 SPI Modes of Operation

15.2.3.1 Motorola SPI

The original SPI protocol was defined by Motorola. It is a full duplex protocol. Multiple data transfers may happen with the $\overline{SS}$ line held at '0'. As a result, slave devices must keep track of the progress of data transfers to separate individual data frames. When not transmitting data, the $\overline{SS}$ line is held at '1' and SCLK is typically pulled low.

Modes of Motorola SPI

The Motorola SPI protocol has four different modes based on how data is driven and captured on the MOSI and MISO lines. These modes are determined by clock polarity (CPOL) and clock phase (CPHA).

Clock polarity determines the value of the SCLK line when not transmitting data. CPOL = '0' indicates that SCLK is '0' when not transmitting data. CPOL = '1' indicates that SCLK is '1' when not transmitting data.

Clock phase determines when data is driven and captured. CPHA=0 means sample (capture data) on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. With CPHA=0, the data must be stable for setup time before the first clock cycle.

- Mode 0: CPOL is '0', CPHA is '0': Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.
- Mode 1; CPOL is '0', CPHA is '1': Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 2: CPOL is '1', CPHA is '0': Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 3: CPOL is '1', CPHA is '1': Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.

Figure 15-2 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

Figure 15-2. SPI Motorola, 4 Modes

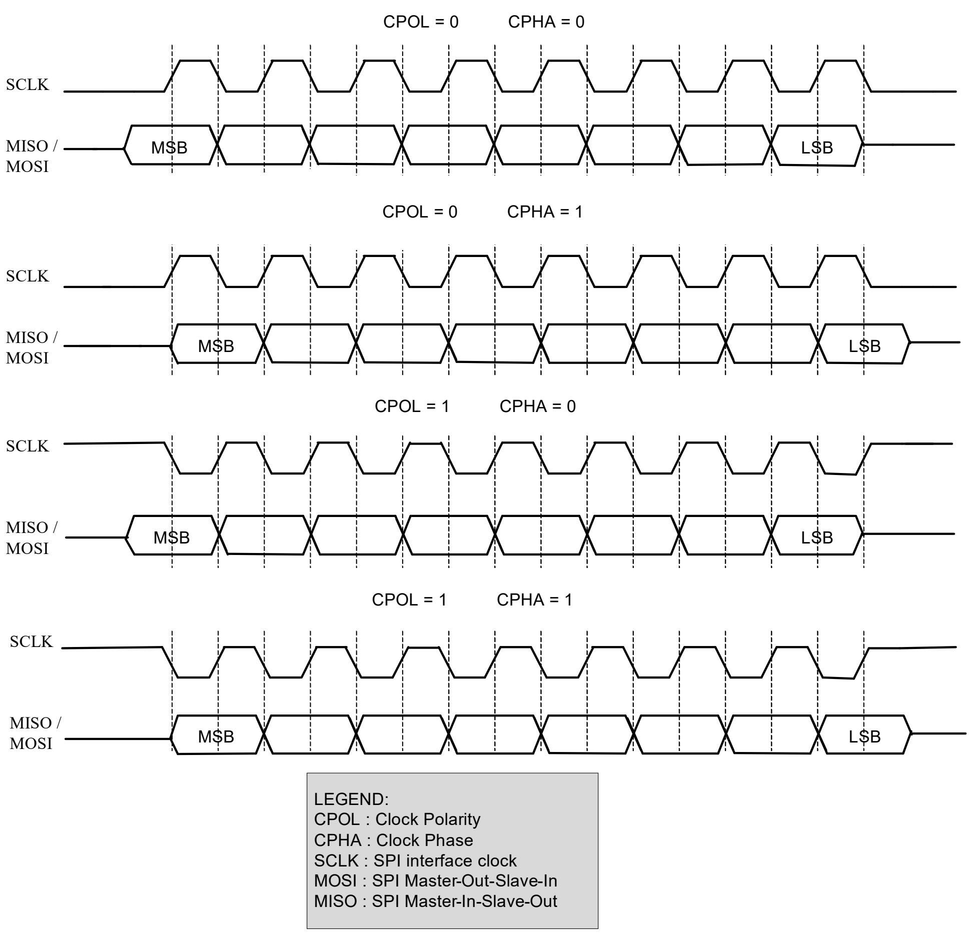
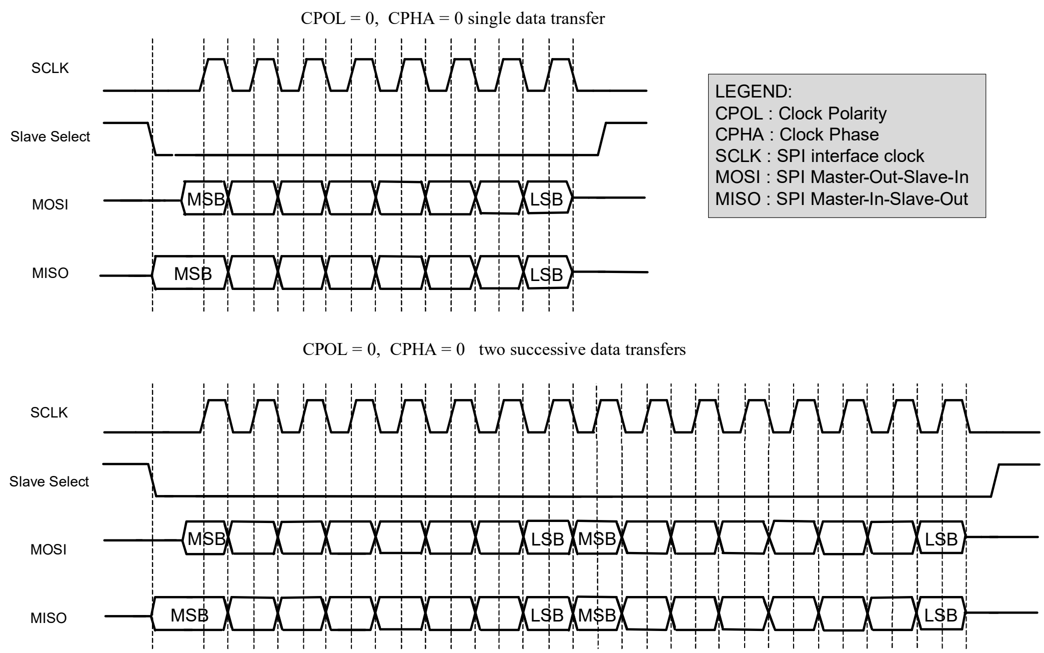


Figure 15-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 15-3. SPI Motorola Data Transfer Example



Configuring SCB for SPI Motorola Mode

To configure the SCB for SPI Motorola mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI Motorola mode by writing '00' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the CPHA and CPOL fields (bits 2 and 3 respectively) of the SCB_SPI_CTRL register.
4. Follow steps 2 to 4 mentioned in "Enabling and Initializing SPI" on page 111.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

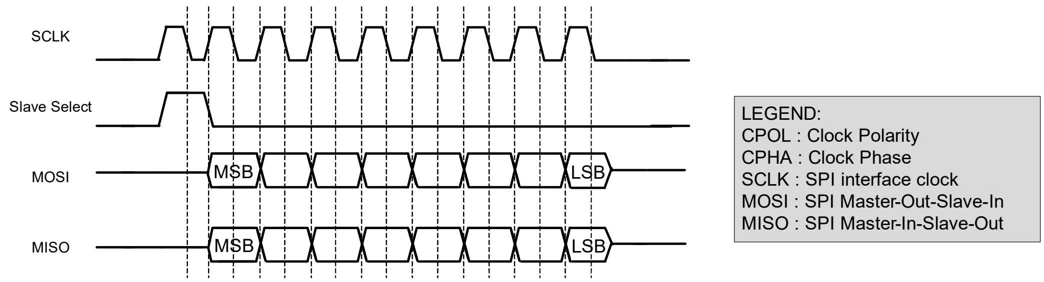
15.2.3.2 Texas Instruments SPI

The Texas Instruments' SPI protocol redefines the use of the $\overline{SS}$ signal. It uses the signal to indicate the start of a data transfer, rather than a low active slave select signal, as in the case of Motorola SPI. As a result, slave devices need not keep track of the progress of data transfers to separate individual data frames. The start of a transfer is indicated by a high active pulse of a single bit transfer period. This pulse may occur one cycle before the transmission of the first data bit, or may coincide with the transmission of the first data bit. The TI SPI protocol supports only mode 1 (CPOL is '0' and CPHA is '1'): data is driven on a rising edge of SCLK and data is captured on a falling edge of SCLK.

Figure 15-4 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse precedes the first data bit. Note how the SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

Figure 15-4. SPI TI Data Transfer Example

CPOL=0, CPHA=1 single data transfer



CPOL=0, CPHA=1 two successive data transfers

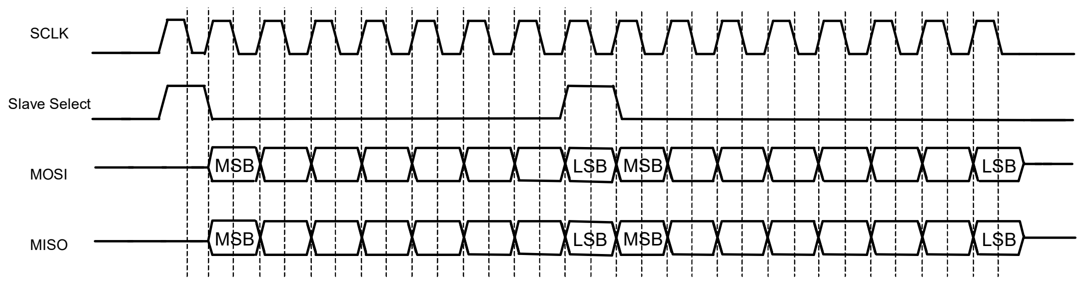
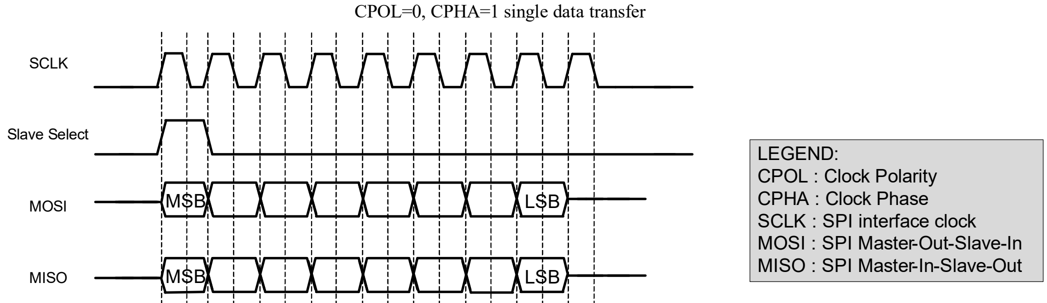


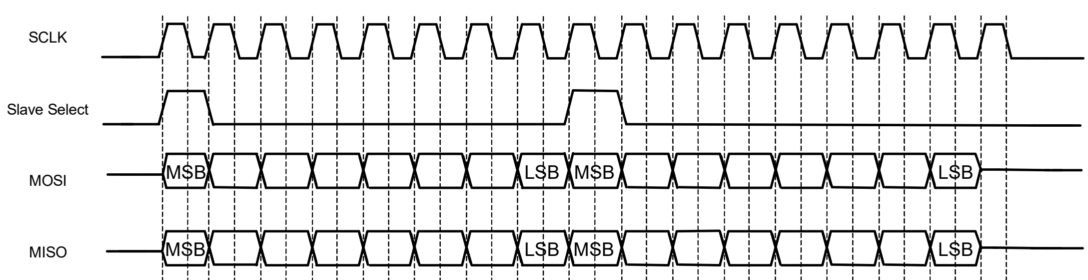
Figure 15-5 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

Figure 15-5. SPI TI Data Transfer Example

CPOL=0, CPHA=1 single data transfer



CPOL=0, CPHA=1 two successive data transfers



Configuring SCB for SPI TI Mode

To configure the SCB for SPI TI mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI TI mode by writing '01' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in TI by writing to the SELECT_PRECEDE field (bit 1) of the SCB_SPI_CTRL register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 5 mentioned in "Enabling and Initializing SPI" on page 111.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

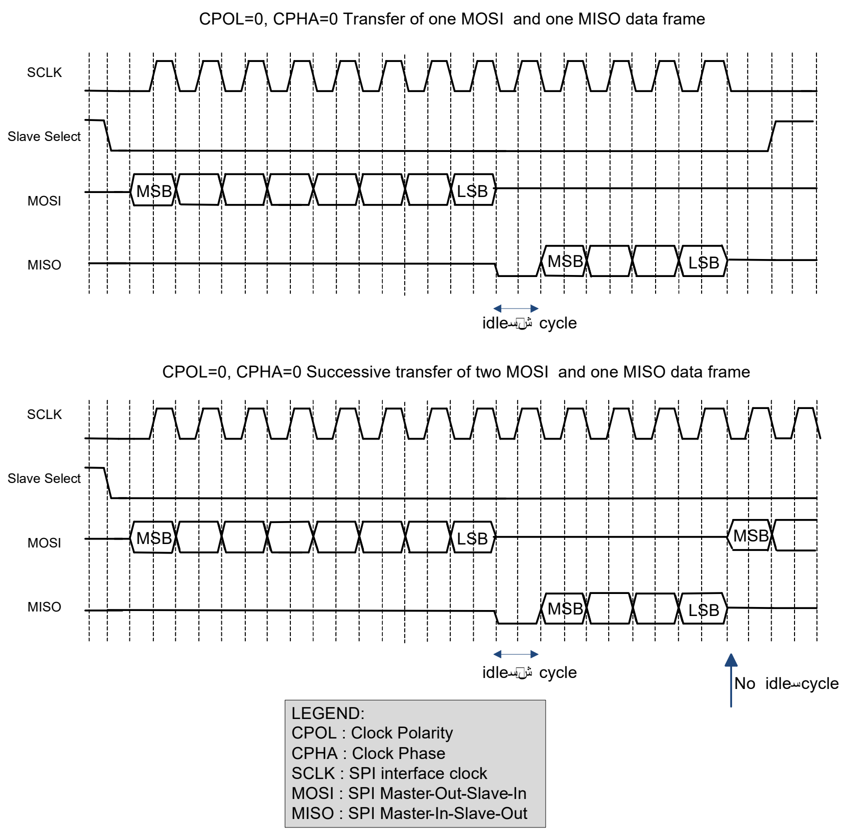
15.2.3.3 National Semiconductors SPI

The National Semiconductors' SPI protocol is a half duplex protocol. Rather than transmission and reception occurring at the same time, they take turns. The transmission and reception data sizes may differ. A single "idle" bit transfer period separates transmission from reception. However, the successive data transfers are NOT separated by an "idle" bit transfer period.

The National Semiconductors SPI protocol only supports mode 0: data is driven on a falling edge of SCLK and data is captured on a rising edge of SCLK.

Figure 15-6 illustrates a single data transfer and two successive data transfers. In both cases the transmission data transfer size is eight bits and the reception data transfer size is four bits.

Figure 15-6. SPI NS Data Transfer Example



Configuring SCB for SPI NS Mode

To configure the SCB for SPI NS mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in “Enabling and Initializing SPI” on page 111.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

15.2.4 Using SPI Master to Clock Slave

In a normal SPI Master mode transmission, the SCLK is generated only when the SCB is enabled and data is being transmitted. This can be changed to always generate a clock on the SCLK line as long as the SCB is enabled. This is used when the slave uses the SCLK for functional operations other than just the SPI functionality. To enable this, write '1' to the SCLK_CONTINUOUS (bit 5) of the SCB_SPI_CTRL register.

15.2.5 Easy SPI Protocol

The easy SPI (EZSPI) protocol is based on the Motorola SPI operating in any mode (0, 1, 2, 3). It allows communication between master and slave without the need for CPU intervention at the level of individual frames.

The EZSPI protocol defines an 8-bit EZ address that indexes a memory array (32-entry array of eight bit per entry is supported) located on the slave device. To address these 32 locations, the lower five bits of the EZ address are used. All EZSPI data transfers have 8-bit data frames.

Note The SCB has a FIFO memory, which is a 16 word by 16-bit SRAM, with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has eight entries of 16 bits per entry. The 16-bit width per entry is used to accommodate configurable data width. In EZ mode, it is used as a single 32x8 bit EZFIFO because only a fixed 8-bit width data is used in EZ mode.

EZSPI has three types of transfers: a write of the EZ address from the master to the slave, a write of data from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

15.2.5.1 EZ Address Write

A write of the EZ address starts with a command byte (0x00) on the MOSI line indicating the master's intent to write the EZ address. The slave then drives a reply byte on the MISO line to indicate that the command is observed (0xFE) or not (0xFF). The second byte on the MOSI line is the EZ address.

15.2.5.2 Memory Array Write

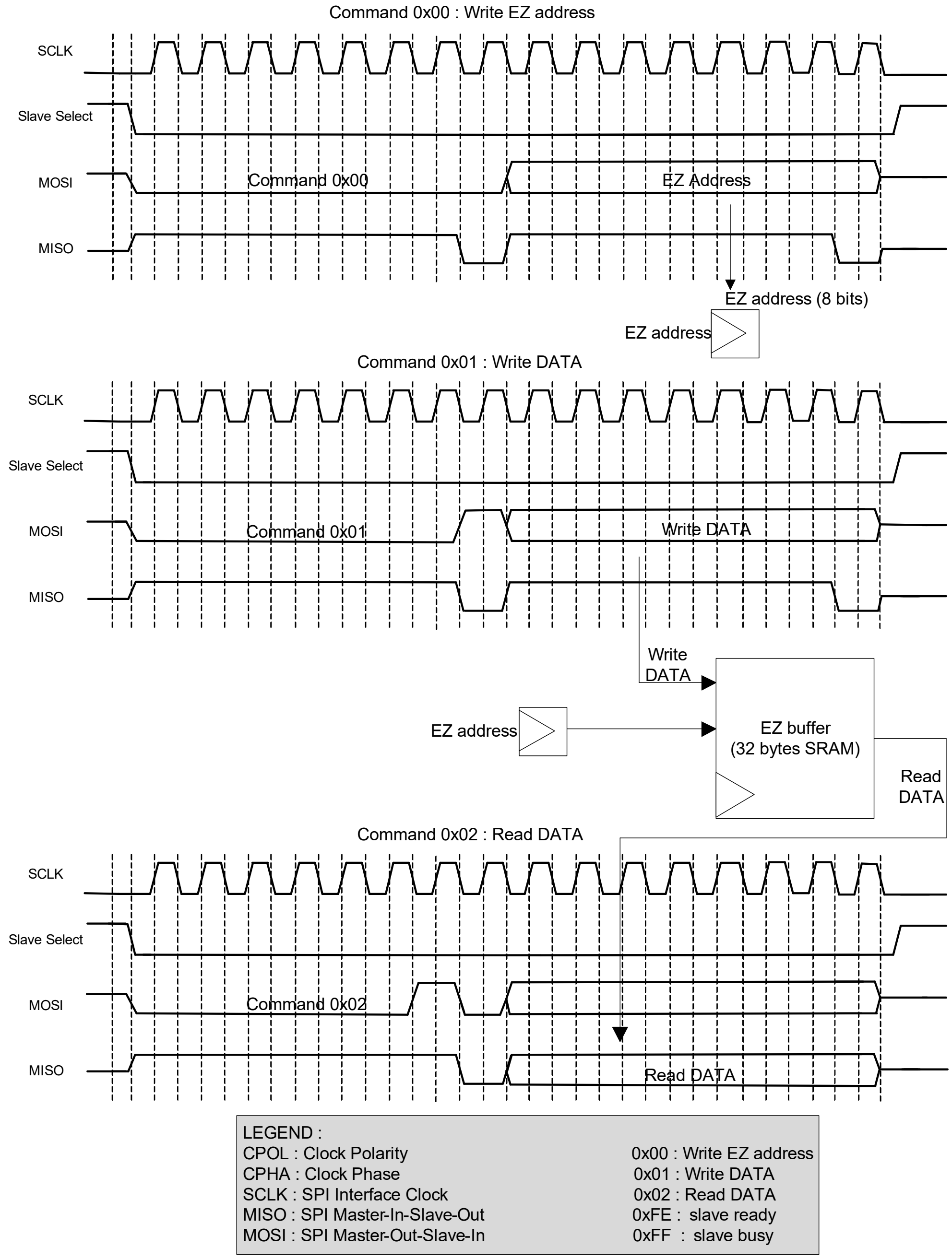
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional write data bytes on the MOSI line are written to the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are written into the memory array. When the EZ address exceeds the maximum number of memory entries (32), it remains there and does not wrap around to 0.

15.2.5.3 Memory Array Read

A read from a memory array index starts with a command byte (0x02) on the MOSI line indicating the master's intent to read from the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional read data bytes on the MISO line are read from the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are read from the memory array. When the EZ address exceeds the maximum number of memory entries (32), it remains there and does not wrap around to 0.

Figure 15-7 illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.

Figure 15-7. EZSPI Example



15.2.5.4 Configuring SCB for EZSPI Mode

By default, the SCB is configured for non-EZ mode of operation. To configure the SCB for EZSPI mode, set the register bits in the following order:

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Use continuous transmission mode for the transmitter by writing '1' to the CONTINUOUS bit of SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in “Enabling and Initializing SPI” on page 111.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

15.2.6 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 15-1. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

Table 15-1. SPI Registers

Register Name	Operation
SCB_CTRL	Enables the SCB, selects the type of serial interface (SPI, UART, I^2C), and selects internally and externally clocked operation, EZ and non-EZ modes of operation.
SCB_STATUS	In EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory.
SCB_SPI_CTRL	Configures the SPI as either a master or a slave, selects SPI protocols (Motorola, TI, National) and clock-based submodes in Motorola SPI (modes 0,1,2,3), selects the type of SELECT signal in TI SPI.
SCB_SPI_STATUS	Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode.
SCB_TX_CTRL	Specifies the data frame width and specifies whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_FIFO_RD	Holds the data frame read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO - behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data frame read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Holds the slave device address and mask values.
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_EZ_DATA	Holds the data in EZ memory location

15.2.7 SPI Interrupts

The SPI supports both internal and external interrupt requests. The internal interrupt events are listed here. PSoC Creator generates the necessary interrupt service routines (ISRs) for handling buffer management interrupts. Custom ISRs can also be used by connecting external interrupt component to the interrupt output of the SPI component (with external interrupts enabled).

The SPI predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled Rx interrupt sources are true. Various interrupt registers are used to determine the actual source of the interrupt.

The SPI supports interrupts on the following events:

- SPI master transfer done
- SPI Bus Error - Slave deselected at an unexpected time in the SPI transfer
- SPI slave deselected after any EZSPI transfer occurred
- SPI slave deselected after a write EZSPI transfer occurred
- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
- RX
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
- SPI Externally clocked
  - Wake up request on slave select
  - SPI STOP detection at the end of each transfer
  - SPI STOP detection at the end of a write transfer
  - SPI STOP detection at the end of a read transfer

Note The SPI interrupt signal is hard-wired to the Cortex-M0 NVIC and cannot be routed to external pins.

15.2.8 Enabling and Initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCB_SPI_CTRL register, according to Table 15-3. This includes selecting the submodes of the protocol and selecting master-slave functionality. EZSPI can be used with slave mode only.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-4:
   a. Specify the data frame width. This should always be 8 for EZSPI.
   b. Specify whether MSB or LSB is the first bit to be transmitted/received. This should always be MSB first for EZSPI.
3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in Table 15-5:
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFO.
4. Program SCB_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in Table 15-2.

5. Enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally clocked to internally clocked operation. The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

Table 15-2. SCB_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I^2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

Table 15-3. SCB_SPI_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	SPI Motorola submode. (This is the only mode supported for EZSPI.)
		01	SPI Texas Instruments submode.
		10	SPI National Semiconductors submode.
		11	Reserved.
31	MASTER_MODE	0	Slave mode. (This is the only mode supported for EZSPI.)
		1	Master mode.

Table 15-4. SCB_TX_CTRL/SCB_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits. For EZSPI, this value should be '0b0111'
8	MSB_FIRST	1= MSB first 0= LSB firstFor EZSPI, this value should be 1.
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. 1=Enabled 0=Disabled

Table 15-5. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[7:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

15.2.9 Internally and Externally Clocked SPI Operations

The SCB supports both internally and externally clocked operations for SPI and I^2C functions. An internally clocked operation uses a clock provided by the chip. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation enables operation in the Deep-Sleep system power mode.

Internally clocked operation uses the high-frequency clock (HFCLK) of the system. For more information on system clocking, see the Clocking System chapter on page 76. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

In SPI master mode, the valid range for oversampling is 4 to 16. Hence, with a clock speed of 48 MHz, the maximum bit rate is 12 Mbps. However, if you consider the I/O cell and routing delays, the oversampling must be set between 6 and 16 for proper operation. So, the maximum bit rate is 8 Mbps. **Note** To achieve maximum possible bit rate, LATE_MISO_SAMPLE must be set to '1' in SPI master mode. This has a default value of ‘0’.

In SPI slave mode, the OVS field (bits [3:0]) of SCB_CTRL register is not used. However, there is a frequency requirement for the SCB clock with respect to the interface clock (SCLK). This requirement is expressed in terms of the ratio (SCB clock/SCLK). This ratio is dependent on two fields: MEDIAN of SCB_RX_CTRL register and LATE_MISO_SAMPLE of SCB_CTRL register. If the external SPI master supports Late MISO sampling and if the median bit is set to ‘0’, the maximum data rate that can be achieved is 16 Mbps. If the external SPI master does not support late MISO sampling, the maximum data rate is limited to 8 Mbps (with the median bit set to ‘0’). Based on these bits, the maximum bit rates are given in Table 15-6.

Table 15-6. SPI Slave Maximum Data Rates

Maximum Bit Rate at Peripheral Clock of 48 MHz	Ratio Requirement	Median of SCB_RX_CTRL	LATE_MISO_SAMPLE of SCB_CTRL
8 Mbps	≥6	0	1
6 Mbps	≥8	1	1
4 Mbps	≥12	0	0
3 Mbps	≥16	1	0

Externally clocked operation is limited to:

- Slave functionality.
- EZ functionality. EZ functionality uses the block's SRAM as a memory structure. Non-EZ functionality uses the block's SRAM as TX and RX FIFOs; FIFO support is not available in externally clocked operation.
- Motorola mode 0, 1, 2, 3.

Externally clocked EZ mode of operation can support a data rate of 48 Mbps (at the interface clock of 48 MHz).

Internally and externally clocked operation is determined by two register fields of the SCB_CTRL register:

- **EC_AM_MODE:** Indicates whether SPI slave selection is internally ('0') or externally ('1') clocked. SPI slave selection comprises the first part of the protocol.
- **EC_OP_MODE:** Indicates whether the rest of the protocol operation (besides SPI slave selection) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does NOT support non-EZ functionality.

These two register fields determine the functional behavior of SPI. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. Table 15-7 and Table 15-8 describe the settings for SPI (in non-EZ and EZ modes).

15.2.9.1 Non-EZ Mode of Operation

In non-EZ mode there are two possible settings. As externally clocked operation is not supported for non-EZ functionality (no FIFO support), EC_OP_MODE should always be set to '0'. However, EC_AM_MODE can be set to '0' or '1'. Table 15-7 gives an overview of the possibilities.

Table 15-7. SPI Operation in Non-EZ Mode

<table>
<tr><th colspan="5">SPI (non-EZ)Mode</th></tr>
<tr><th rowspan="2">System Power Mode</th><th colspan="2">EC_OP_MODE = 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th></tr>
<tr><td>Active and Sleep</td><td>Selection using internal clock.
Operation using internal clock.</td><td>Selection using external clock:
Operation using internal clock.
In Active mode, the Wakeup interrupt cause is disabled (MASK = 0).
In Sleep mode, the MASK bit can be configured by the user.</td><td rowspan="2">Not supported</td><td rowspan="2">Not supported</td></tr>
<tr><td>Deep-Sleep</td><td>Not supported</td><td>Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1).
Send 0xFF.</td></tr>
</table>

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power mode and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by the externally clocked logic: in Active system power mode, both internally and externally clocked logic are active, and in Deep-Sleep system power mode, only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active system power mode, the CPU and the block's internally clocked operation are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in the Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked operation takes care of the ongoing SPI transfer.
- In Deep-Sleep system power mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged ('1' bit or "0xFF" byte is sent out on the MISO line) and the internally clocked operation takes care of the next SPI transfer when it is woken up.

15.2.9.2 EZ Mode of Operation

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-8 gives an overview of the possibilities. The grey cells indicate a possible, yet not recommended, setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

Table 15-8. SPI Operation in EZ Mode

<table>
<tr><th colspan="5">SPI, EZ Mode</th></tr>
<tr><th rowspan="2">System Power Mode</th><th colspan="2">EC_OP_MODE = 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th></tr>
<tr><td>Active and Sleep</td><td>Selection using internal clock.
Operation using internal clock.</td><td>Selection using external clock.
Operation using internal clock.
In Active mode, the Wakeup interrupt cause is disabled (MASK = 0).
In Sleep mode, the MASK bit can be configured by the user.</td><td rowspan="2">Invalid</td><td>Selection using external clock.
Operation using external clock.</td></tr>
<tr><td>Deep-Sleep</td><td>Not supported</td><td>Selection using external clock:
Wakeup interrupt cause is enabled (MASK = 1).
Send 0xFF.</td><td>Selection using external clock.
Operation using external clock.</td></tr>
</table>

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power modes and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by the externally clocked logic: in Active system power mode, both internally and externally clocked logic are active, and in Deep-Sleep system power mode, only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active system power mode, the CPU and the block's internally clocked operation are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked operation takes care of the ongoing SPI transfer.
- In Deep-Sleep system power mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged ('1' bit or "0xFF" byte is sent out on the MISO line) and the internally clocked operation takes care of the next SPI transfer when it is woken up.

EC_OP_MODE is '1' and EC_AM_MODE is '1': This setting works in Active, Sleep, and Deep-Sleep system power modes. The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK of the SCB_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

15.3 UART

The Universal Asynchronous Receiver/Transmitter (UART) protocol is an asynchronous serial interface protocol. UART communication is typically point-to-point. The UART interface consists of two signals:

- TX: Transmitter output
- RX: Receiver input

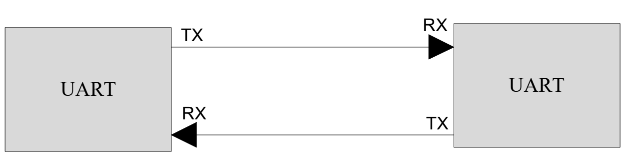
15.3.1 Features

- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 3 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports Local Interconnect Network (LIN)
  - Break detection
  - Baud rate detection
  - Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus)
- Multi-processor mode
- Data frame size programmable from 4 to 9 bits
- Programmable number of STOP bits, which can be set in terms of half bit periods between 1 and 4
- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

15.3.2 General Description

Figure 15-8 illustrates a standard UART TX and RX.

Figure 15-8. UART Example



A typical UART transfer consists of a "Start Bit" followed by multiple "Data Bits", optionally followed by a "Parity Bit" and finally completed by one or more "Stop Bits". The Start and Stop bits indicate the start and end of data transmission. The Parity bit is sent by the transmitter and is used by the receiver to detect single bit errors. As the interface does not have a clock (asynchronous), the transmitter and receiver use their own clocks; also, they need to agree upon the period of a bit transfer.

Three different serial interface protocols are supported:

- Standard UART protocol
  - Multi-Processor Mode
  - Local Interconnect Network (LIN)
- SmartCard, similar to UART, but with a possibility to send a negative acknowledgement
- IrDA, modification to the UART with a modulation scheme

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 4. The parity bit can be either enabled or disabled. If enabled, the type of parity can be set to either even parity or odd parity. The option of using the parity bit is available only in the Standard UART and SmartCard UART modes. For IrDA UART mode, the parity bit is automatically disabled. Figure 15-9 depicts the default configuration of the UART interface of the SCB.

Note UART interface does not support external clocking operation. Hence, UART operates only in the Active and Sleep system power modes.

15.3.3 UART Modes of Operation

15.3.3.1 Standard Protocol

A typical UART transfer consists of a start bit followed by multiple data bits, optionally followed by a parity bit and finally completed by one or more stop bits. The start bit value is always '0', the data bits values are dependent on the data transferred, the parity bit value is set to a value guaranteeing an even or odd parity over the data bits, and the stop bit value is '1'. The parity bit is generated by the transmitter and can be used by the receiver to detect single bit transmission errors. When not transmitting data, the TX line is '1' – the same value as the stop bits.

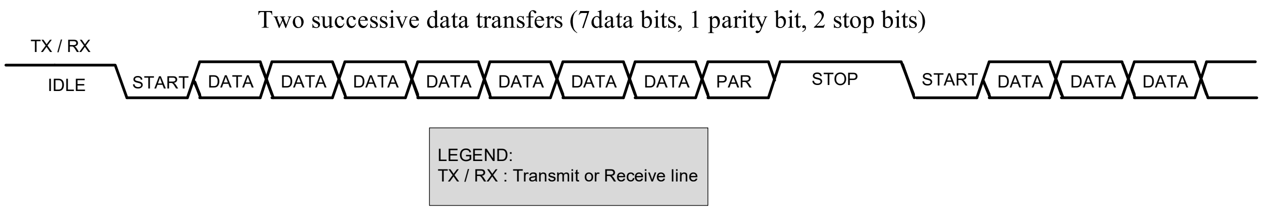
Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample the incoming signal.

The transition of a stop bit to a start bit is represented by a change from '1' to '0' on the TX line. This transition can be used by the receiver to synchronize with the transmitter clock. Synchronization at the start of each data transfer allows error-free transmission even in the presence of frequency drift between transmitter and receiver clocks. The required clock accuracy is dependent on the data transfer size.

The stop period or the amount of stop bits between successive data transfers is typically agreed upon between transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods.

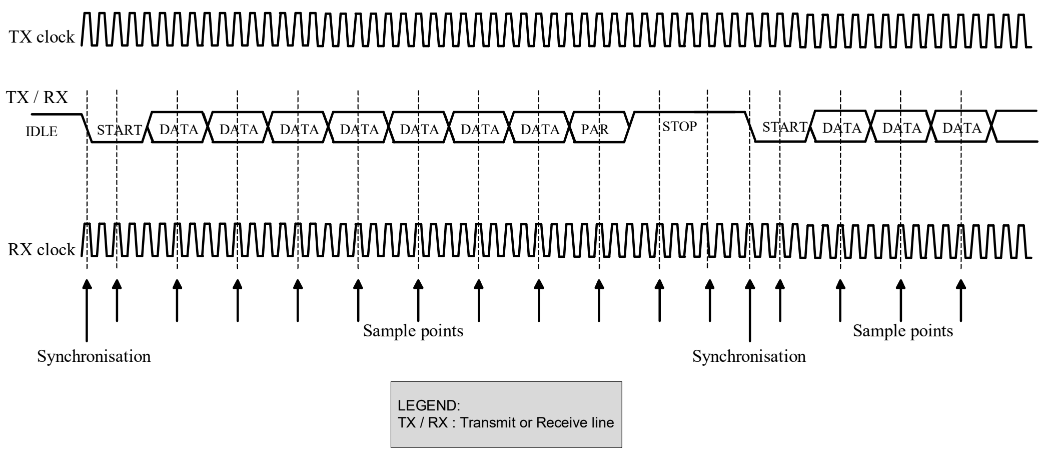
Figure 15-9 illustrates the UART protocol.

Figure 15-9. UART, Standard Protocol Example



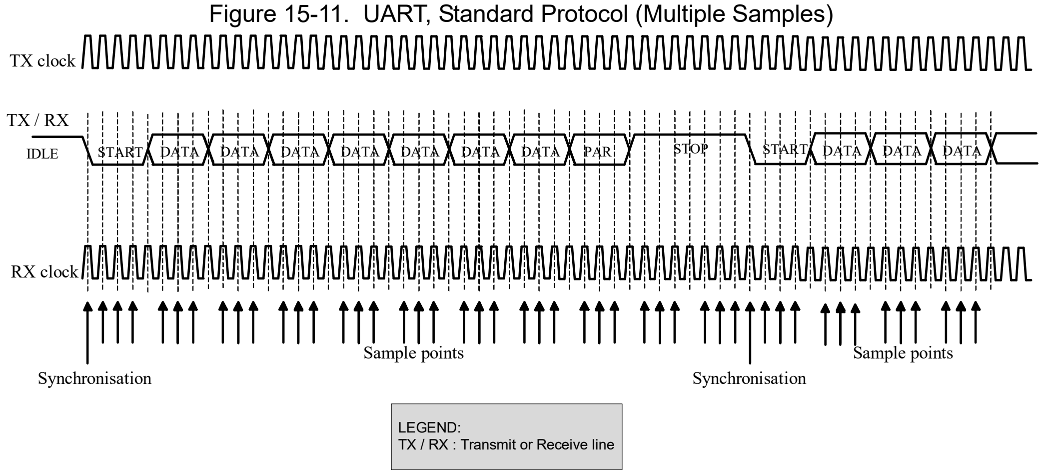
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. Figure 15-10 illustrates this.

Figure 15-10. UART, Standard Protocol Example (Single Sample)



Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to increase accuracy. Figure 15-11 illustrates this.

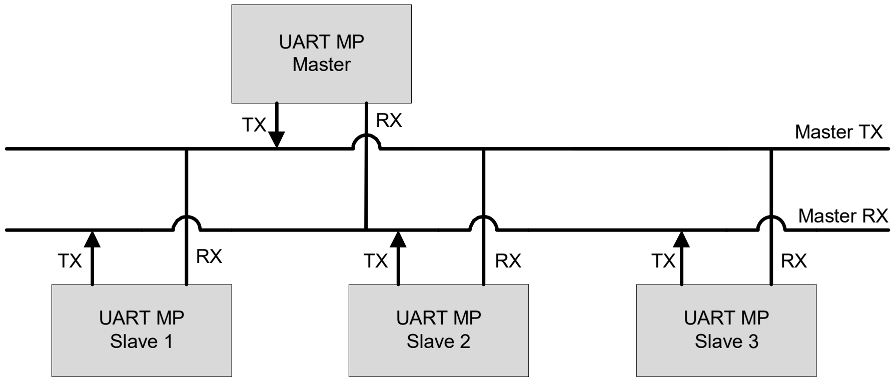
Figure 15-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor Mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as Figure 15-12 shows. This mode is also known as UART 9-bit protocol because the data field is nine bits wide. UART_MP is part of Standard UART mode.

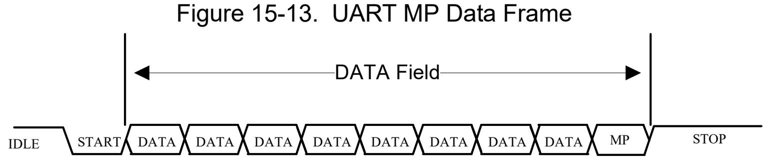
Figure 15-12. UART MP Mode Bus Connections



The main properties of UART_MP mode are:

- Single master with multiple slave concept (multi-drop network).
- Each slave is identified by a unique address.
- Using 9-bit data field, with the ninth bit as address/data flag (MP bit). When set high, it indicates an address byte; when set low it indicates a data byte. A data frame is illustrated in Figure 15-13.
- Parity bit is disabled.

Figure 15-13. UART MP Data Frame

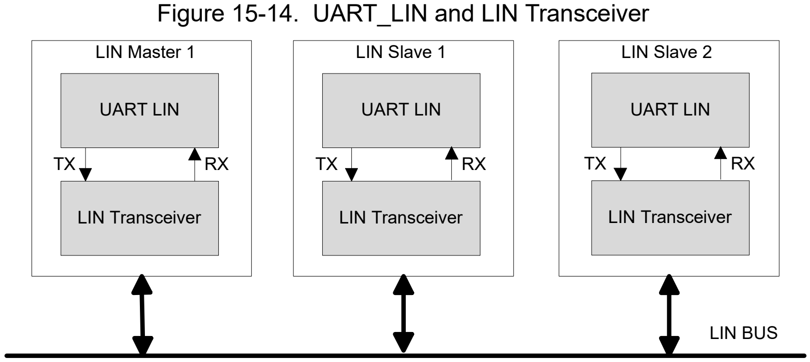


The SCB can be used as either master or slave device in UART_MP mode. Both SCB_TX_CTRL and SCB_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device, the MP_MODE field of the SCB_UART_RX_CTRL register should be set to '1'. The SCB_RX_MATCH register should be set for the slave address and address mask. The matched address is written in the RX_FIFO when ADDR_ACCEPT field of the SCB_CTRL register is set to '1'. If received address does not match its own address, then the interface ignores the following data, until next address is received for compare.

UART Local Interconnect Network (LIN) Mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with single-master-multi-slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports both LIN master and slave functionality. The LIN specification defines both physical layer (layer 1) and data link layer (layer 2). Figure 15-14 illustrates the UART_LIN and LIN Transceiver.

Figure 15-14. UART_LIN and LIN Transceiver

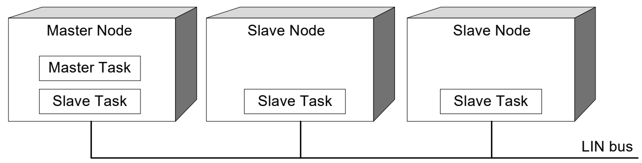


LIN protocol defines two tasks:

- Master task: This task involves sending a header packet to initiate a LIN transfer.
- Slave task: This task involves transmitting or receiving a response.

The master node supports master task and slave task; the slave node supports only slave task, as shown in Figure 15-15.

Figure 15-15. LIN Bus Nodes and Tasks

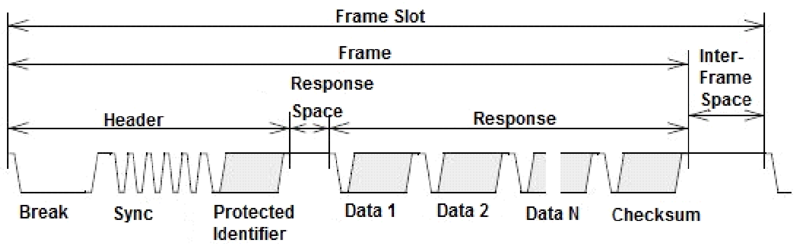


LIN Frame Structure

LIN is based on the transmission of frames at pre-determined moments of time. A frame is divided into header and response fields, as shown in Figure 15-16.

- The header field consists of:
  - Break field (at least 13 bit periods with the value '0').
  - Sync field (a 0x55 byte frame). A sync field can be used to synchronize the clock of the slave task with that of the master task.
  - Identifier field (a frame specifying a specific slave).
- The response field consists of data and checksum.

Figure 15-16. LIN Frame Structure

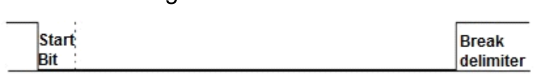


In LIN protocol communication, the least significant bit (LSB) of the data is sent first and the most significant bit (MSB) last. The start bit is encoded as zero and the stop bit is encoded as one. The following sections describe all the byte fields in the LIN frame.

Break Field

Every new frame starts with a break field, which is always generated by the master. The break filed has logical zero with a minimum of 13 bit times and followed by a break delimiter. The break field structure is as shown in Figure 15-17.

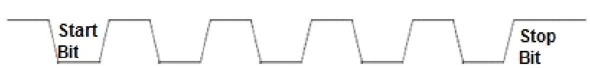
Figure 15-17. LIN Break Field



Sync Field

This is the second field transmitted by the master in the header field; its value is 0x55. A sync field can be used to synchronize the clock of the slave task with that of the master task for automatic baud rate detection. Figure 15-18 shows the LIN sync field structure.

Figure 15-18. LIN Sync Field



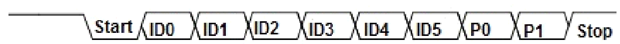
Protected identifier (PID) Field

A protected identifier field consists of two sub-fields: the frame identifier (bits 0-5) and the parity (bit 6 and bit 7). The PID field structure is shown in Figure 15-19.

- Frame identifier: The frame identifiers are split into three categories
  - Values 0 to 59 (0x3B) are used for signal carrying frames
  - 60 (0x3C) and 61 (0x3D) are used to carry diagnostic and configuration data
  - 62 (0x3E) and 63 (0x3F) are reserved for future protocol enhancements
- Parity: Frame identifier bits are used to calculate the parity

Figure 15-19 shows the PID field structure.

Figure 15-19. PID Field



Data. In LIN, every frame can carry a minimum of one byte and maximum of 8 bytes of data. Here, the LSB of the data byte is sent first and the MSB of the data byte is sent last.

Checksum

The checksum is the last byte field in the LIN frame. It is calculated by inverting the 8-bit sum along with carryover of all data bytes only or the 8-bit sum with the carryover of all data bytes and the PID field. There are two types of checksums in LIN frames. They are:

- Classic checksum: the checksum calculated over all the data bytes only (used in LIN 1.x slaves).
- Enhanced checksum: the checksum calculated over all the data bytes along with the protected identifier (used in LIN 2.x slaves).

LIN Frame Types

The type of frame refers to the conditions that need to be valid to transmit the frame. According to the LIN specification, there are five different types of LIN frames. A node or cluster does not have to support all frame types.

Unconditional Frame

These frames carry the signals and their frame identifiers (of 0x00 to 0x3B range). The subscriber will receive the frames and make it available to the application; the publisher of the frame will provide the response to the header.

Event-Triggered Frame

The purpose of an event-triggered frame is to increase the responsiveness of the LIN cluster without assigning too much of the bus bandwidth to polling of multiple slave nodes with seldom occurring events. Event-triggered frames carry the response of one or more unconditional frames. The unconditional frames associated with an event triggered frame should:

- Have equal length
- Use the same checksum model (either classic or enhanced)
- Reserve the first data field to its protected identifier
- Be published by different slave nodes
- Not be included directly in the same schedule table as the event-triggered frame

Sporadic Frame

The purpose of the sporadic frames is to merge some dynamic behavior into the schedule table without affecting the rest of the schedule table. These frames have a group of unconditional frames that share the frame slot. When the sporadic frame is due for transmission, the unconditional frames are checked if they have any updated signals. If no signals are updated, no frame will be transmitted and the frame slot will be empty.

Diagnostic Frames

Diagnostic frames always carry transport layer, and contains eight data bytes.

The frame identifier for diagnostic frame is:

- Master request frame (0x3C), or
- Slave response frame (0x3D)

Before transmitting a master request frame, the master task queries its diagnostic module to see if it will be transmitted or if the bus will be silent. A slave response frame header will be sent unconditionally. The slave tasks publish and subscribe to the response according to their diagnostic modules.

Reserved Frames

These frames are reserved for future use; their frame identifiers are 0x3E and 0x3F.

LIN Go-To-Sleep and Wake-Up

The LIN protocol has the feature of keeping the LIN bus in Sleep mode, if the master sends the go-to-sleep command. The go-to-sleep command is a master request frame (ID = 0x3C) with the first byte field is equal to 0x00 and rest set to 0xFF. The slave node application may still be active after the go-to-sleep command is received. This behavior is application specific. The LIN slave nodes automatically enter Sleep mode if the LIN bus inactivity is more than four seconds.

Wake-up can be initiated by any node connected to the LIN bus – either LIN master or any of the LIN slaves by forcing the bus to be dominant for 250 µs to 5 ms. Each slave should detect the wakeup request and be ready to process headers within 100 ms. The master should also detect the wakeup request and start sending headers when the slave nodes are active.

To support LIN, a dedicated (off-chip) line driver/receiver is required. Supply voltage range on the LIN bus is 7 V to 18 V. Typically, LIN line drivers will drive the LIN line with the value provided on the SCB TX line and present the value on the LIN line to the SCB RX line. By comparing TX and RX lines in the SCB, bus collisions can be detected (indicated by the SCB_UART_ARB_LOST field of the SCB_INTR_TX register).

Configuring the SCB as Standard UART Interface

To configure the SCB as a standard UART interface, set various register bits in the following order:

1. Configure the SCB as UART interface by writing '10' to the MODE field (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCB_UART_CTRL register.
3. To enable the UART MP Mode or UART LIN Mode, write '1' to the MP_MODE (bit 10) or LIN_MODE (bit 12) respectively of the SCB_UART_RX_CTRL register.
4. Follow steps 2 to 5 described in “Enabling and Initializing UART” on page 124.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

15.3.3.2 SmartCard (ISO7816)

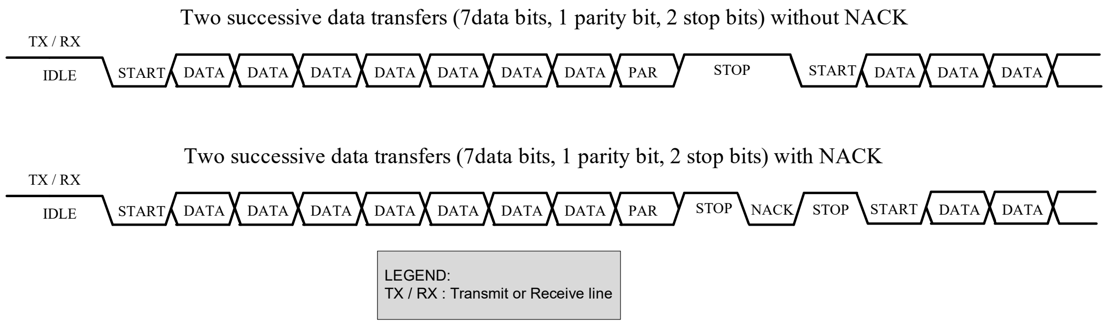
ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. UART_TX line is connected to SmartCard I/O line, by internally multiplexing between UART_TX and UART_RX control modules.

The SmartCard transfer is similar to a UART transfer, with the addition of a negative acknowledgement (NACK) that may be sent from the receiver to the transmitter. A NACK is always '0'. Both master and slave may drive the same line, although never at the same time.

A SmartCard transfer has the transmitter drive the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus. Releasing results in the line being '1' (the value of a stop bit). After one bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of '0') for one bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by one bit transfer period. For this protocol to work, the stop period should be longer than one bit transfer period. Note that a data transfer with a NACK takes one bit transfer period longer, than a data transfer without a NACK. Typically, implementations use a tristate driver with a pull-up resistor, such that when the line is not transmitting data or transmitting the Stop bit, its value is '1'.

Figure 15-20 illustrates the SmartCard protocol.

Figure 15-20. SmartCard Example



The communication Baud rate for ISO7816 is given as:

Baud rate= f_{7816} × (D/F)

Where f_{7816} is the clock frequency, F is the clock rate conversion integer, and D is the baud rate adjustment integer.

By default, F = 372, D = f1, and the maximum clock frequency is 5 MHz. Thus, maximum baud rate is 13.4 Kbps. Typically, a 3.57-MHz clock is selected. The typical value of the baud rate is 9.6 Kbps.

Configuring SCB as UART SmartCard Interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a SmartCard protocol by writing '01' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Follow steps 2 to 5 described in “Enabling and Initializing UART” on page 124.

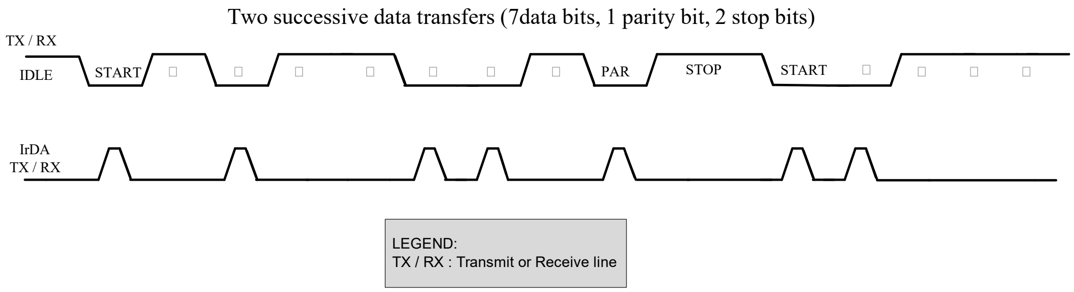
15.3.3.3 IrDA

The SCB supports the Infrared Data Association (IrDA) protocol for data rates of up to 115.2 Kbps using the UART interface. It supports only the basic physical layer of IrDA protocol with rates less than 115.2 Kbps. Hence, the system instantiating this block must consider how to implement a complete IrDA communication system with other available system resources.

The IrDA protocol adds a modulation scheme to the UART signaling. At the transmitter, bits are modulated. At the receiver, bits are demodulated. The modulation scheme uses a Return-to-Zero-Inverted (RZI) format. A bit value of '0' is signaled by a short '1' pulse on the line and a bit value of '1' is signaled by holding the line to '0'. For these data rates (<=115.2 Kbps), the RZI modulation scheme is used and the pulse duration is 3/16 of the bit period. The sampling clock frequency should be set 16 times the selected baud rate, by configuring the SCB_OVS field of the SCB_CTRL register.

Different communication speeds under 115.2 Kbps can be achieved by configuring corresponding block clock frequency. Additional allowable rates are 2.4 Kbps, 9.6 Kbps, 19.2 Kbps, 38.4 Kbps, and 57.6 Kbps. An IrDA serial infrared interface operates at 9.6 Kbps. Figure 15-21 shows how a UART transfer is IrDA modulated.

Figure 15-21. IrDA Example



Configuring the SCB as UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Enable the Median filter on the input interface line by writing ‘1’ to MEDIAN (bit 9) of the SCB_RX_CTRL register.
4. Configure the SCB as described in “Enabling and Initializing UART” on page 124.

15.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 15-9. For more information on these registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

Table 15-9. UART Registers

Register Name	Operation
SCB_CTRL	Enables the SCB; selects the type of serial interface (SPI, UART, I^2C)
SCB_UART_CTRL	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCB_UART_RX_STATUS	Used to specify the BR_COUNTER value that determines the bit period. This is used to set the accuracy of the SCB clock. This value provides more granularity than the OVS bit in SCB_CTRL register.
SCB_UART_TX_CTRL	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.
SCB_UART_RX_CTRL	Performs same function as SCB_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error.
SCB_TX_CTRL	Used to specify the data frame width and to specify whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_UART_FLOW_CONTROL	Configures flow control for UART transmitter.

15.3.5 UART Interrupts

The UART supports both internal and external interrupt requests. The internal interrupt events are listed in this section. PSoC Creator generates the necessary interrupt service routines (ISRs) for handling buffer management interrupts. Custom ISRs can also be used by connecting the external interrupt component to the interrupt output of the UART component (with external interrupts enabled).

The UART predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources is true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled Rx interrupt sources is true. The UART provides interrupts on the following events:

- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
  - TX received a NACK in SmartCard mode
  - TX done
  - Arbitration lost (in LIN or SmartCard modes)
- RX
  - RX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_RX_FIFO_CTRL
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
  - Frame error in received data frame
  - Parity error in received data frame
  - LIN baud rate detection is completed
  - LIN break detection is successful

15.3.6 Enabling and Initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCB_UART_CTRL register, according to Table 15-10. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-11.
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted or received.
3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in Table 15-12.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFOs.
4. Program the SCB_CTRL register to enable the SCB block. Also select the mode of operation (Table 15-13).
5. Enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from

SmartCard to IrDA). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example FIFO content).

Table 15-10. SCB_UART_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	Standard UART
		01	SmartCard
		10	IrDA
		11	Reserved
16	LOOP_BACK	Loop back control. This allows a SCB UART transmitter to communicate with its receiver counterpart.	

Table 15-11. SCB_TX_CTRL/SCB_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the no. of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits.
8	MSB_FIRST	1 = MSB first 0 = LSB first
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. For the UART IrDA mode, this should always be '1'. 1 = Enabled 0 = Disabled

Table 15-12. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[7:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared/invalidated.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze will not advance the TX or RX FIFO read/write pointer.

Table 15-13. SCB_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I^2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

15.4 Inter Integrated Circuit (I^2C)

This section explains the I^2C implementation in PSoC. For more information on the I^2C protocol specification, refer to the I^2C-bus specification available on the NXP website.

15.4.1 Features

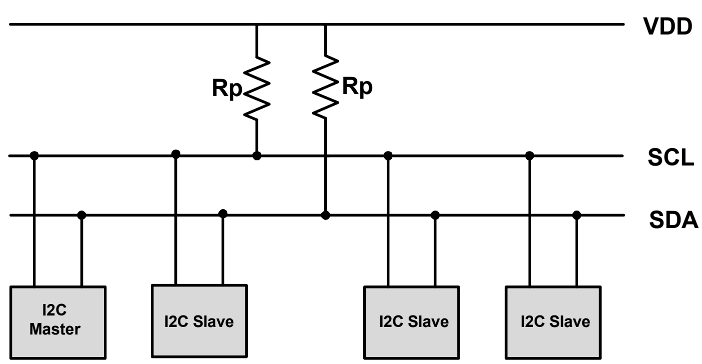
This block supports the following features:

- Master, slave, and master/slave mode
- Slow-mode (50 kbps), standard-mode (100 kbps), fast-mode (400 kbps), and fast-mode plus (1000 kbps) data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of I^2C clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I^2C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface

15.4.2 General Description

Figure 15-22 illustrates an example of an I^2C communication network.

Figure 15-22. I^2C Interface Block Diagram



The standard I^2C bus is a two wire interface with the following lines:

- Serial Data (SDA)
- Serial Clock (SCL)

I^2C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address. PSoC also supports 10-bit address matching for I^2C with firmware support.

15.4.3 Terms and Definitions

Table 15-14 explains the commonly used terms in an I^{2}C communication network.

Table 15-14. Definition of I^{2}C Bus Terminology

Term	Description
Transmitter	The device that sends data to the bus
Receiver	The device that receives data from the bus
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer
Slave	The device addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

15.4.3.1 Clock Stretching

When a slave device is not yet ready to process data, it may drive a ‘0’ on the SCL line to hold it down. Due to the implementation of the I/O signal interface, the SCL line value will be '0', independent of the values that any other master or slave may be driving on the SCL line. This is known as clock stretching and is the only situation in which a slave drives the SCL line. The master device monitors the SCL line and detects it when it cannot generate a positive clock pulse ('1') on the SCL line. It then reacts by delaying the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock.

15.4.3.2 Bus Arbitration

The I^{2}C protocol is a multi-master, multi-slave interface. Bus arbitration is implemented on master devices by monitoring the SDA line. Bus collisions are detected when the master observes an SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value '1' on the SDA line and master 2 is driving the value '0' on the SDA line, the actual line value will be '0' due to the implementation of the I/O signal interface. Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

15.4.4 I^{2}C Modes of Operation

I^{2}C is a synchronous single master, multi-master, multi-slave serial interface. Devices operate in either master mode, slave mode, or master/slave mode. In master/slave mode, the device switches from master to slave mode when it is addressed. Only a single master may be active during a data transfer. The active master is responsible for driving the clock on the SCL line. Table 15-15 illustrates the I^{2}C modes of operation.

Table 15-15. I^{2}C Modes

Mode	Description
Slave	Slave only operation (default)
Master	Master only operation
Multi-master	Supports more than one master on the bus
Multi-master-slave	Simultaneous slave and multi-master operation

Data transfer through the I^{2}C bus follows a specific format. Table 15-16 lists some common bus events that are part of an I^{2}C data transfer. The Write Transfer and Read Transfer sections explain the I^{2}C bus bit format during data transfer.

Table 15-16. I^{2}C Bus Events Terminology

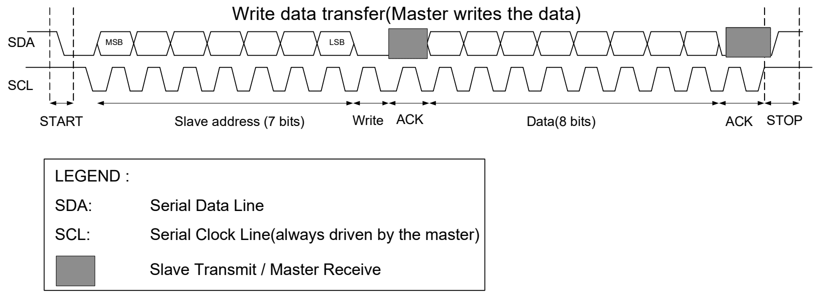
Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH
ACK	The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse, after the transmitter transmits each byte. This indicates to the transmitter that the receiver received the byte properly.
NACK	The receiver does not pull the SDA line LOW and it remains HIGH during the HIGH period of clock pulse after the transmitter transmits each byte. This indicates to the transmitter that the receiver received the byte properly.
Repeated START	START condition generated by master at the end of a transfer instead of a STOP condition
DATA	SDA status change while SCL is low (data changing), and no change while SCL is high (data valid)

When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see Table 15-16, Figure 15-23, and Figure 15-24). The master looks for a STOP signal as an indicator that it can start its data transmission.

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt, so that the device is ready to respond to any other master on the bus. With all of these modes, there are two types of transfer - read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in “Master Mode Transfer Examples” on page 135, “Slave Mode Transfer Examples” on page 137, and “Multi-Master Mode Transfer Example” on page 141.

15.4.4.1 Write Transfer

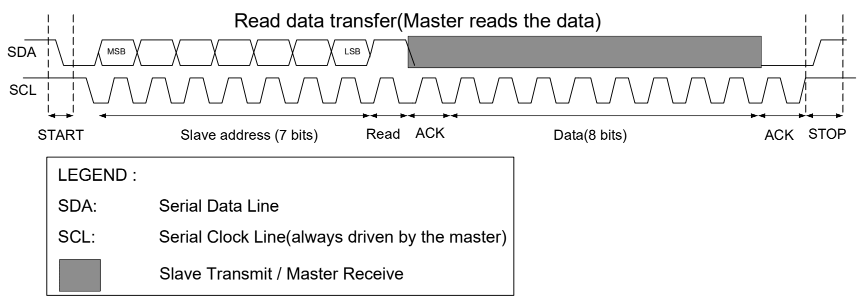
Figure 15-23. Master Write Data Transfer



- A typical write transfer begins with the master generating a START condition on the I^2C bus. The master then writes a 7-bit I^2C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match any of the slave devices or if the addressed device does not want to acknowledge the request, it transmits a no acknowledgement (NACK) by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- The master may transmit data to the bus if it receives an acknowledgement. The addressed slave transmits an acknowledgement to confirm the receipt of every byte of data written. Upon receipt of this acknowledgement, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

15.4.4.2 Read Transfer

Figure 15-24. Master Read Data Transfer



- A typical read transfer begins with the master generating a START condition on the I^2C bus. The master then writes a 7-bit I^2C slave address and a read indicator ('1') after the START condition. The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgement (NACK) is transmitted by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgement signal. The master transmits an acknowledgement to confirm the receipt of each data byte sent by the slave. Upon receipt of this acknowledgement, the addressed slave may transmit another data byte.
- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.

15.4.5 Easy I2C (EZI2C) Protocol

The Easy I2C (EZI2C) protocol is a unique communication scheme built on top of the I^2C protocol by Cypress. It uses a software wrapper around the standard I^2C protocol to communicate to an I^2C slave using indexed memory transfers. This removes the need for CPU intervention at the level of individual frames.

The EZI2C protocol defines an 8-bit address that indexes a memory array (8-bit wide 32 locations) located on the slave device. Five lower bits of the EZ address are used to address these 32 locations. The number of bytes transferred to or from the EZI2C memory array can be found by comparing the EZ address at the START event and the EZ address at the STOP event.

Note The I^2C block has a hardware FIFO memory, which is 16 bits wide and 16 locations deep with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has 16-bit wide eight locations. In EZ mode, the FIFO is used as a single memory unit with 8-bit wide 32 locations.

EZI2C has two types of transfers: a data write from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

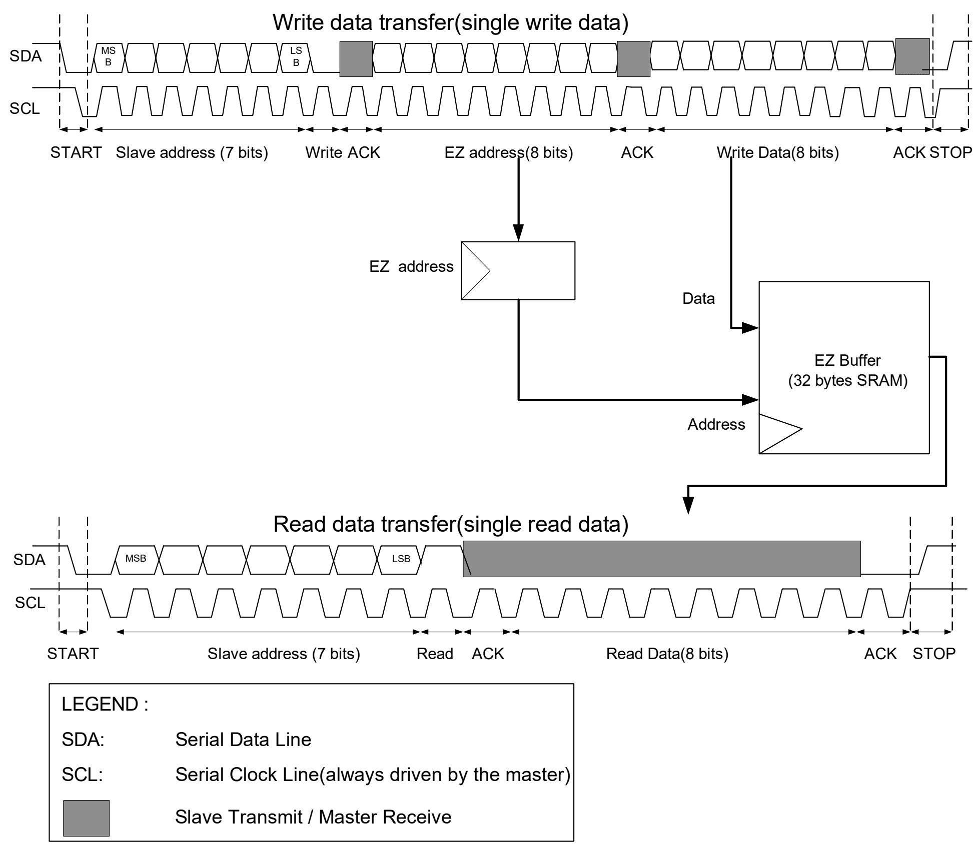
15.4.5.1 Memory Array Write

An EZ write to a memory array index is by means of an I^2C write transfer. The first transmitted write data is used to send an EZ address from the master to the slave. The five lowest significant bits of the write data are used as the "new" EZ address at the slave. Any additional write data elements in the write transfer are bytes that are written to the memory array. The EZ address is automatically incremented by the slave as bytes are written into the memory array. If the number of continuous data bytes written to the EZI2C buffer exceeds EZI2C buffer boundary, it overwrites the last location for every subsequent byte.

15.4.5.2 Memory Array Read

An EZ read from a memory array index is by means of an I^2C read transfer. The EZ read relies on an earlier EZ write to have set the EZ address at the slave. The first received read data is the byte from the memory array at the EZ address memory location. The EZ address is automatically incremented as bytes are read from the memory array. The address wraps around to zero when the final memory location is reached.

Figure 15-25. EZI2C Write and Read Data Transfer



15.4.6 I2C Registers

The I^2C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in Table 15-17.

Table 15-17. I2C Registers

Register	Function
SCB_CTRL	Enables the I2C block and selects the type of serial interface (I2C). Also used to select internally and externally clocked operation and EZ and non-EZ modes of operation.
SCB_I2C_CTRL	Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status.
SCB_I2C_STATUS	Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address.
SCB_I2C_M_CMD	Enables the master to generate START, STOP, and ACK/NACK signals.
SCB_I2C_S_CMD	Enables the slave to generate ACK/NACK signals.
SCB_STATUS	Indicates whether the externally clocked logic is using the EZ memory. This bit can be used by software to determine whether it is safe to issue a software access to the EZ memory.
SCB_I2C_CFG	Configures filters, which remove glitches from the SDA and SCL lines.
SCB_TX_CTRL	Specifies the data frame width; also used to specify whether MSB or LSB is the first bit in transmission.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clearing of the transmitter FIFO and shift registers, and FREEZE operation of the transmitter FIFO.
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_RX_FIFO_RD	Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO; behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Stores slave device address and is also used as slave device address MASK.
SCB_EZ_DATA	Holds the data in an EZ memory location.

Note Detailed descriptions of the I^2C register bits are available in the *PSoC Analog Coprocessor Family Registers TRM*.

15.4.7 I2C Interrupts

The fixed-function I^2C block generates interrupts for the following conditions.

- I2C Master
  - I2C master lost arbitration
  - I2C master received NACK
  - I2C master received ACK
  - I2C master sent STOP
  - I2C bus error (unexpected stop/start condition detected)
- I2C Slave
  - I2C slave lost arbitration
  - I2C slave received NACK
  - I2C slave received ACK
  - I2C slave received STOP
  - I2C slave received START
  - I2C slave address matched
  - I2C bus error (unexpected stop/start condition detected)
- TX
  - TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
  - TX FIFO is not full
  - TX FIFO is empty
  - TX FIFO overflow
  - TX FIFO underflow
- RX
  - RX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_RX_FIFO_CTRL
  - RX FIFO is full
  - RX FIFO is not empty
  - RX FIFO overflow
  - RX FIFO underflow
- I2C Externally Clocked
  - Wake up request on address match
  - I2C STOP detection at the end of each transfer
  - I2C STOP detection at the end of a write transfer
  - I2C STOP detection at the end of a read transfer

The I2C interrupt signal is hard-wired to the Cortex-M0 NVIC and cannot be routed to external pins.

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

15.4.8 Enabling and Initializing the I2C

The following section describes the method to configure the I2C block for standard (non-EZ) mode and EZI2C mode.

15.4.8.1 I2C Standard (Non-EZ) Mode Configuration

The I2C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to Table 15-18. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-19.
   a. Specify the data frame width.
   b. Specify that MSB is the first bit to be transmitted/received.
3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in Table 15-20.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCB_CTRL register to enable the I2C block and select the I2C mode. These register bits are shown in Table 15-21. For a complete description of the I2C registers, see the *PSoC Analog Coprocessor Family Registers TRM*.

Table 15-18. SCB_I2C_CTRL Register

Bits	Name	Value	Description
30	SLAVE_MODE	1	Slave mode
31	MASTER_MODE	1	Master mode

Table 15-19. SCB_TX_CTRL/SCB_RX_CTRL Register

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. For I2C, this is always 7.
8	MSB_FIRST	1= MSB first (this should always be true for I2C)
		0= LSB first
9	MEDIAN	This is for SCB_RX_CTRL only. Decides whether a digital three-tap median filter is applied on the input interface lines. This filter should reduce susceptibility to errors, but it requires higher oversampling values. 1=Enabled 0=Disabled

Table 15-20. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL

Bits	Name	Description
[7:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or the receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

Table 15-21. SCB_CTRL Registers

Bits	Name	Value	Description
[25:24]	MODE	00	I2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block disabled
		1	SCB block enabled

15.4.8.2 EZI2C Mode Configuration

To configure the I2C block for EZI2C mode, set the following I2C register bits

1. Select the EZI2C mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Follow steps 2 to 4 mentioned in I2C Standard (Non-EZ) Mode Configuration.
3. Set the S_READY_ADDR_ACK (bit 12) and S_READY_DATA_ACK (bit 13) bits of the SCB_I2C_CTRL register.

15.4.9 Internal and External Clock Operation in I2C

The I2C block supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the PSoC system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the Clocking System chapter on page 76.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality.

TX and RX FIFOs do not support externally clocked operation; therefore, it is not used for non-EZ functionality.

Internally and externally clocked operations are determined by two register fields of the SCB_CTRL register:

- **EC_AM_MODE (Externally Clocked Address Matching Mode)**: Indicates whether I2C address matching is internally ('0') or externally ('1') clocked.
- **EC_OP_MODE (Externally Clocked Operation Mode)**: Indicates whether the rest of the protocol operation (besides I2C address match) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does not support non-EZ functionality.

These two register fields determine the functional behavior of I2C. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power modes. Improper setting may result in faulty behavior in certain power modes. Table 15-22 and Table 15-23 describe the settings for I2C in EZ and non-EZ mode.

15.4.9.1 I2C Non-EZ Mode of Operation

Externally clocked operation is not supported for non-EZ functionality because there is no FIFO support for this mode. So, the EC_OP_MODE should always be set to '0'for non-EZ mode. However, EC_AM_MODE can be set to '0' or '1'.Table 15-22 gives an overview of the possibilities. The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

EC_AM_MODE is '0' and EC_OP_MODE is '0'.

This setting only works in Active and Sleep system power modes. All the functionality of the I2C is provided in the internally clocked domain.

EC_AM_MODE is '1' and EC_OP_MODE is '0'.

This setting works in Active, Sleep, and Deep-Sleep system power modes. I2C address matching is performed by the externally clocked logic in Active, Sleep, and Deep-Sleep system power modes. When the externally clocked logic matches the address, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wakeup the CPU.

Table 15-22. I2C Operation in Non-EZ Mode

<table>
<tr><th colspan="5">I2C (Non-EZ) Mode</th></tr>
<tr><th rowspan="2">System Power Mode</th><th colspan="2">EC_OP_MODE = 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th></tr>
<tr><td>Active and Sleep</td><td>Address match using internal clock.
Operation using internal clock.</td><td>Address match using external clock.
Operation using internal clock.</td><td colspan="2" rowspan="2">Not supported</td></tr>
<tr><td>Deep-Sleep</td><td>Not supported</td><td>Address match using external clock.
Operation using internal clock.</td></tr>
</table>

- In Active system power mode, the CPU is active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). The externally clocked logic takes care of the address matching and the internally locked logic takes care of the rest of the I2C transfer.
- In the Sleep mode, wakeup interrupt cause can be either enabled or disabled based on the application. The remaining operations are similar to the Active mode.
- In the Deep-Sleep mode, the CPU is shut down and will wake up on I2C activity if the wakeup interrupt cause is enabled. CPU wakeup up takes time and the ongoing I2C transfer is either negatively acknowledged (NACK) or the clock is stretched. In the case of a NACK, the internally clocked logic takes care of the first I2C transfer after it wakes up. For clock stretching, the internally clocked logic takes care of the ongoing/stretched transfer when it wakes up. The register bit S_NOT_READY_ADDR_NACK (bit 14) of the SCB_I2C_CTRL register determines whether the externally clocked logic performs a negative acknowledge ('1') or clock stretch ('0').

15.4.9.2 I2C EZ Operation Mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-23 gives an overview of the possibilities. The grey cells indicate a possible, yet not recommended setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

Table 15-23. I2C Operation in EZ Mode

I2C, EZ Mode				
System Power Mode	**EC_OP_MODE= 0**		**EC_OP_MODE = 1**	
	EC_AM_MODE = 0	**EC_AM_MODE = 1**	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**
Active and Sleep	Address match using internal clock Operation using internal clock	Address match using external clock Operation using internal clock	Invalid	Address match using external clock Operation using external clock
Deep-Sleep	Not supported	Address match using external clock Operation using internal clock		Address match using external clock Operation using external clock

- **EC_AM_MODE is '0' and EC_OP_MODE is '0'**. This setting only works in Active and Sleep system power modes.
- **EC_AM_MODE is '1' and EC_OP_MODE is '0'**. This setting works same as I2C non-EZ mode.
- **EC_AM_MODE is '1' and EC_OP_MODE is '1'**. This setting works in Active and Deep-Sleep system power modes.

The I2C block’s functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK (bit 17) of the SCB_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

15.4.10 Wake up from Sleep

The system wakes up from Sleep or Deep-Sleep system power modes when an I2C address match occurs. The fixed-function I2C block performs either of two actions after address match: Address ACK or Address NACK.

Address ACK - The I2C slave executes clock stretching and waits until the device wakes up and ACKs the address.

Address NACK - The I2C slave NACKs the address immediately. The master must poll the slave again after the device wakeup time is passed. This option is only valid in the slave or multi-master-slave modes.

Note The interrupt bit WAKE_UP (bit 0) of the SCB_INTR_I2C_EC register must be enabled for the I2C to wake up the device on slave address match while switching to the Sleep mode.

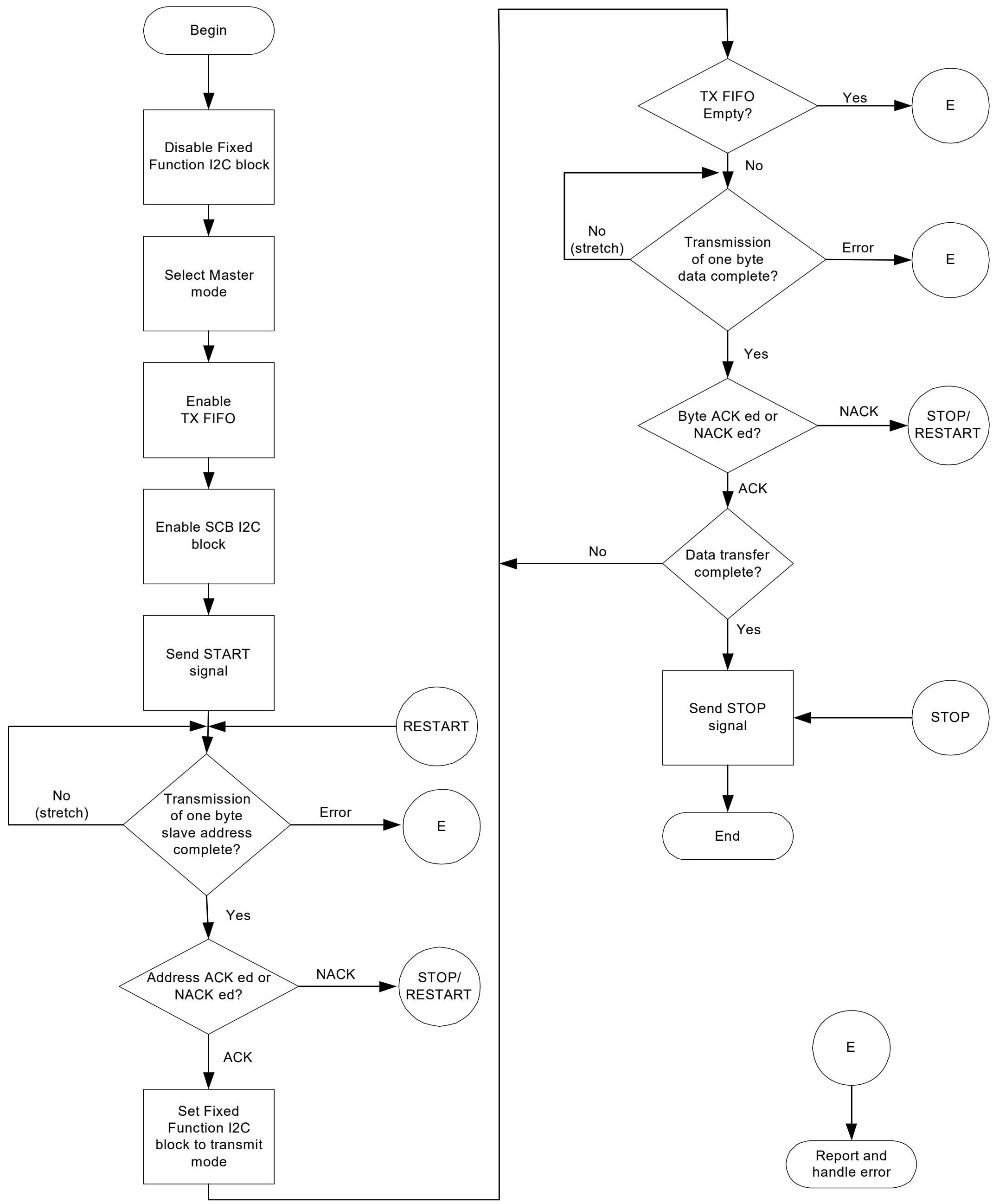
Note If the device is configured in I2C slave mode, the clock to the SCB should be disabled when entering Deep-Sleep power mode; enable the clock when waking up from Deep-Sleep mode.

15.4.11 Master Mode Transfer Examples

Master mode transmits or receives data.

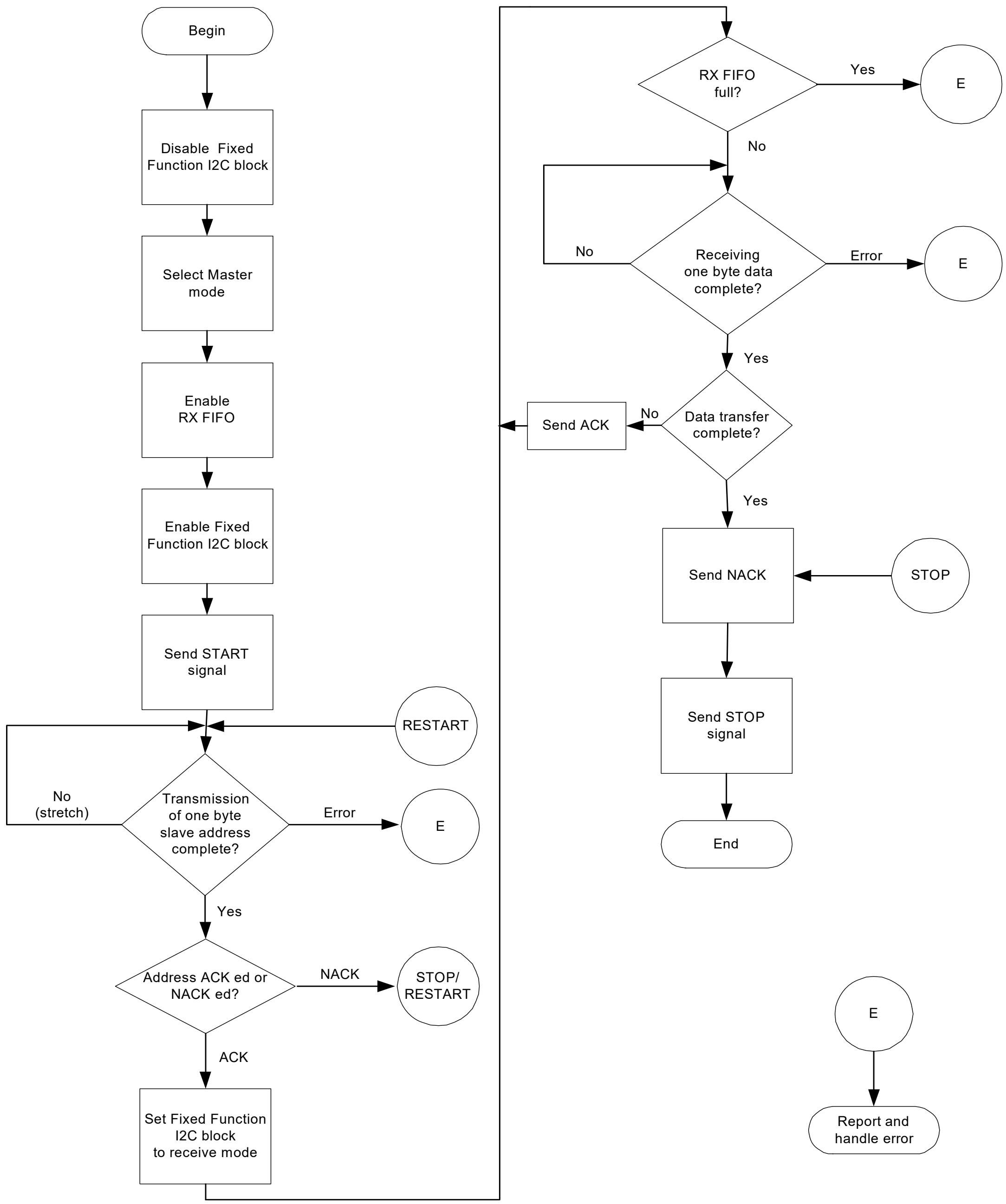
15.4.11.1 Master Transmit

Figure 15-26. Single Master Mode Write Operation Flow Chart



15.4.11.2 Master Receive

Figure 15-27. Single Master Mode Read Operation Flow Chart

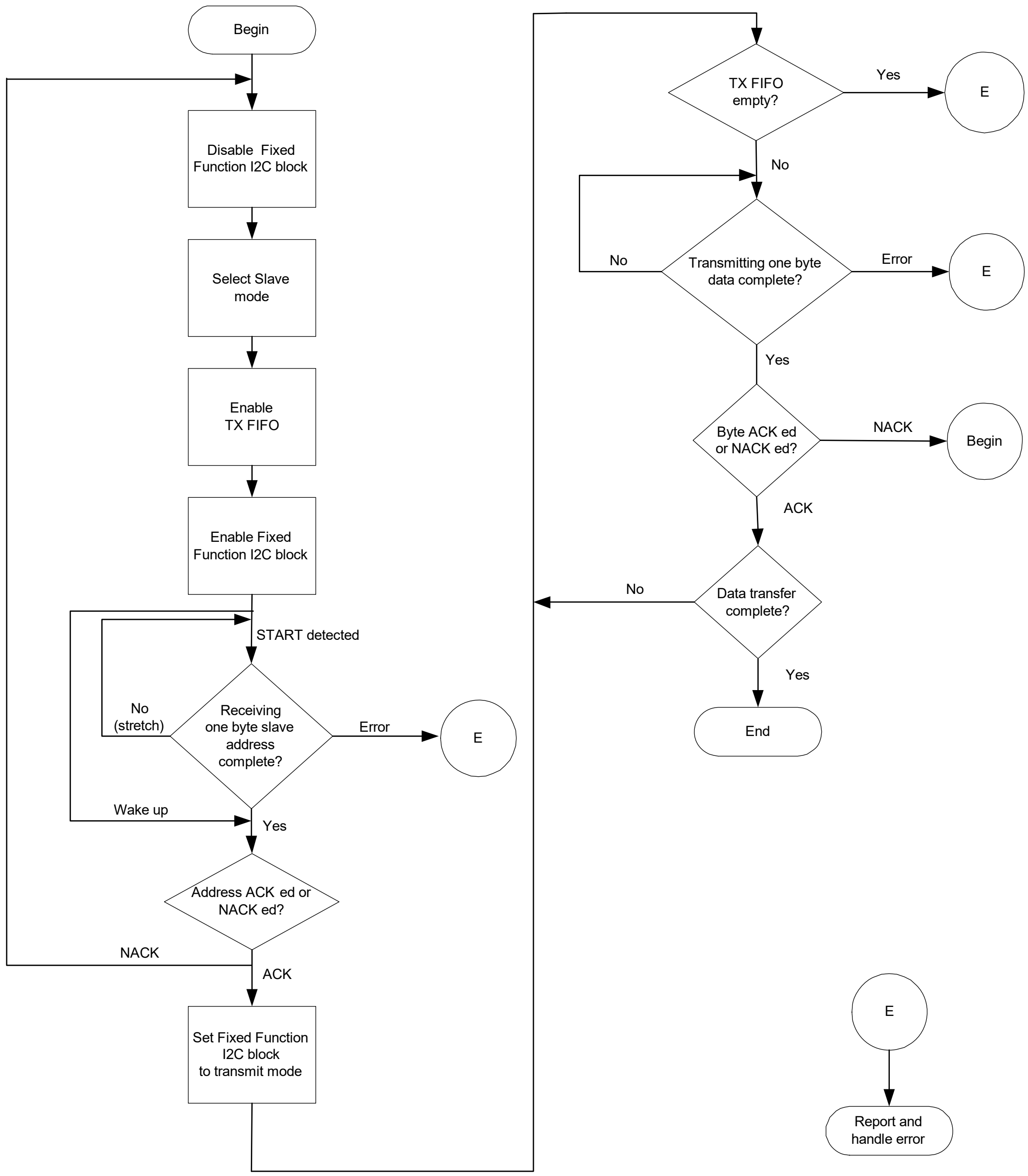


15.4.12 Slave Mode Transfer Examples

Slave mode transmits or receives data.

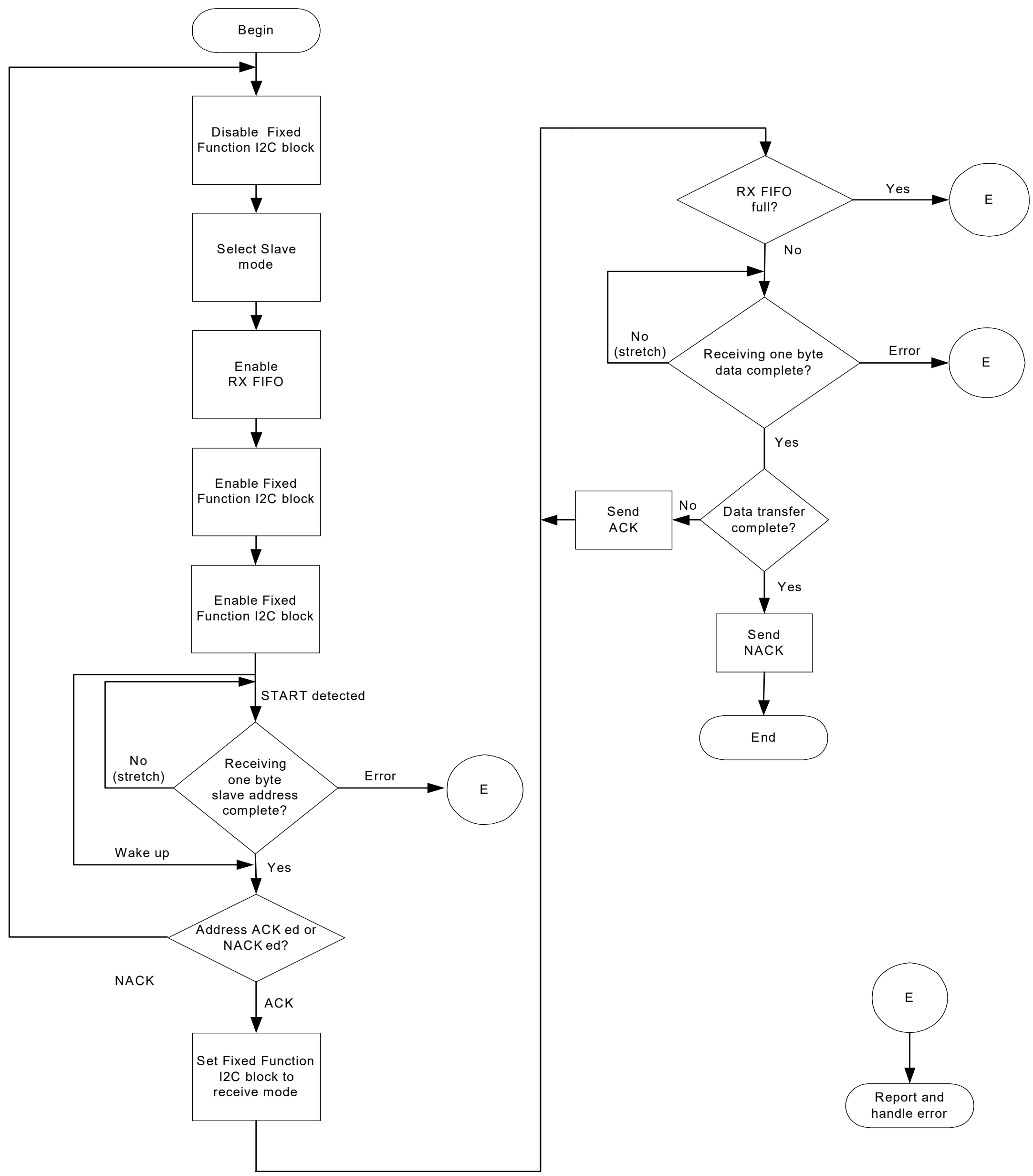
15.4.12.1 Slave Transmit

Figure 15-28. Slave Mode Write Operation Flow Chart



15.4.12.2 Slave Receive

Figure 15-29. Slave Mode Read Operation Flow Chart

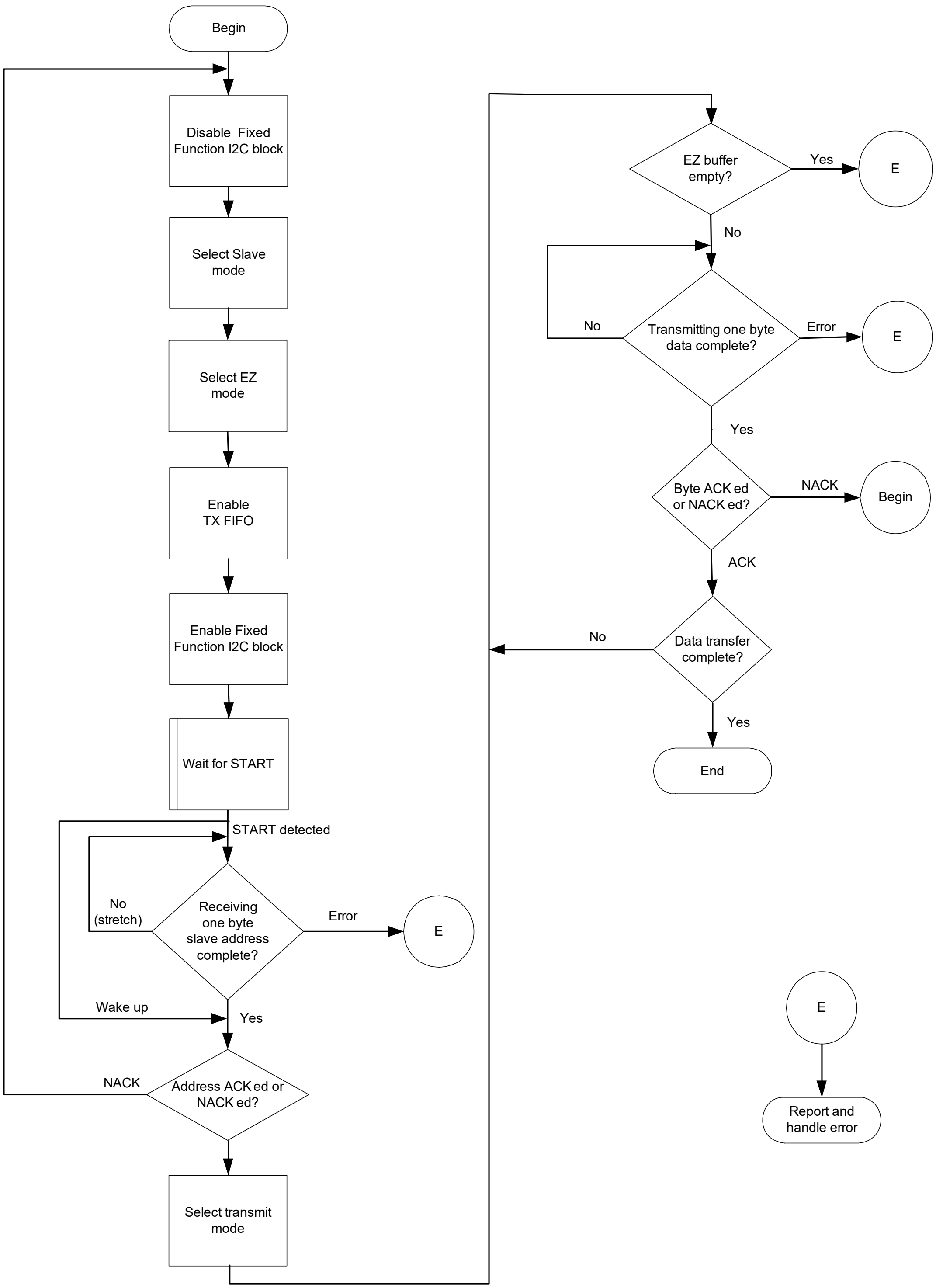


15.4.13 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

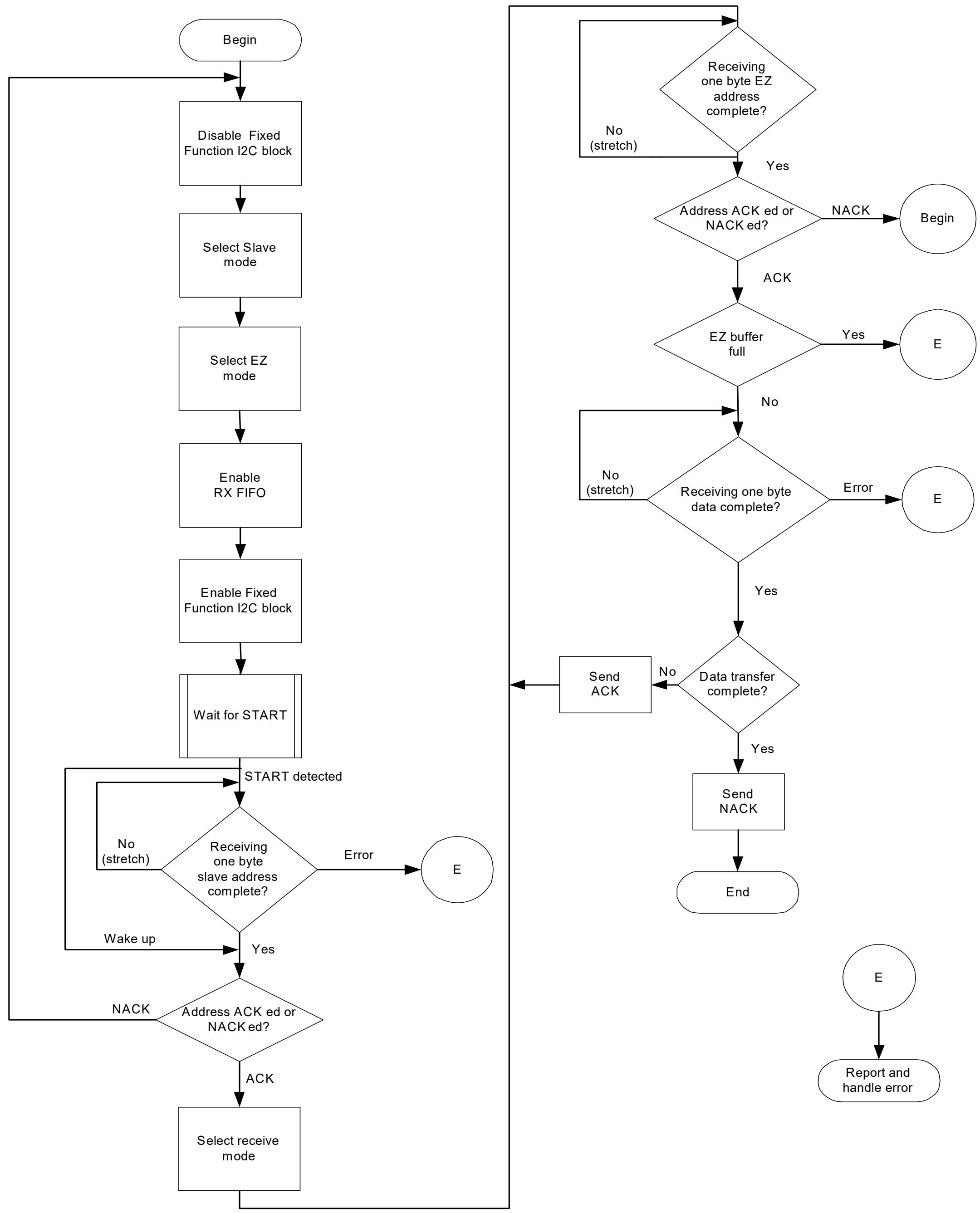
15.4.13.1 EZ Slave Transmit

Figure 15-30. EZI2C Slave Mode Write Operation Flow Chart



15.4.13.2 EZ Slave Receive

Figure 15-31. EZI2C Slave Mode Read Operation Flow Chart

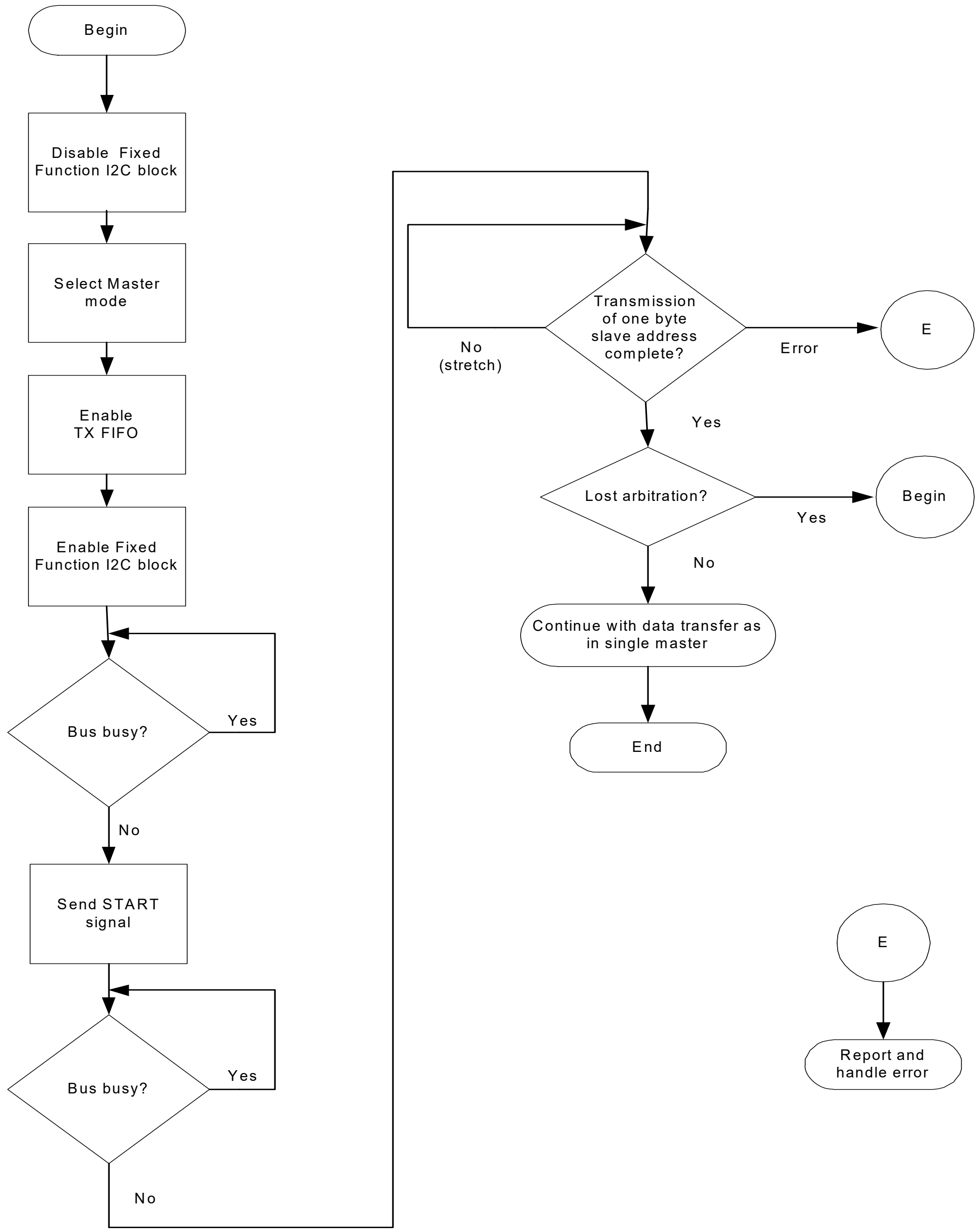


15.4.14 Multi-Master Mode Transfer Example

In multi-master mode, data can be transferred with the slave mode enabled or not enabled.

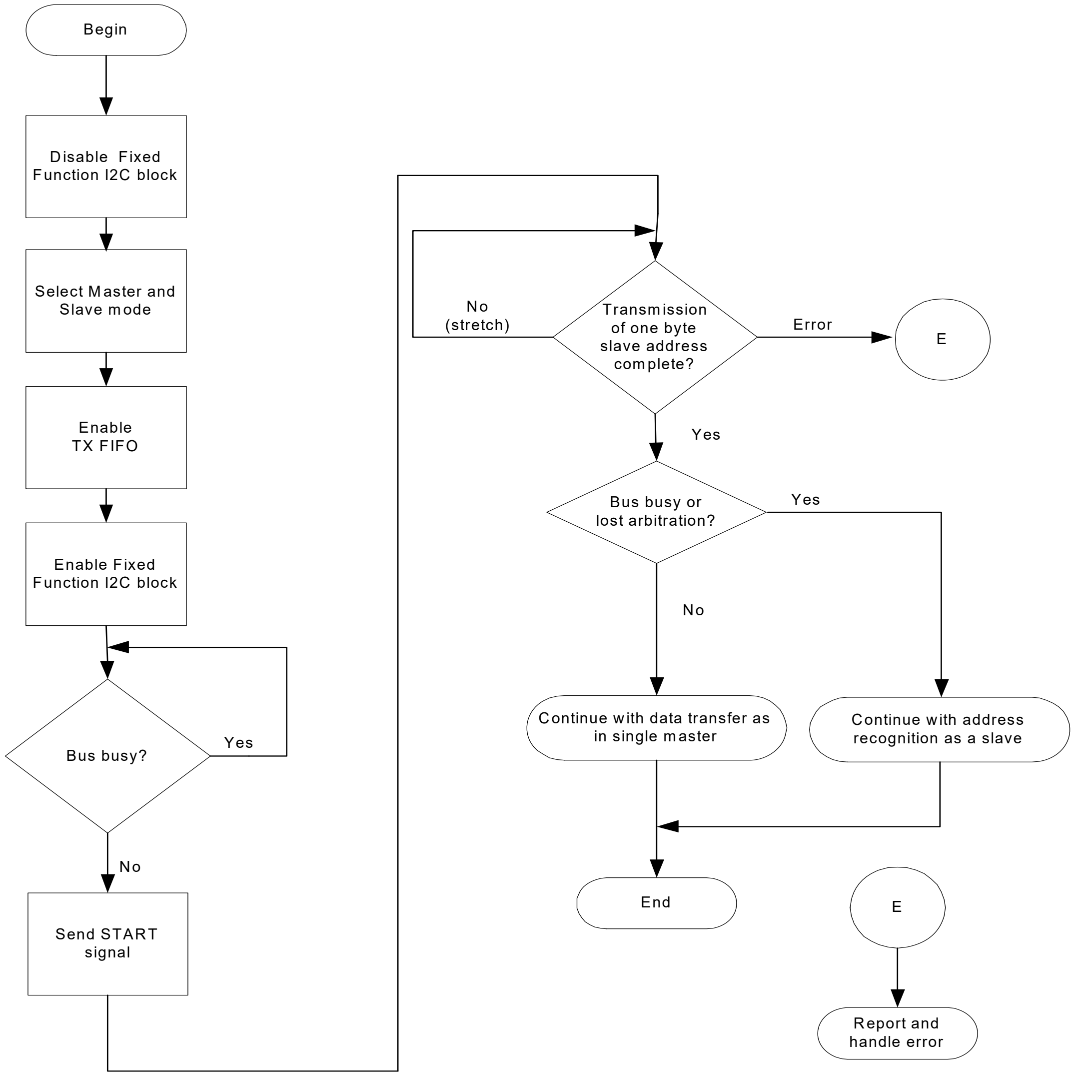
15.4.14.1 Multi-Master - Slave Not Enabled

Figure 15-32. Multi-Master, Slave Not Enabled Flow Chart



15.4.14.2 Multi-Master - Slave Enabled

Figure 15-33. Multi-Master, Slave Enabled Flow Chart



16. Timer, Counter, and PWM

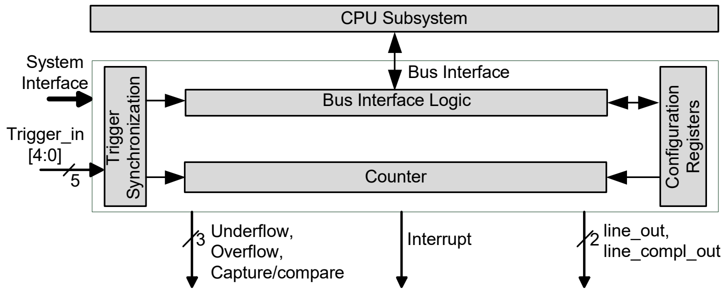
The Timer, Counter, and Pulse Width Modulator (TCPWM) block in the PSoC® Analog Coprocessor implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular event occurs (counter), generate PWM signals, or decode quadrature signals. This chapter explains the features, implementation, and operational modes of the TCPWM block.

16.1 Features

- One 16-bit timer, counter, or pulse width modulator (PWM)
- The TCPWM block supports the following operational modes:
  - Timer
  - Capture
  - Quadrature decoding
  - Pulse width modulation
  - Pseudo-random PWM
  - PWM with dead time
- Multiple counting modes – up, down, and up/down
- Clock prescaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count – The final value in the counter register is reached
  - Capture/Compare – The count is captured to the capture/compare register or the counter value equals the compare value
- Underflow, overflow, and capture/compare output signals that can be routed to dedicated GPIOs and peripherals such as DMA and SAR ADC
- Complementary line output for PWMs
- Selectable start, reload, stop, count, and capture event signals for the TCPWM from the dedicated GPIOs and on-chip peripherals with rising edge, falling edge, both edges, and level trigger options

16.2 Block Diagram

Figure 16-1. TCPWM Block Diagram



The block has these interfaces:

- Bus interface: Connects the block to the CPU subsystem.
- I/O signal interface: Connects input triggers (such as reload, start, stop, count, and capture) and output signals (such as overflow (OV), underflow (UN), and capture/compare (CC)) to dedicated GPIOs.
- Interrupts: Provides interrupt request signals from the counter, based on terminal count (TC) or CC conditions.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem.

This TCPWM block can be configured by writing to the TCPWM registers. See "TCPWM Registers" on page 166 for more information on all registers required for this block.

16.2.1 Enabling and Disabling Counter in TCPWM Block

The counter can be enabled by setting the COUNTER_ENABLED field (bit 0) of the control register TCPWM_CTRL.

Note The counter must be configured before enabling it. If the counter is enabled after being configured, registers are updated with the new configuration values. Disabling the counter retains the values in the registers until it is enabled again (or reconfigured).

16.2.2 Clocking

The TCPWM receives the HFCLK through the system interface to synchronize all events in the block. The counter enable signal (counter_en), which is generated when the counter is enabled, gates the HFCLK to provide a counter-specific clock (counter_clock). Output triggers (explained later in this chapter) are also synchronized with the HFCLK.

Clock Prescaling: counter_clock can be prescaled, with divider values of 1, 2, 4… 64, 128. This prescaling is done by modifying the GENERIC field of the counter control (TCPWM_CNT_CTRL) register, as shown in Table 16-1.

Table 16-1. Bit-Field Setting to Prescale Counter Clock

GENERIC[10:8]	Description
0	Divide by 1
1	Divide by 2
2	Divide by 4
3	Divide by 8
4	Divide by 16
5	Divide by 32
6	Divide by 64
7	Divide by 128

Note Clock prescaling cannot be done in quadrature mode and PWM-DT mode.

16.2.3 Events Based on Trigger Inputs

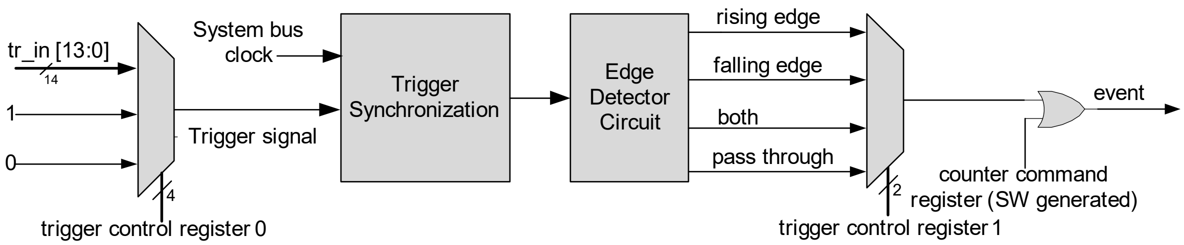
These are the events triggered by hardware or software.

- Reload
- Start
- Stop
- Count
- Capture/switch

Hardware triggers can be level signal, rising edge, falling edge, or both edges. Figure 16-2 shows the selection of edge detection type for any event trigger signal. The trigger control register 0 (TCPWM_CNT_TR_CTRL0) selects one of the 1416five trigger inputs as the event signal, which includes constant '0' and '1' signals.

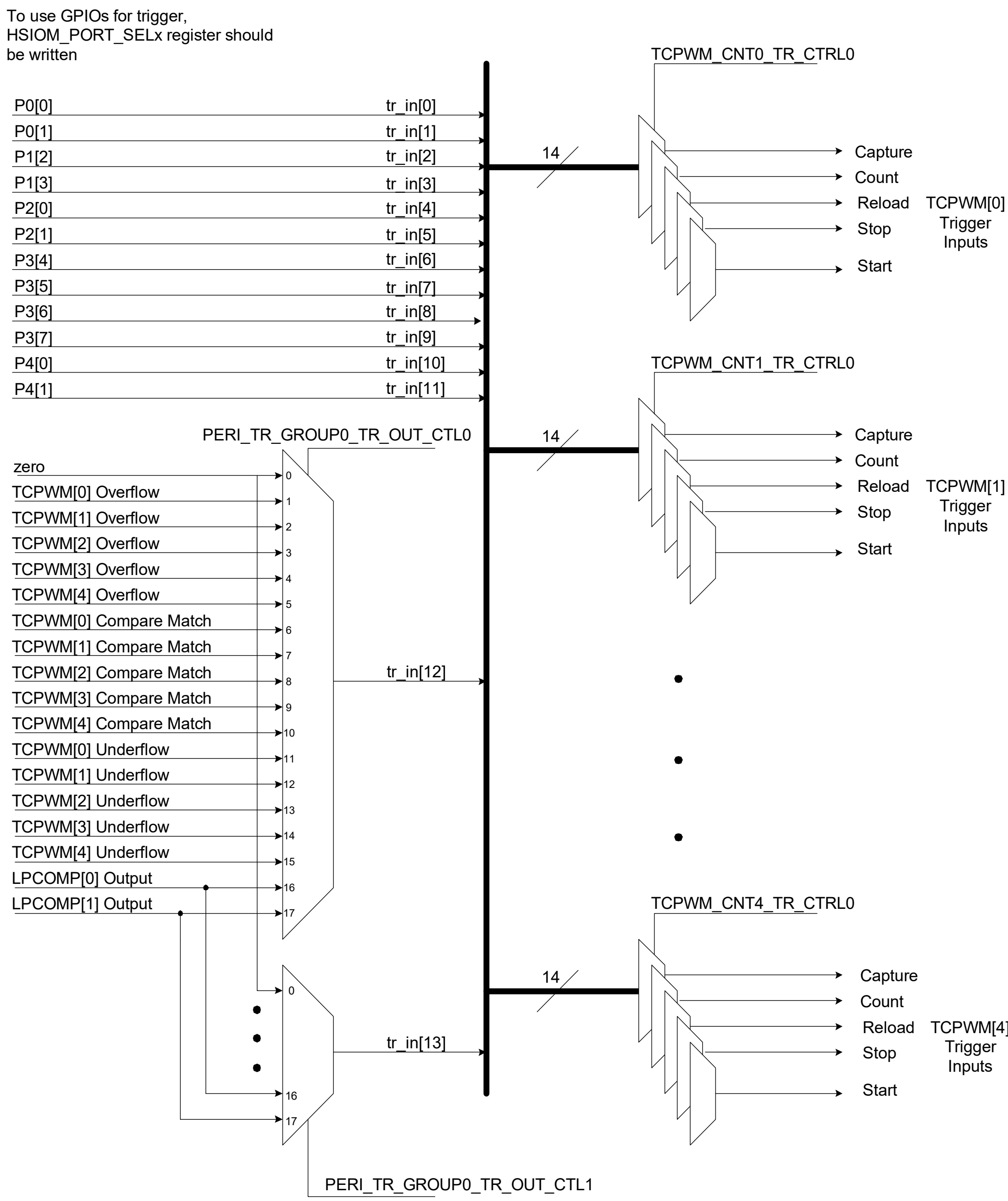
Any edge (rising, falling, or both) or level (high or low) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNT_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 16-2.

Figure 16-2. Trigger Signal Edge Detection



The trigger signal to generate an event can be a GPIO signal, TCPWM's underflow, compare match or overflow signal, or a low-power comparator (LPCOMP) output signal. Figure 16-3 shows the trigger signal selection for all the events.

Figure 16-3. Trigger Mux



The events derived from these triggers can have different definitions in different modes of the TCPWM block.

- **Reload:** A reload event initializes and starts the counter.
  - In UP counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with ‘0’.
  - In DOWN counting mode, the counter is initialized with the period value stored in the TCPWM_CNT_PERIOD register.
  - In UP/DOWN counting mode, the count register is initialized with ‘1’.
  - In quadrature mode, the reload event acts as a quadrature index event. An index/reload event indicates a completed rotation and can be used to synchronize quadrature decoding.
- **Start:** A start event is used to start counting; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
  - In quadrature mode, the start event acts as quadrature phase input phiB, which is explained in detail in “Quadrature Decoder Mode” on page 155.
- **Count:** A count event causes the counter to increment or decrement, depending on its configuration.
  - In quadrature mode, the count event acts as quadrature phase input phiA.

- **Stop:** A stop event stops the counter from incrementing or decrementing. A start event will start the counting again.
  - In the PWM modes, the stop event acts as a kill event. A kill event disables all the PWM output lines.
- **Capture:** A capture event copies the counter register value to the capture register and capture register value to the buffer capture register. In the PWM modes, the capture event acts as a switch event. It switches the values of the capture/compare and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

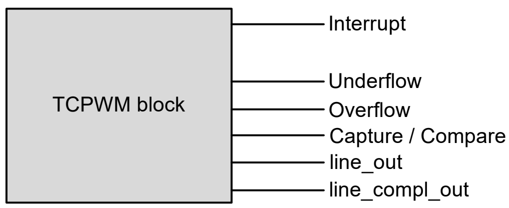
Notes

- All trigger inputs are synchronized to the HFCLK.
- When more than one event occurs in the same counter clock period, one or more events may be missed. This can happen for high-frequency events (frequencies close to the counter frequency) and a timer configuration in which a pre-scaled (divided) counter clock is used.

16.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 16-4.

Figure 16-4. TCPWM Output Signals



16.2.4.1 Signals upon Trigger Conditions

- Counter generates an internal overflow (OV) condition when counting up and the count register reaches the period value.
- Counter generates an internal underflow (UN) condition when counting down and the count register reaches zero.
- The capture/compare (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:
  - The counter value equals the compare value.
  - A capture event occurs - When a capture event occurs, the TCPWM_CNT_COUNTER register value is copied to the capture register and the capture register value is copied to the buffer capture register.

Note These signals, when they occur, remain at logic high for two cycles of the HFCLK. For reliable operation, the condition that causes this trigger should be less than a quarter of the HFCLK. For example, if the HFCLK is running at 24 MHz, the condition causing the trigger should occur at a frequency less than 6 MHz.

16.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific.

Four registers are used for interrupt handling in this block, as shown in Table 16-2.

Table 16-2. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNT_INTR (Interrupt request register)	0	TC	This bit is set to '1', when a terminal count is detected. Write '1' to clear this bit.
	1	CC_MATCH	This bit is set to ‘1’ when the counter value matches capture/compare register value. Write '1' to clear this bit.
TCPWM_CNT_INTR_SET (Interrupt set request register)	0	TC	Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
	1	CC_MATCH	Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.

Table 16-2. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNT_INTR_MASK (Interrupt mask register)	0	TC	Mask bit for the corresponding TC bit in the interrupt request register.
	1	CC_MATCH	Mask bit for the corresponding CC_MATCH bit in the interrupt request register.
TCPWM_CNT_INTR_MASKED (Interrupt masked request register)	0	TC	Logical AND of the corresponding TC request and mask bits.
	1	CC_MATCH	Logical AND of the corresponding CC_MATCH request and mask bits.

16.2.4.3 Outputs

The TCPWM has two outputs, line_out and line_compl_out (complementary of line_out). Note that the OV, UN, and CC conditions can be used to drive line_out and line_compl_out if needed, by configuring the TCPWM_CNT_TR_CTRL2 register (Table 16-3).

Table 16-3. Configuring Output Line for OV, UN, and CC Conditions

Field	Bit	Value	Event	Description
CC_MATCH_MODE Default Value = 3	1:0	0	Set line_out to '1	Configures output line on a compare match (CC) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	
OVERFLOW_MODE Default Value = 3	3:2	0	Set line_out to '1	Configures output line on a overflow (OV) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	
UNDERFLOW_MODE Default Value = 3	5:4	0	Set line_out to '1	Configures output line on a underflow (UN) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	

16.2.5 Power Modes

The TCPWM block works in Active and Sleep modes. The TCPWM block is powered from V_{CCD}. The configuration registers and other logic are powered in Deep-Sleep mode to keep the states of configuration registers. See Table 16-4 for details.

Table 16-4. Power Modes in TCPWM Block

Power Mode	Block Status
Active	This block is fully operational in this mode with clock running and power switched on.
Sleep	All counter clocks are on, but bus interface cannot be accessed.
Deep-Sleep	In this mode, the power to this block is still on but no bus clock is provided; hence, the logic is not functional. All the configuration registers will keep their state.

16.3 Modes of Operation

The counter block can function in six operational modes, as shown in Table 16-5. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

Table 16-5. Operational Mode Configuration

Mode	MODE Field [26:24]	Description
Timer	000	Implements a timer or counter. The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected.
Capture	010	Implements a timer or counter with capture input. The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected. When a capture event occurs, the counter value copies into the capture register.
Quadrature Decoder	011	Implements a quadrature decoder, where the counter is decremented or incremented, based on two phase inputs according to the selected (X1, X2 or X4) encoding scheme.
PWM	100	Implements edge/center-aligned PWMs with an 8-bit clock prescaler and buffered compare/period registers.
PWM-DT	101	Implements edge/center-aligned PWMs with configurable 8-bit dead time (on both outputs) and buffered compare/period registers.
PWM-PR	110	Implements a pseudo-random PWM using a 16-bit linear feedback shift register (LFSR).

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the TCPWM_CNT_CTRL register, as shown in Table 16-6.

Table 16-6. Counting Mode Configuration

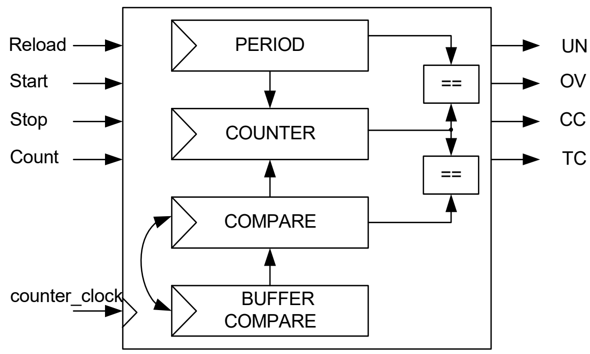
Counting Modes	UP_DOWN_ MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until 0 is reached. A TC condition is generated when the counter reaches ‘0’.
UP/DOWN Counting Mode 0	10	Increments the counter until the period value is reached, and then decrements the counter until ‘0’ is reached. A TC condition is generated only when ‘0’ is reached.
UP/DOWN Counting Mode 1	11	Similar to up/down counting mode 0 but a TC condition is generated when the counter reaches ‘0’ and when the counter value reaches the period value.

16.3.1 Timer Mode

The timer mode is commonly used to measure the time of occurrence of an event or to measure the time difference between two events.

16.3.1.1 Block Diagram

Figure 16-5. Timer Mode Block Diagram



16.3.1.2 How It Works

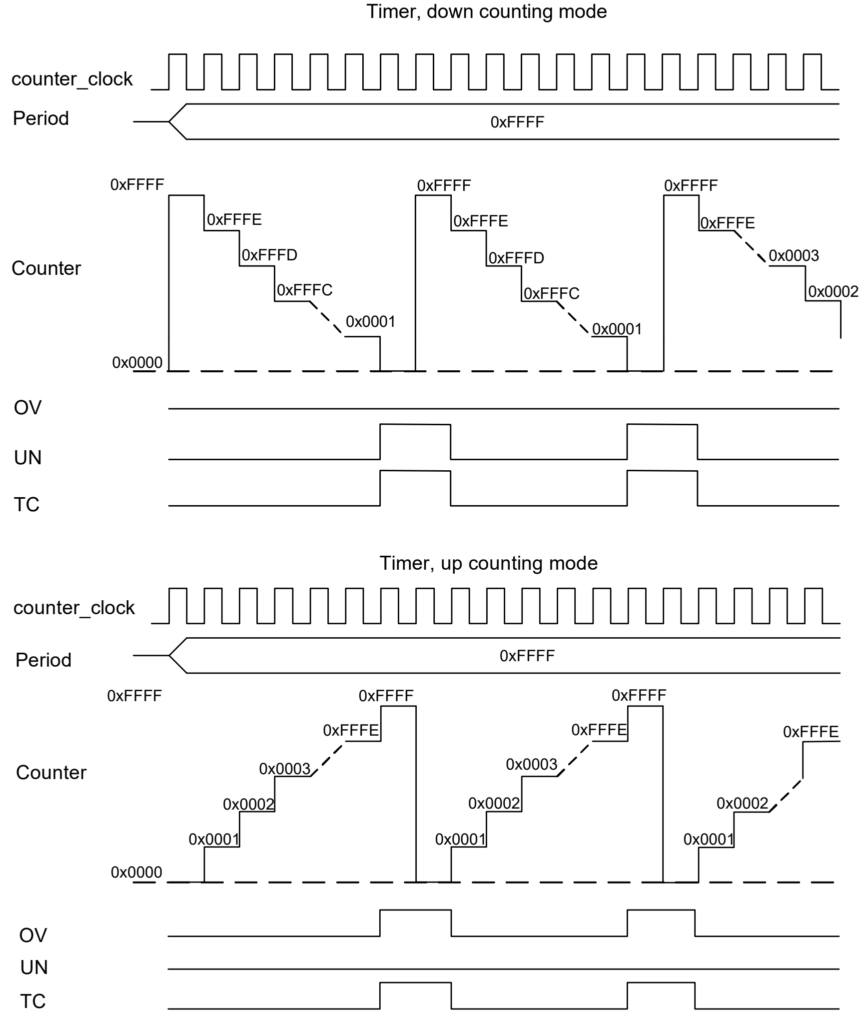
The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode. The following explains the working of the timer:

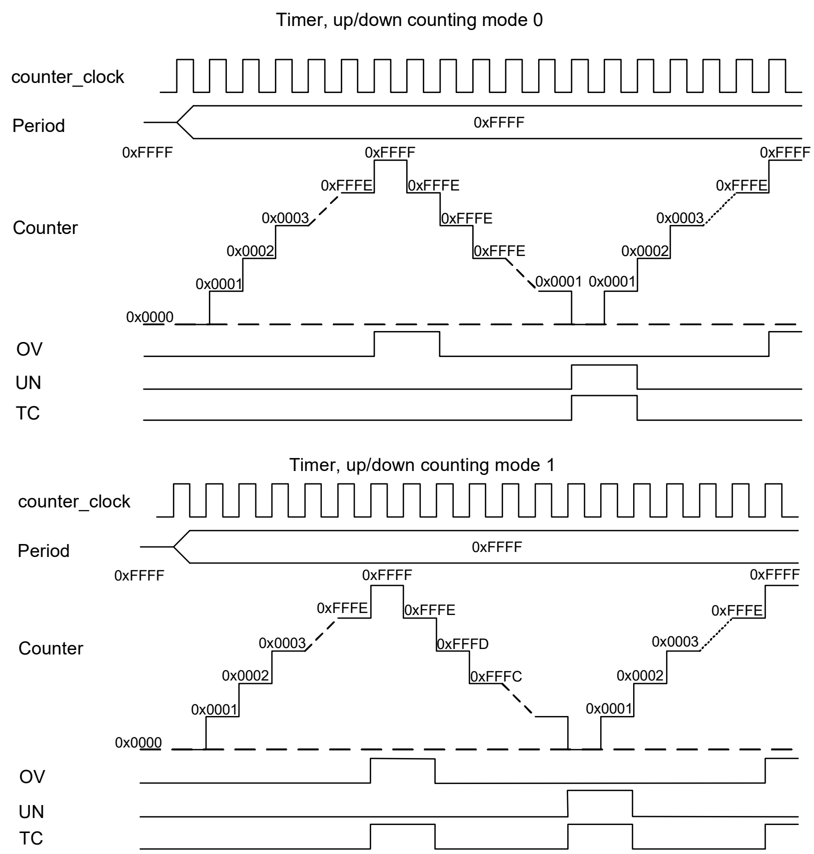
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
    Note It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with the value in the period register.
  - In the up/down counting modes, the count register value is not updated upon reaching the terminal values. Instead the direction of counting changes when the count value reaches 0 or the period value.
- The CC condition is generated when the count register value equals the compare register value. Upon this condition, the compare register and buffer compare register switch their values if enabled by the AUTO_RELOAD_CC bit-field of the counter control (TCPWM_CNT_CTRL) register. This condition can be used to generate an interrupt request.

Figure 16-6 shows the timer operational mode of the counter in four different counting modes. The period register contains the maximum counter value.

- In the up counting mode, a period value of A results in A+1 counter cycles (0 to A).
- In the down counting mode, a period value of A results in A+1 counter cycles (A to 0).
- In the two up/down counting modes (0 and 1), a period value of A results in 2*A counter cycles (0 to A and back to 0).

Figure 16-6. Timing Diagram for Timer in Multiple Counting Modes





Note The OV and UN signals remain at logic high for two cycles of the HFCLK, as explained in “Signals upon Trigger Conditions” on page 147. The figures in this chapter assume that HFCLK and counter clock are the same.

16.3.1.3 Configuring Counter for Timer Mode

The steps to configure the counter for Timer mode of operation and the affected register bits are as follows.

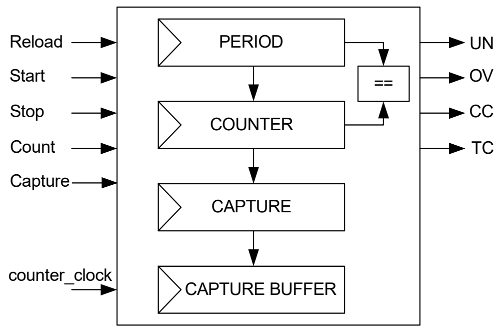
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register.
5. Set AUTO_RELOAD_CC field of the TCPWM_CNT_CTRL register, if required to switch values at every CC condition.
6. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
7. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in Table 16-6.
8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of TCPWM_CNT_CTRL.
9. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
10. Set the TCPWM_CNT_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
11. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 147.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

16.3.2 Capture Mode

In the capture mode, the counter value can be captured at any time either through a firmware write to command register (TCPWM_CMD) or a capture trigger input. This mode is used for period and pulse width measurement.

16.3.2.1 Block Diagram

Figure 16-7. Capture Mode Block Diagram



16.3.2.2 How it Works

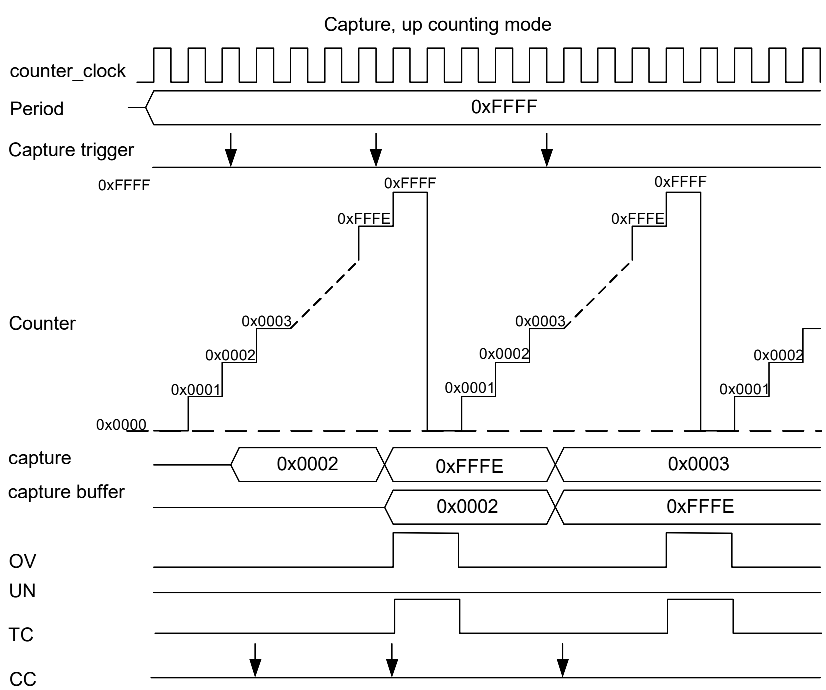
The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNT_CTRL).

Operation in capture mode occurs as follows:

- During a capture event, generated either by hardware or software, the current count register value is copied to the capture register (TCPWM_CNT_CC) and the capture register value is copied to the buffer capture register (TCPWM_CNT_CC_BUFF).
- A pulse on the CC output signal is generated when the counter value is copied to the capture register. This condition can also be used to generate an interrupt request.

Figure 16-8 illustrates the capture behavior in the up counting mode.

Figure 16-8. Timing Diagram of Counter in Capture Mode, Up Counting Mode



In the figure, observe that:

- The period register contains the maximum count value.
- Internal overflow (OV) and TC conditions are generated when the counter reaches the period value.
- A capture event is only possible at the edges or through software. Use trigger control register 1 to configure the edge detection.
- Multiple capture events in a single clock cycle are handled as:
  - Even number of capture events - no event is observed
  - Odd number of capture events - single event is observed

This happens when the capture signal frequency is greater than the counter_clock frequency.

16.3.2.3 Configuring Counter for Capture Mode

The steps to configure the counter for Capture mode operation and the affected register bits are as follows.

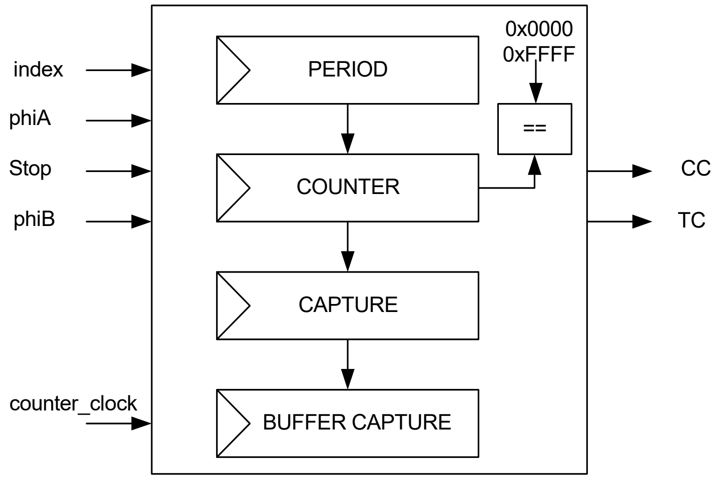
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in Table 16-6.
6. Counter can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the TCPWM_CNT_CTRL register.
7. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
8. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Stop, Capture, and Count).
9. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 147.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

16.3.3 Quadrature Decoder Mode

Quadrature decoders are used to determine speed and position of a rotary device (such as servo motors, volume control wheels, and PC mice). The quadrature encoder signals are used as phiA and phiB inputs to the decoder.

16.3.3.1 Block Diagram

Figure 16-9. Quadrature Mode Block Diagram



16.3.3.2 How It Works

Quadrature decoding only runs on counter_clock. It can operate in three sub-modes: X1, X2, and X4 modes. These encoding modes can be controlled by the QUADRATURE_MODE[21:20] field of the counter control register (TCPWM_CNT_CTRL). This mode uses double buffered capture registers.

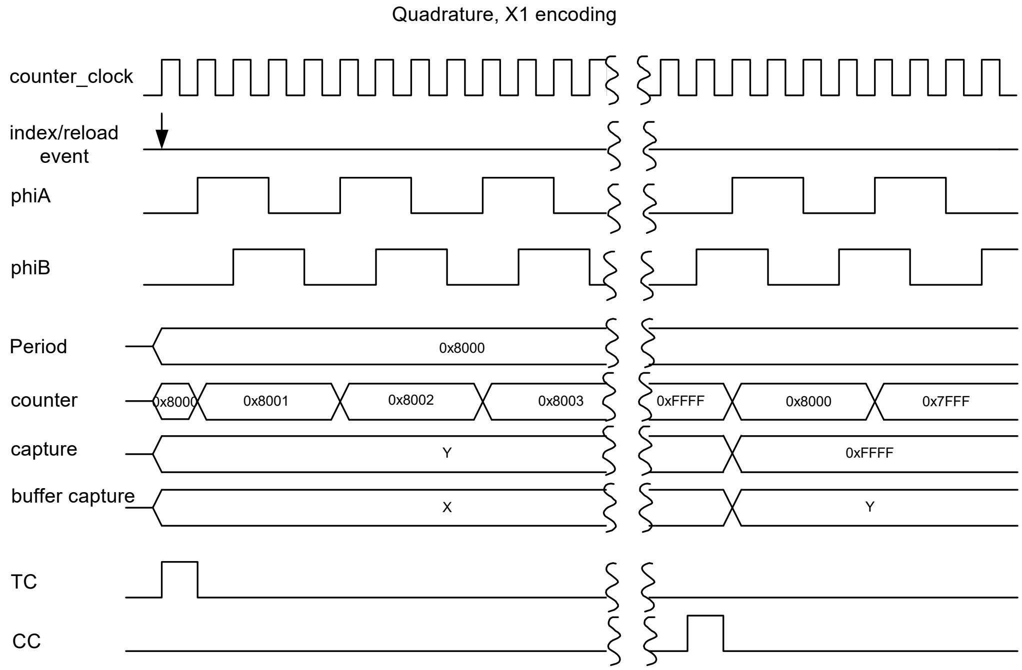
The Quadrature mode operation occurs as follows:

- Quadrature phases phiA and phiB: Counting direction is determined by the phase relationship between phiA and phiB. These phases are connected to the count and the start trigger inputs, respectively as hardware input to the decoder.
- Quadrature index signal: This is connected to the reload signal as a hardware input. This event generates a TC condition, as shown in Figure 16-10.

  On TC, the counter is set to 0x0000 (in the up counting mode) or to the period value (in the down counting mode).

  Note The down counting mode is recommended to be used with a period value of 0x8000 (the mid-point value).
- A pulse on CC output signal is generated when the count register value reaches 0x0000 or 0xFFFF. On a CC condition, the count register is set to the period value (0x8000 in this case).
- On TC or CC condition:
  - Count register value is copied to the capture register
  - Capture register value is copied to the buffer capture register
  - This condition can be used to generate an interrupt request
- The value in the capture register can be used to determine which condition caused the event and whether:
  - A counter underflow occurred (value 0)
  - A counter overflow occurred (value 0xFFFF)
  - An index/TC event occurred (value is not equal to either 0 or 0xFFFF)
- The DOWN bit field of counter status (TCPWM_CNTx_STATUS) register can be read to determine the current counting direction. Value '0' indicates a previous increment operation and value '1' indicates previous decrement operation. Figure 16-10 illustrates quadrature behavior in the X1 encoding mode.
  - A positive edge on phiA increments the counter when phiB is '0' and decrements the counter when phiB is '1'.
  - The count register is initialized with the period value on an index/reload event.
  - Terminal count is generated when the counter is initialized by index event. This event can be used to generate an interrupt.
  - When the count register reaches 0xFFFF (the maximum count register value), the count register value is copied to the capture register and the count register is initialized with period value (0x8000 in this case).

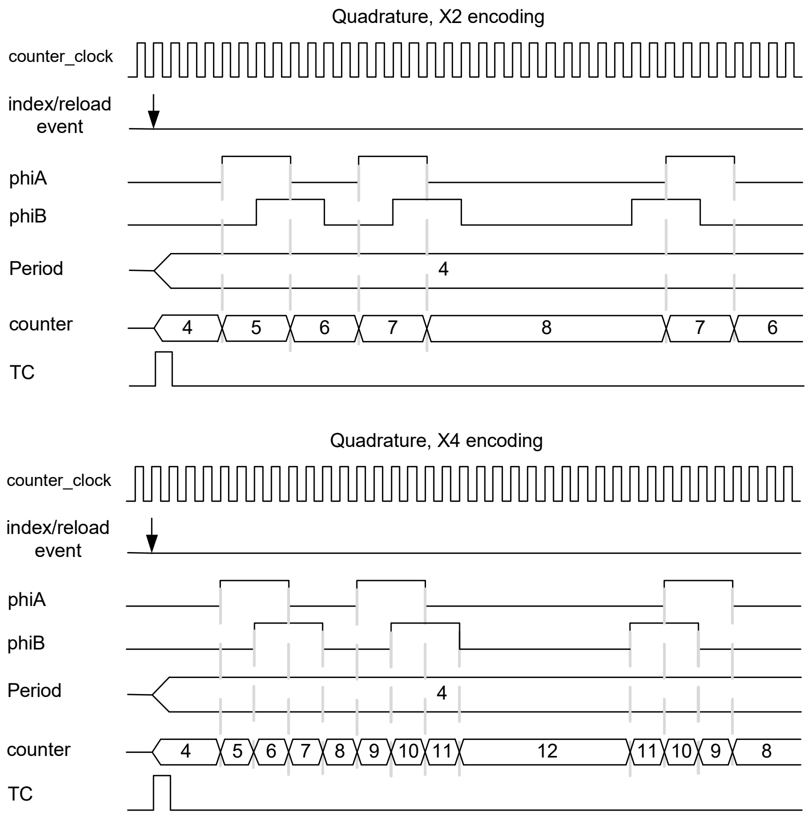
Figure 16-10. Timing Diagram for Quadrature Mode, X1 Encoding



The quadrature phases are detected on the counter_clock. Within a single counter_clock period, the phases should not change value more than once. The X2 and X4 quadrature encoding modes count twice and four times as fast as the X1 encoding mode.

Figure 16-11 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

Figure 16-11. Timing Diagram for Quadrature Mode, X2 and X4 Encoding



16.3.3.3 Configuring Counter for Quadrature Mode

The steps to configure the counter for quadrature mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of the TCPWM_CNT_CTRL register.
5. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Index and Stop).
6. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Index and Stop).
7. If required, set the interrupt upon TC or CC condition, as shown in "Interrupts" on page 147.
8. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

16.3.4 Pulse Width Modulation Mode

The PWM mode is also called the Digital Comparator mode. The comparison output is a PWM signal whose period depends on the period register value and duty cycle depends on the compare and period register values.

PWM period = (period value/counter clock frequency) in left- and right-aligned modes

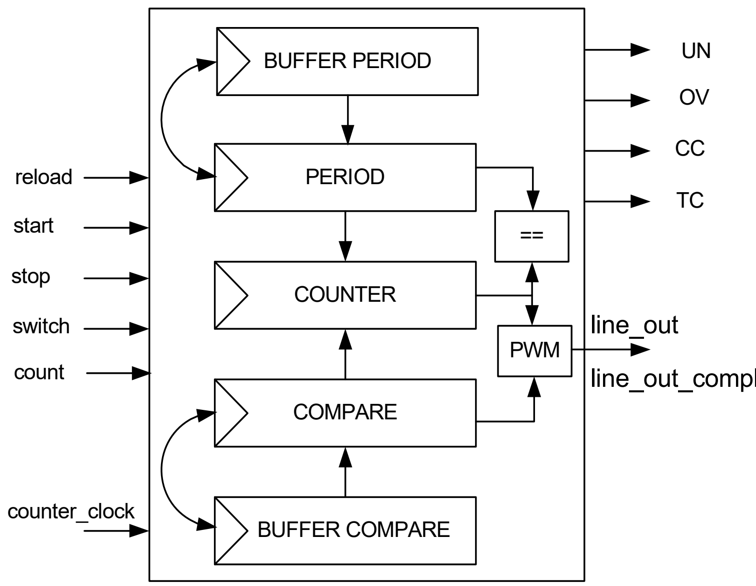
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

Duty cycle = (compare value/period value) in left- and right-aligned modes

Duty cycle = ((period value-compare value)/period value) in center-aligned mode

16.3.4.1 Block Diagram

Figure 16-12. PWM Mode Block Diagram



16.3.4.2 How It Works

The PWM mode can output left, right, center, or asymmetrically aligned PWM signals. The desired output alignment is achieved by using the counter's up, down, and up/down counting modes selected using UP_DOWN_MODE [17:16] bits in the TCPWM_CNT_CTRL register, as shown in Table 16-6.

This CC signal along with OV and UN signals control the PWM output line. The signals can toggle the output line or set it to a logic '0' or '1' by configuring the TCPWM_CNT_TR_CTRL2 register. By configuring how the signals impact the output line, the desired PWM output alignment can be obtained.

The recommended way to modify the duty cycle is:

- The buffer period register and buffer compare register are updated with new values.
- On TC, the period and compare registers are automatically updated with the buffer period and buffer compare registers when there is an active switch event. The AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register are set to '1'. When a switch event is detected, it is remembered until the next TC event. Pass through signal (selected during event detection setting) cannot trigger a switch event.
- Updates to the buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in Figure 16-14.

In the center-aligned mode, the output line is set to '0' at Terminal Count and toggled at the CC condition

At the reload event, the count register is initialized and starts counting in the appropriate mode. At every count, the count register value is compared with compare register value to generate the CC signal on match.

Figure 16-13 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 16-13. Timing Diagram for Center Aligned PWM

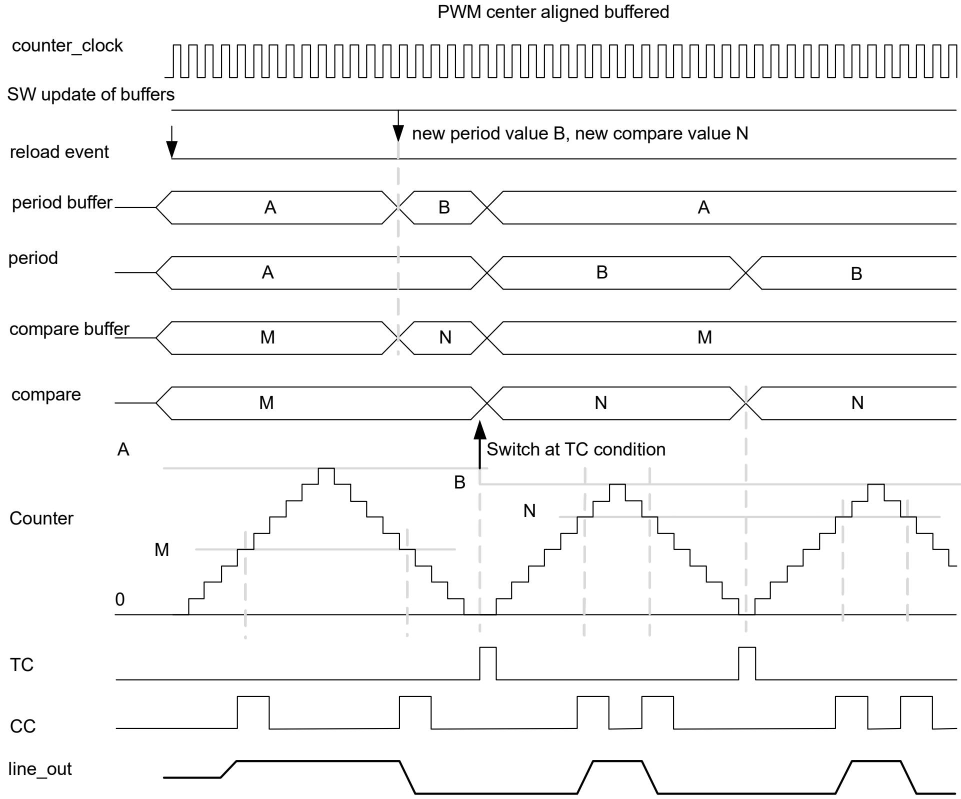
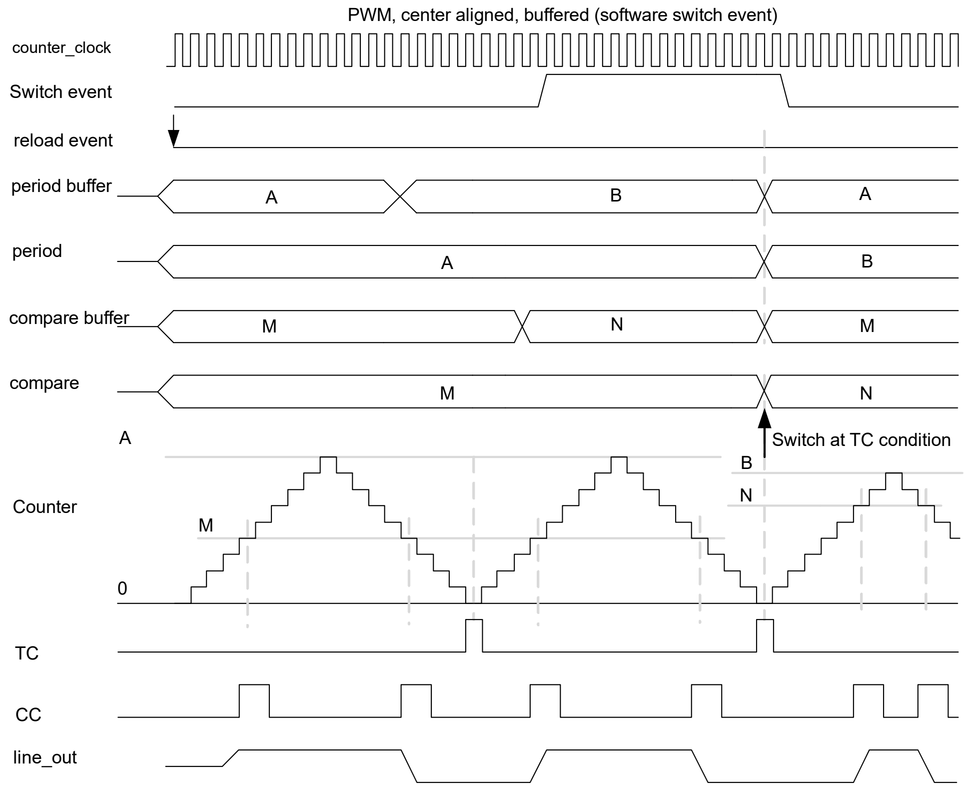


Figure 16-13 illustrates center-aligned PWM with software generated switch events:

- Software generates a switch event only after both the period buffer and compare buffer registers are updated.
- Because the updates of the second PWM pulse come late (after the terminal count), the first PWM pulse is repeated.
- Note that the switch event is automatically cleared by hardware at TC after the event takes effect.

Figure 16-14. Timing Diagram for Center Aligned PWM (software switch event



16.3.4.3 Other Configurations

- For asymmetric PWM, the up/down counting mode 1 should be used. This causes a TC when the counter reaches either ‘0’ or the period value. To create an asymmetric PWM, the compare register is changed at every TC (when the counter reaches either ‘0’ or the period value), whereas the period register is only changed at every other TC (only when the counter reaches ‘0’).
- For left-aligned PWM, use the up counting mode; configure the OV condition to set output line to '1' and CC condition to reset the output line to '0'. See Table 16-3.
- For right-aligned PWM, use the down counting mode; configure UN condition to reset output line to '0' and CC condition to set the output line to '1'. See Table 16-3.

16.3.4.4 Kill Feature

The kill feature gives the ability to disable both output lines immediately. This event can be programmed to stop the counter by modifying the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the counter control register, as shown in Table 16-7.

Table 16-7. Field Setting for Stop on Kill Feature

PWM_STOP_ON_KILL Field	Comments
0	The kill trigger temporarily blocks the PWM output line but the counter is still running.
1	The kill trigger temporarily blocks the PWM output line and the counter is also stopped.

A kill event can be programmed to be asynchronous or synchronous, as shown in Table 16-8.

Table 16-8. Field Setting for Synchronous/Asynchronous Kill

PWM_SYNC_KILL Field	Comments
0	An asynchronous kill event lasts as long as it is present. This event requires pass through mode.
1	A synchronous kill event disables the output lines until the next TC event. This event requires rising edge mode.

In the synchronous kill, PWM cannot be started before the next TC. To restart the PWM immediately after kill input is removed, kill event should be asynchronous (see Table 16-8). The generated stop event disables both output lines. In this case, the reload event can use the same trigger input signal but should be used in falling edge detection mode.

16.3.4.5 Configuring Counter for PWM Mode

The steps to configure the counter for the PWM mode of operation and the affected register bits are as follows.

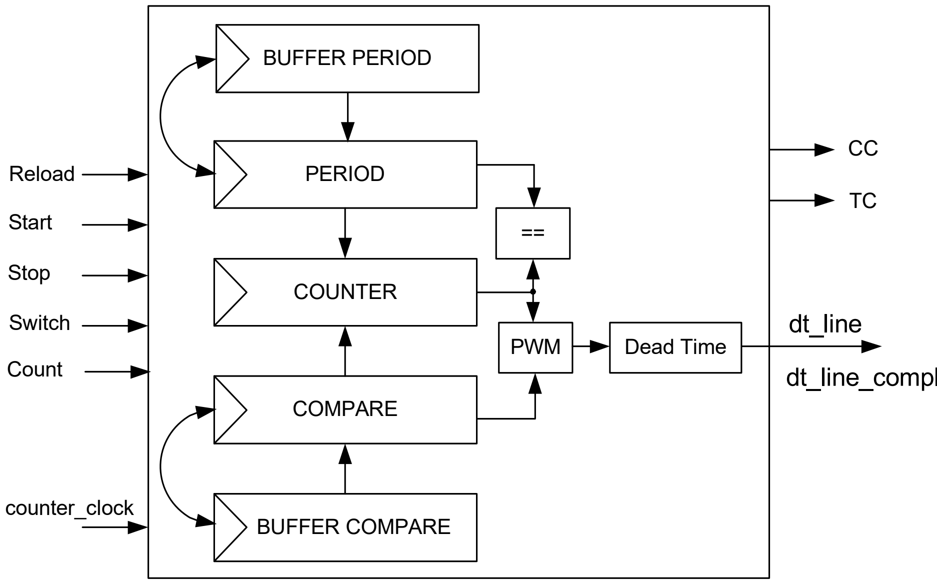
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 16-6.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 147.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

16.3.5 Pulse Width Modulation with Dead Time Mode

Dead time is used to delay the transitions of both 'line_out' and 'line_out_compl' signals. It separates the transition edges of these two signals by a specified time interval. Two complementary output lines 'dt_line' and 'dt_line_compl' are derived from these two lines. During the dead band period, both compare output and complement compare output are at logic '0' for a fixed period. The dead band feature allows the generation of two non-overlapping PWM pulses. A maximum dead time of 255 clocks can be generated using this feature.

16.3.5.1 Block Diagram

Figure 16-15. PWM-DT Mode Block Diagram



16.3.5.2 How It Works

The PWM operation with Dead Time mode occurs as follows:

- On the rising edge of the PWM line_out, depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to '0'.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period is complete, dt_line is set to '1'.
- On the falling edge of the PWM line_out depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to '0'.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period has completed, dt_line_compl is set to '1'.
- A dead band period of zero has no effect on the dt_line and is the same as line_out.
- When the duration of the dead time equals or exceeds the width of a pulse, the pulse is removed.

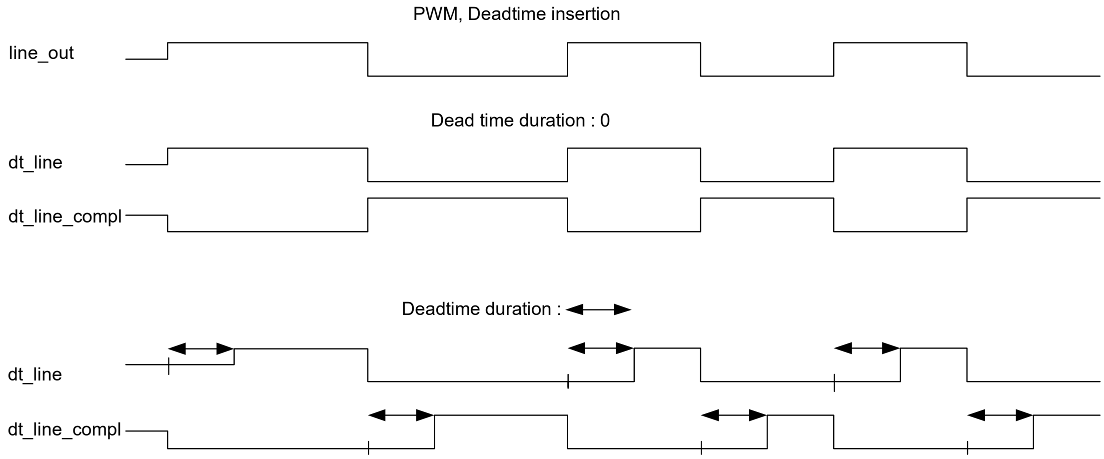
This mode follows PWM mode and supports the following features available with that mode:

- Various output alignment modes
- Two complementary output lines, dt_line and dt_line_compl, derived from PWM "line_out" and "line _out_compl", respectively
  - Stop/kill event with synchronous and asynchronous modes
  - Conditional switch event for compare and buffer compare registers and period and buffer period registers

This mode does not support clock prescaling.

Figure 16-16 illustrates how the complementary output lines "dt_line" and "dt_line_compl" are generated from the PWM output line, "line_out".

Figure 16-16. Timing Diagram for PWM, with and without Dead Time



16.3.5.3 Configuring Counter for PWM with Dead Time Mode

The steps to configure the counter for PWM with Dead Time mode of operation and the affected register bits are as follows:

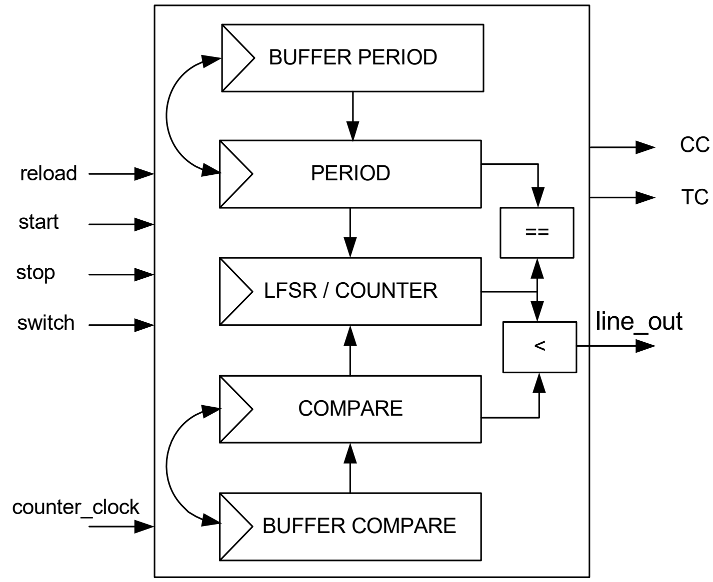
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 16-6.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required, as shown in the "Pulse Width Modulation Mode" on page 158.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in "Interrupts" on page 147.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if hardware start signal is not enabled.

16.3.6 Pulse Width Modulation Pseudo-Random Mode

This mode uses the linear feedback shift register (LFSR). LFSR is a shift register whose input bit is a linear function of its previous state.

16.3.6.1 Block Diagram

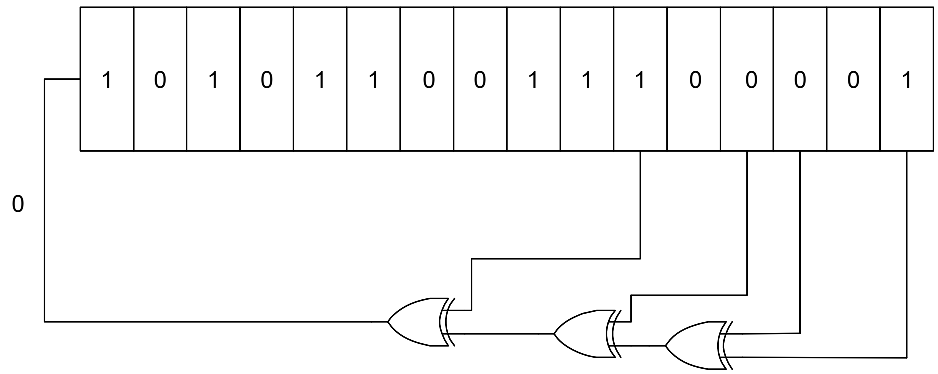
Figure 16-17. PWM-PR Mode Block Diagram



16.3.6.2 How It Works

The counter register is used to implement LFSR with the polynomial: $x^{16}+x^{14}+x^{13}+x^{11}+1$, as shown in Figure 16-18. It generates all the numbers in the range [1, 0xFFFF] in a pseudo-random sequence. Note that the counter register should be initialized with a non-zero value.

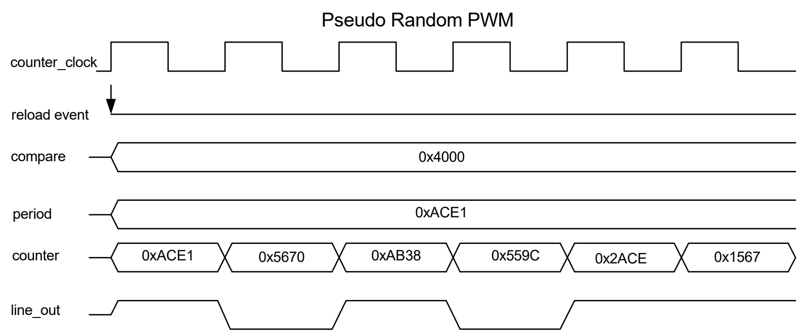
Figure 16-18. Pseudo-Random Sequence Generation using Counter Register



The following steps describe the process:

- The PWM output line, 'line_out', is driven with '1' when the lower 15-bit value of the counter register is smaller than the value in the compare register (when counter[14:0] < compare[15:0]). A compare value of '0x8000' or higher always results in a '1' on the PWM output line. A compare value of '0' always results in a '0' on the PWM output line.
- A reload event behaves similar to a start event; however, it does not initialize the counter.
- Terminal count is generated when the counter value equals the period value. LFSR generates a predictable pattern of counter values for a certain initial value. This predictability can be used to calculate the counter value after a certain amount of LFSR iterations 'n'. This calculated counter value can be used as a period value and the TC is generated after 'n' iterations.
- At TC, a switch/capture event conditionally switches the compare and period register pairs (based on the AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register).
- A kill event can be programmed to stop the counter as described in previous sections.
- One shot mode can be configured by setting the ONE_SHOT field of the counter control register. At terminal count, the counter is stopped by hardware.
- In this mode, underflow, overflow, and trigger condition events do not occur.
- CC condition occurs when the counter is running and its value equals compare value. Figure 16-19 illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50 percent duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

Figure 16-19. Timing Diagram for Pseudo-Random PWM



A capture/switch input signal may switch the values between the compare and compare buffer registers and the period and period buffer registers. This functionality can be used to modulate between two different compare values using a trigger input signal to control the modulation.

Note Capture/switch input signal can only be triggered by an edge (rising, falling, or both). This input signal is remembered until the next terminal count.

16.3.6.3 Configuring Counter for Pseudo-Random PWM Mode

The steps to configure the counter for pseudo-random PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to COUNTER_ENABLED of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNT_PERIOD register and buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
6. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in "Interrupts" on page 147.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

16.4 TCPWM Registers

Table 16-9. List of TCPWM Registers

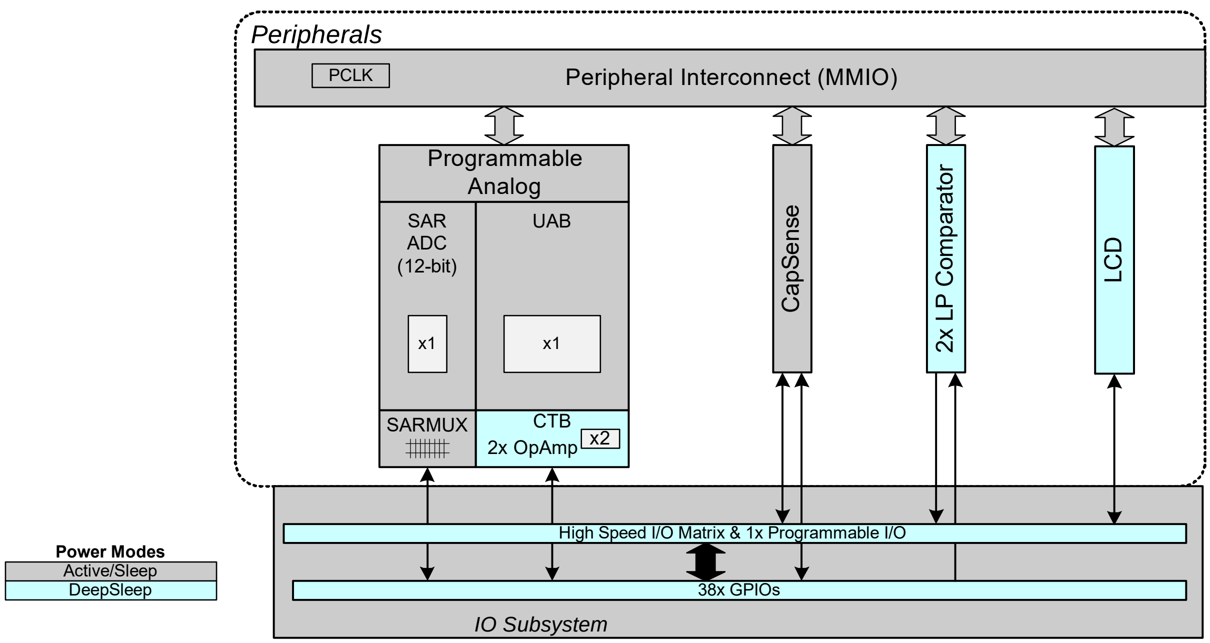
Register	Comment	Features
TCPWM_CTRL	TCPWM control register	Enables the counter block
TCPWM_CMD	TCPWM command register	Generates software events
TCPWM_INTR_CAUSE	TCPWM counter interrupt cause register	Determines the source of the combined interrupt signal
TCPWM_CNT_CTRL	Counter control register	Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock pre-scaling, and counting direction
TCPWM_CNT_STATUS	Counter status register	Reads the direction of counting, dead time duration, and clock pre-scaling; checks if the counter is running
TCPWM_CNT_COUNTER	Count register	Contains the 16-bit counter value
TCPWM_CNT_CC	Counter compare/capture register	Captures the counter value or compares the value with counter value
TCPWM_CNT_CC_BUFF	Counter buffered compare/capture register	Buffer register for counter CC register; switches period value
TCPWM_CNT_PERIOD	Counter period register	Contains upper value of the counter
TCPWM_CNT_PERIOD_BUFF	Counter buffered period register	Buffer register for counter period register; switches compare value
TCPWM_CNT_TR_CTRL0	Counter trigger control register 0	Selects trigger for specific counter events
TCPWM_CNT_TR_CTRL1	Counter trigger control register 1	Determine edge detection for specific counter input signals
TCPWM_CNT_TR_CTRL2	Counter trigger control register 2	Controls counter output lines upon CC, OV, and UN conditions
TCPWM_CNT_INTR	Interrupt request register	Sets the register bit when TC or CC condition is detected
TCPWM_CNT_INTR_SET	Interrupt set request register	Sets the corresponding bits in interrupt request register
TCPWM_CNT_INTR_MASK	Interrupt mask register	Mask for interrupt request register
TCPWM_CNT_INTR_MASKED	Interrupt masked request register	Bitwise AND of interrupt request and mask registers

Section E: Analog System

This section encompasses the following chapter:

Top Level Architecture

Analog System Block Diagram



17. Programmable Reference Block (PRB)

The programmable reference block (PRB) provides various reference voltages without the need for a precision DAC.

17.1 Features

The PRB has the following features:

- Four different output voltages
- Reference voltage can be derived from either a 1.2-V bandgap reference voltage or from the analog supply voltage (VDDA)

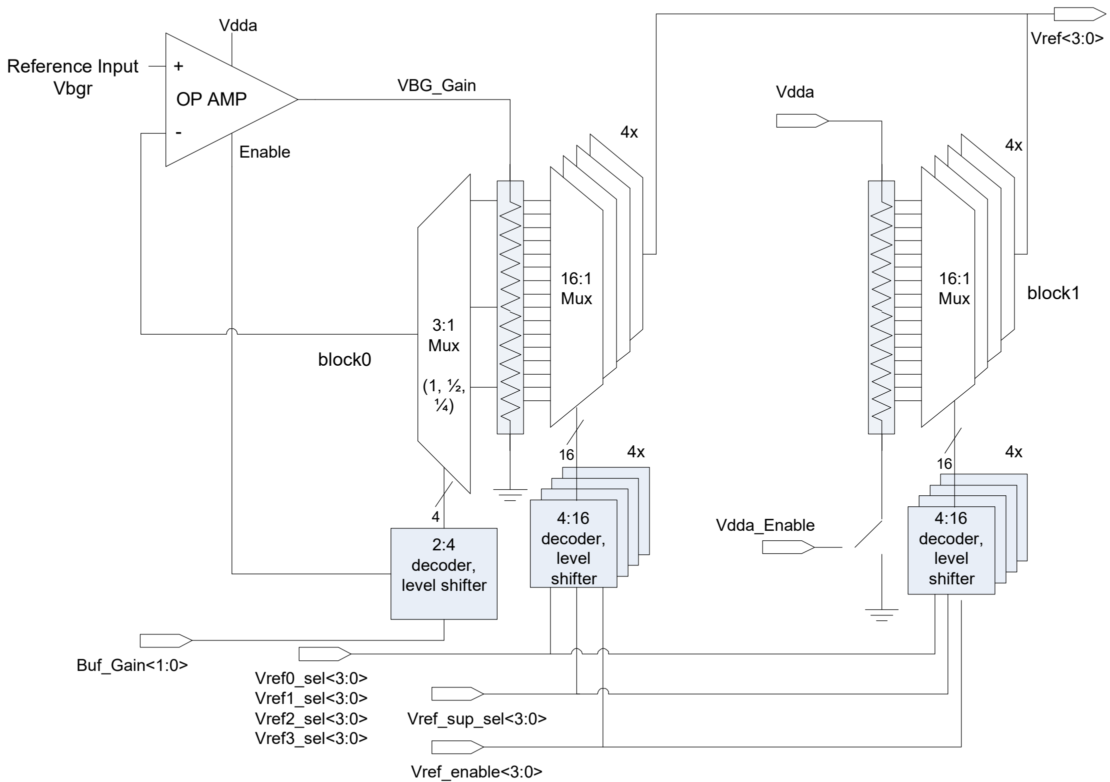
17.2 Architecture

Figure 17-1 shows the architecture of the PRB. The PRB can produce output voltages derived from the bandgap reference or VDDA. To run a reference voltage, such as the bandgap, a buffer opamp is required internal to the PRB, which guarantees high input impedance into the PRB. The PRB provides up to four different output voltages. This is achieved by using two R-strings, each having four 16:1 muxes connected to them. The quad muxes are connected together at their respective outputs onto the four VREF bus lines. One of the R-strings operates off the buffered bandgap (or 2x/4x multiples of bandgap voltage); the other R-string operates off VDDA.

When decoding the outputs of the 4-bit DACs (combination of 4:16 decoders, 16:1 muxes, resistor string, and associated source), the source (VBG_Gain or VDDA) for deriving the required reference is selected. In addition, it is possible to let the VrefBus lines floating (disconnected). This is important so that external voltages can drive the VrefBus.

The PRB has an OFF mode in which the reference buffer is switched off and the VDDA string can be disconnected.

Figure 17-1. PRB Architecture



17.2.1 Reference Options

Table 17-1 shows the available reference options and ranges using PRB.

Table 17-1. Reference Options

Reference Source	Output Voltage	Usage Example
VDDA	VDDA/16 – VDDA, 16 tap points	As reference for CTB opamp, comparator thresholds, and ratiometric biasing for internal or external circuits.
VBGR	VBGR/16 – VBGR, or VBGR/8 – 2VBGR, or VBGR/4 – 4VBGR, 16 tap points	As UAB reference (for DAC, ADC, and so on), comparator trip points, and bias.

Table 17-2 shows the resistor tap and associated ratio to generate the required voltage reference.

Table 17-2. Resistor Tap and Ratio

Resistor Tap	Ratio
16	1
15	0.9375
14	0.875
13	0.8125
12	0.75
11	0.6875
10	0.625
9	0.5625

Table 17-2. Resistor Tap and Ratio

Resistor Tap	Ratio
8	0.5
7	0.4375
6	0.375
5	0.3125
4	0.25
3	0.1875
2	0.125
1	0.0625

17.2.2 Power Modes

The PRB will be on in Active and Sleep modes; in Deep-Sleep mode, this block will be powered down. Power-down signals are provided for PRB-buffer and R-string separately. For the R-string, a series switch must be opened to kill any current flowing through the resistors. The power spent on the PRB buffer should be less than 50 µA.

17.3 Registers Summary

Table 17-3. PRB Registers

Register	Description
PRB_CTRL	Global PRB Control Register
PRB_REF0	VREF0 Control Register
PRB_REF1	VREF1 Control Register
PRB_REF2	VREF2 Control Register
PRB_REF3	VREF3 Control Register
PRB_TRIM	PRB Buffer OpAmp Trim Register

18. Continuous Time Block (CTB)

The Continuous Time Block (CTB) provides operational amplifiers (opamps) inside the chip for use in continuous-time signal chains. Each CTB block includes two identical opamps, a switch matrix for input/output routing, and the resistor array for each opamp gain setting. The PSoC Analog Coprocessor has two such CTB blocks.

18.1 Features

The opamps in the CTB have the following features:

- Opamps with built-in resistor array for amplifier implementation with a configurable gain up to 32x
- Programmable switch matrix for flexible routing and to support different functions such as amplifier, trans-impedance amplifier (TIA), filter, and comparator
- Programmable power, compensation, and output drive strength
- 1 mA or 10 mA selectable output current drive capability
- Comparator mode with optional 10-mV hysteresis
- Deep-Sleep low-power mode operation with reduced specifications

18.2 Architecture

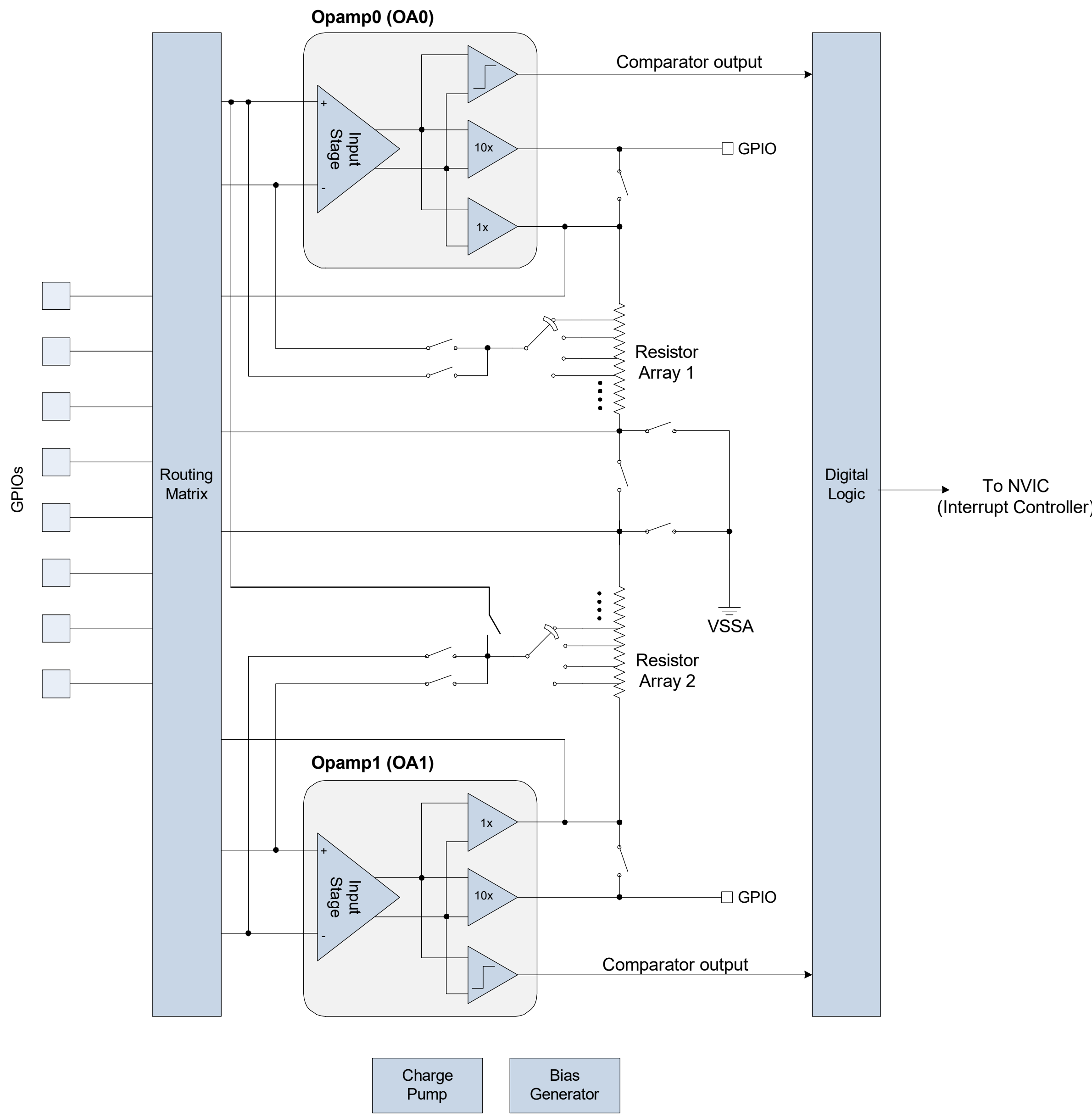
Figure 18-1 shows the block diagram of the CTB. A CTB consists of two opamps – OA0 and OA1. Each opamp is capable of driving a GPIO using the 10x driver, connecting to the internal peripherals using the 1x driver, and acting as a comparator. CTBs are generally used to build programmable gain amplifiers. This functionality makes use of the resistor array available for each opamp, which are independently controlled. Using a single CTB, two non-inverting programmable gain amplifiers can be built with a maximum gain of 32x.

The routing matrix enables connection of opamp inputs and outputs to internal resources such as Programmable Reference Block (PRB) and Analog Route (AROUTE) using the switches. AROUTE is used to make connections between CTBs, UAB, SAR ADC, and PRB. This routing is useful in various applications; for example, the CTB output is routed to the UAB for filtering or SAR ADC for measurement. Note that the 10x output of the opamp is routed to the pin directly without using the routing matrix switches. This provides a least resistance path for the opamp output to connect to the pin.

The opamps can also be operated in the comparator mode with the two inputs connected via the routing matrix. When comparator mode is enabled, the 10x and 1x drivers are disabled. The comparator outputs of all the CTBs are ORed and connected to the interrupt controller for generating the interrupts.

Charge pump and bias generator are also present in each CTB. The use of charge pump allows the opamp input voltage in excess of VDDA–1.2 V; whereas, the bias generator controls the power level of the opamps in the CTB.

Figure 18-1. CTB Block Diagram



The details of each function block are described in the following sections.

18.2.1 Opamps

The two opamps present in the CTB can be operated independently or as a pair. Each opamp has two output drivers – 1x for driving internal peripherals with a maximum current of 1 mA and 10x for driving to the pin with a maximum current of 10 mA. The CTB can be enabled by setting the ENABLED bit in the CTBx_CTB_CTRL register to ‘1’, where x is the CTB number, which can be either 0 or 1. If the ENABLED bit is set to ‘0’, the CTB is put to power down mode. CTB also has an additional control: the DEEPSLEEP_ON bit in the CTBx_CTB_CTRL register to keep it functional in Deep-Sleep mode.

Table 18-1. CTB Enable Control

CTBx_CTB_CTRL [31] - ENABLED	CTBx_CTB_CTRL[30] - DEEPSLEEP_ON	Description
0	0	CTB is disabled
0	1	CTB is disabled
1	0	CTB is enabled, but functional only in Active and Sleep mode of the device.
1	1	CTB is enabled, and functional in Active, Sleep, and Deep-Sleep mode of the device.

18.2.2 Power Levels

Four power levels can be set for each opamp using the OAy_PWR_MODE bits of the CTBx_OA_RESy_CTRL register, where y is the opamp number in the CTBx; it can be either 0 or 1. These bits are used to vary the bias currents, which results in achieving optimum bandwidth and power consumption. Higher the power level, more the bandwidth and power consumption.

Table 18-2. Opamp Power Setting

Power Setting (OAy_PWR_MODE[1:0])		Description
0	0	OFF
0	1	Low Power
1	0	Medium Power
1	1	High Power

18.2.3 Output Driver

The output driver of the opamps (either 1x or 10x) are selected using the OAy_DRIVE_STR_SEL bit of the CTBx_OA_RESy_CTRL register. When the 10x output driver is selected, the output of opamps is directly driven to the pin.

Table 18-3. Opamp Output Driver Selection

CTBx_OA_RESy_CTRL[2] - OAy_DRIVE_STR_SEL	Selected Driver
0	1x
1	10x

18.2.4 Charge Pump

The CTB has a charge pump circuitry, which is shared between the two opamps. This circuit allows higher opamp input range when enabled. If disabled, the input is limited to (VDDA – 1.5 V). The charge pump is enabled by setting the OAy_PUMP_EN bit of the CTBx_OA_RESy_CTRL register to ‘1’.

18.2.5 Offset Trimming

The opamp input offset voltage is trimmed using CTBx_OAy_OFFSET_TRIM and CTBx_OAy_SLOPE_OFFSET_TRIM registers. The CTBx_OAy_OFFSET_TRIM register is used to correct the offset at a given operating conditions (VDDA and ambient temperature); the CTBx_OAy_SLOPE_OFFSET_TRIM register is used to correct the drift in offset due to temperature. Register details are given in Table 18-4 and Table 18-5. Note that the MSB bit defines the direction of trimming: 0 for negative direction and 1 for positive direction. The trim steps are linear in either direction.

Table 18-4. Opamp Input Offset Voltage Trimming

CTBx_OAy_OFFSET _TRIM[5]	CTBx_OAy_OFFSET _TRIM[4:0]	Description
0	00000	Negative Trim Direction - Min Setting
0	11111	Negative Trim Direction - Max Setting
1	00000	Positive Trim Direction - Min Setting
1	11111	Positive Trim Direction - Max Setting

Table 18-5. Opamp Input Offset Voltage Drift Trimming

CTBx_OAy_SLOPE_ OFFSET_TRIM[5]	CTBx_OAy_SLOPE_ OFFSET_TRIM[4:0]	Description
0	00000	Negative Trim Direction - Min Setting
0	11111	Negative Trim Direction - Max Setting
1	00000	Positive Trim Direction - Min Setting
1	11111	Positive Trim Direction - Max Setting

18.2.6 Programmable Compensation

Each opamp offers four compensation settings – High, Medium, Low, and OFF. This programmability provides control over stability of the opamp circuit. Higher the compensation, lower the bandwidth and greater the stability of the opamp circuit. The CTBx_OAy_COMP_TRIM register is used to configure the compensation as shown in Table 18-6.

Table 18-6. Opamp Compensation

CTBx_OAy_COMP_TRIM [1:0]	Setting	Description
00	OFF	No compensation (highest gain-bandwidth product)
01	Low	Lowest compensation
10	Medium	Medium compensation
11	High	Largest compensation (lowest gain-bandwidth product)

18.2.7 Comparator Mode

Besides operating for analog function, the opamps can also be operated as a comparator with digital output. Setting the OAy_COMP_EN bit in the CTBx_OA_RESy_CTRL register to ‘1’ configures the opamp in the comparator mode. The opamp compensation is also turned OFF to get a fast response. The hysteresis of 10 mV can be enabled by setting the OAy_HYST_EN bit of the CTBx_OA_RESy_CTRL register to ‘1’. The comparator mode of the opamp provides a low input offset voltage alternative to the on-chip dedicated low-power comparators. Note that the comparator mode can be used only to generate the interrupt and check the comparator status in firmware. The digital output cannot be used for any other purpose.

18.2.8 Resistor Arrays

Each CTB has two resistor arrays, one for each opamp. The top end of the resistor array is connected at the output of the opamp. The bottom end is configurable – it can be either connected to VSSA, bottom end of the other resistor array in the CTB, or routed through the routing matrix. Each array has several wiper settings (tap points) with a gain up to 32x with each array programmable independent of the other. The resistor tap is selected using the RESy_TAP bits of the CTBx_OA_RESy_CTRL register. Table 18-7 shows the selected gain based on RESy_TAP bits.

Table 18-7. Resistor Tap Selection and Gain

CTBx_OA_RESy_CTRL[19:16] - RESy_TAP bits	Gain
0000	1.0
0001	1.42
0010	2
0011	2.78
0100	4
0101	5.82
0110	8
0111	10.67
1000	16
1001	21.33
1010	32

Besides the 11 tap points on the resistor array, there are another 11 appropriately selected tap points to match the gain if the resistor array is to be made upside down. This swap feature is useful in creating the two-opamp instrumentation amplifier as described in Example Configurations on page 180. Swap is activated by writing ‘1’ into the RESy_SWAP bit of the CTBx_OA_RESy_CTRL register.

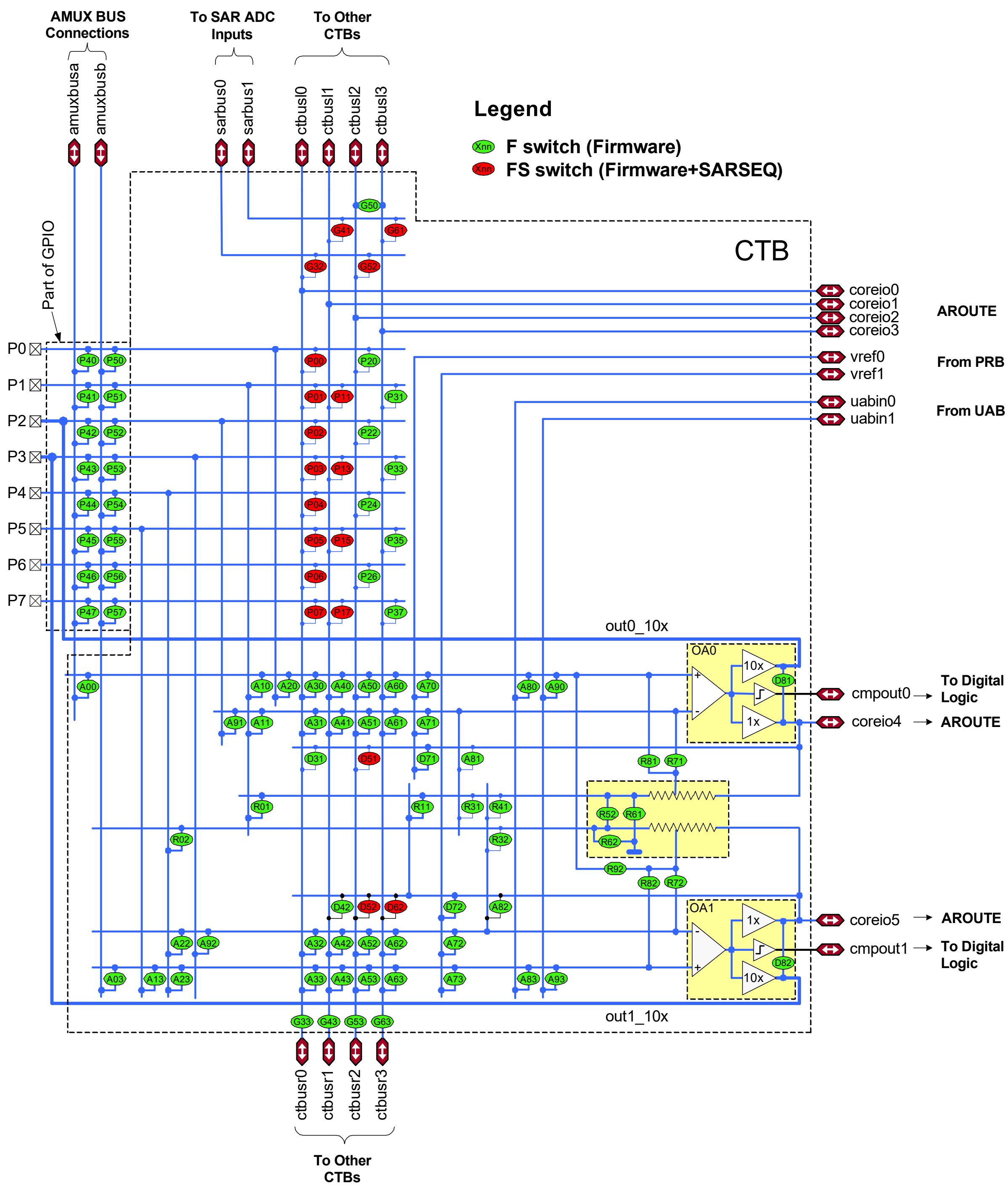
18.2.9 Routing

The flexible routing matrix facilitates the connection of inputs and outputs of opamps to the GPIOs; other peripherals such as UAB, SAR ADC, and PRB; routing resources such as analog mux bus; and opamps of the same or other CTBs. Figure 18-2 shows the routing diagram of the CTB.

Following are the key aspects of routing:

- All the switches are firmware (FW) controllable. A subset is FW and SAR Sequencer (SARSEQ) controllable. See the legend in Figure 18-2.
- The analog mux bus is primarily used for CapSense. However, the CTB can tap onto this bus, which enables it to connect to any GPIO in the device.
- The CTB can connect to the SAR mux buses – sarbus0 and sarbus1. One application of this feature is to connect the CTB opamp output to the SAR ADC input using SARSEQ.
- The CTB should be enabled using the ENABLED parameter of the CTBx_CTB_CTRL register to control the switches outside the yellow box (opamps). If the CTB is disabled, all the switches will be forced open, irrespective of the switch settings or SARSEQ controls.

Figure 18-2. CTB Analog Routing



The switches for opamp connections are controlled using the CTBx_OAy_SW register. Writing '1' to this register closes the switch and writing '0' opens the switch. Table 18-8 and Table 18-9 provide the register details for OA0 and OA1 respectively.

Table 18-8. Opamp 0 Switch Control Register

Bit Index of CTBx_OA0_SW	Switch	Connection
0	A00	Opamp0 positive terminal to AMUXBUS-A
1	A10	Opamp0 positive terminal to port pin P1[1] or P2[1]
2	A20	Opamp0 positive terminal to port pin P1[0] or P2[0]
3	A30	Opamp0 positive terminal to ctbbus0
4	A40	Opamp0 positive terminal to ctbbus1
5	A50	Opamp0 positive terminal to ctbbus2
6	A60	Opamp0 positive terminal to ctbbus3
7	A70	Opamp0 positive terminal to vref0
8	A80	Opamp0 positive terminal to uabin0vdacin0
9	A90	Opamp0 positive terminal to uabin1vdacin1
11	A11	Opamp0 negative terminal to port pin P1[1] or P2[1]
13	A31	Opamp0 negative terminal to ctbbus0
14	A41	Opamp0 negative terminal to ctbbus1
15	A51	Opamp0 negative terminal to ctbbus2
16	A61	Opamp0 negative terminal to ctbbus3
17	A71	Opamp0 negative terminal to vref0
18	A81	Opamp0 negative terminal to Opamp0 bottom
19	A91	Opamp0 negative terminal to port pin P1[2] or P2[2]
20	D31	Opamp0 output to ctbbus0
21	D51	Opamp0 output to ctbbus2
22	D71	Opamp0 output to vref0
23	D81	Opamp0 output switch to short 1x with 10x drive
24	R01	Resistor0 bottom to Pin P1
25	R11	Resistor0 bottom to Opamp1 output
26	R31	Resistor0 bottom to Opamp0 negative terminal
27	R41	Resistor0 bottom to Opamp1 negative terminal
28	R61	Resistor0 bottom to VSSA
29	R71	Resistor0 tap to Opamp0 negative terminal
30	R81	Resistor0 tap to Opamp0 positive terminal

Table 18-9. Opamp 1 Switch Control Register

Bit Index of CTBx_OA1_SW	Switch	Connection
0	A03	Opamp1 positive terminal to AMUXBUS-B
1	A13	Opamp1 positive terminal to port pin P1[5] or P2[5]
2	A23	Opamp1 positive terminal to port pin P1[4] or P2[4]
3	A33	Opamp1 positive terminal to ctbbus0
4	A44	Opamp1 positive terminal to ctbbus1
5	A53	Opamp1 positive terminal to ctbbus2
6	A63	Opamp1 positive terminal to ctbbus3
7	A73	Opamp1 positive terminal to vref1
8	A83	Opamp1 positive terminal to uabin0vdacin0
9	A93	Opamp1 positive terminal to uabin1
11	A22	Opamp1 negative terminal to port pin P1[4] or P2[4]
13	A32	Opamp1 negative terminal to ctbbus0
14	A42	Opamp1 negative terminal to ctbbus1

Table 18-9. Opamp 1 Switch Control Register

Bit Index of CTBx_OA1_SW	Switch	Connection
15	A52	Opamp1 negative terminal to ctbbus2
16	A62	Opamp1 negative terminal to ctbbus3
17	A72	Opamp1 negative terminal to vref1
18	A82	Opamp1 negative terminal to Opamp1 bottom
19	A92	Opamp1 negative terminal to port pin P1[3] or P2[3]
20	D42	Opamp1 output to ctbbus1
21	D52	Opamp1 output to ctbbus2
22	D62	Opamp1 output to ctbbus3
23	D72	Opamp1 output to vref1
24	D82	Opamp1 output switch to short 1x with 10x drive
25	R02	Resistor1 bottom to pin P4
26	R32	Resistor1 bottom to Opamp1 negative terminal
27	R52	Resistor1 bottom to Resistor0 bottom
28	R62	Resistor1 bottom to VSSA
29	R72	Resistor1 tap to Opamp1 negative terminal
30	R82	Resistor1 tap to Opamp1 positive terminal
31	R92	Resistor1 tap to Opamp0 positive terminal

Note that the dedicated connections for CTB0 are available on Port 1 and for CTB1 on Port 2. For example, writing ‘1’ to CTB0_OA0_SW[1] connects the OA0 positive terminal to P1[1]; writing ‘1’ to CTB1_OA0_SW[1] connects the OA1 positive terminal to P2[1]. Similarly for OA1, writing ‘1’ to CTB0_OA1_SW[1] connects the OA1 positive terminal to P1[5]; writing ‘1’ to CTB1_OA1_SW[1] connects the OA1 positive terminal to P2[5].

CTB bus (ctbus0 to ctbus3) switches are controlled by writing to the CTBx_CTBBUS_SW register. Writing ‘1’ closes the switch and ‘0’ opens the switch.

Table 18-10. Firmware Control

Bit Index of CTBx_CTBBUS_SW	Switch	Connection
0	P00	Ctbus0 to port pin P1[0] or P2[0]
1	P01	Ctbus0 to port pin P1[1] or P2[1]
2	P02	Ctbus0 to port pin P1[2] or P2[2]
3	P03	Ctbus0 to port pin P1[3] or P2[3]
4	P04	Ctbus0 to port pin P1[4] or P2[4]
5	P05	Ctbus0 to port pin P1[5] or P2[5]
6	P06	Ctbus0 to port pin P1[6] or P2[6]
7	P07	Ctbus0 to port pin P1[7] or P2[7]
8	P11	Ctbus1 to port pin P1[1] or P2[1]
9	P13	Ctbus1 to port pin P1[3] or P2[3]
10	P15	Ctbus1 to port pin P1[5] or P2[5]
11	P17	Ctbus1 to port pin P1[7] or P2[7]
12	P20	Ctbus2 to port pin P1[0] or P2[0]
13	P22	Ctbus2 to port pin P1[2] or P2[2]
14	P24	Ctbus2 to port pin P1[4] or P2[4]
15	P26	Ctbus2 to port pin P1[6] or P2[6]

Table 18-10. Firmware Control

Bit Index of CTBx_CTBBUS_SW	Switch	Connection
16	P31	Ctbus3 to port pin P1[1] or P2[1]
17	P33	Ctbus3 to port pin P1[3] or P2[3]
18	P35	Ctbus3 to port pin P1[5] or P2[5]
19	P37	Ctbus3 to port pin P1[7] or P2[7]
20	G33	Ctbus0 to ctbbus0 of next CTB
21	G43	Ctbus1 to ctbbus1 of next CTB
22	G53	Ctbus2 to ctbbus2 of next CTB
23	G63	Ctbus3 to ctbbus3 of next CTB
24	G32	Ctbus0 to sarbus0
25	G41	Ctbus1 to sarbus1
26	G52	Ctbus2 to sarbus0
27	G61	Ctbus3 to sarbus1
28	G50	Ctbus2 to ctbus3

The hardware-controlled switches are enabled using the CTBx_CTB_SW_HW_CTRL register. Writing ‘1’ to this register allows the hardware to control the switches. Writing ‘0’ disables the hardware control. Note that if the CTBx_OAy_SW or CTBx_CTBBUS_SW register is set to control the switch in firmware, setting the CTBx_CTB_SW_HW_CTRL will have no effect. For the switch to be hardware-controlled, corresponding bits for firmware control should be 0. Table 18-11 provides the details of the CTBx_CTB_SW_HW_CTRL register.

Table 18-11. HW/SW Control Selection

Bit Index of CTBx_CTB_SW_HW_CTRL	Switch
0	P00
1	P01 and P11
2	P02
3	P03 and P13
4	P04
5	P05 and P15
6	P06
7	P07 and P17
8	P20
9	P31
10	P22 and D51
11	P33, D52 and D62
12	P24
13	P35
14	P26
15	P37
16	sarbus switches G32 and G52
17	sarbus switches G41 and G61

Hardware trigger is selected using the TR_OUT_CTLx register. The trigger mux group is used to select the control source of the hardware controlled switches.

Table 18-12. Trigger Source Selection

TR_OUT_CTLx[4:0]	Trigger Source
0	Hardwired to 0 (for firmware trigger)
1	TCPWM 4 Line
2	TCPWM 4 Compl
3	TCPWM 5 Line
4	TCPWM 5 Compl
5	TCPWM 6 Line
6	TCPWM 6 Compl
7	TCPWM 7 Line
8	TCPWM 7 Compl
9	CTB0 Cmp0
10	CTB0 Compl Cmp0
11	CTB0 Cmp1
12	CTB0 Compl Cmp1
13	CTB1 Cmp0
14	CTB1 Compl Cmp0
15	CTB1 Cmp1
16	CTB1 Compl Cmp1

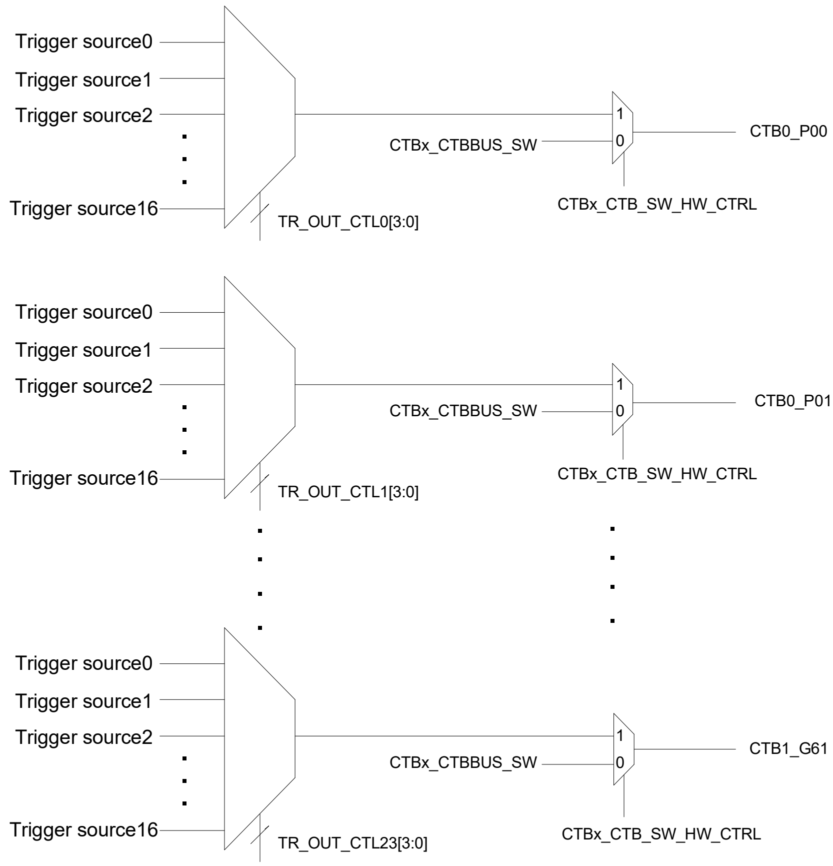
Table 18-13 shows the mapping between trigger mux and the CTB switch.

Table 18-13. Trigger Mux Mapping

Trigger Mux	Hardware-controllable CTB Switch
TR_OUT_CTL0	CTB0_P00
TR_OUT_CTL1	CTB0_P01
TR_OUT_CTL2	CTB0_P02
TR_OUT_CTL3	CTB0_P03
TR_OUT_CTL4	CTB0_P04
TR_OUT_CTL5	CTB0_P05
TR_OUT_CTL6	CTB0_P06
TR_OUT_CTL7	CTB0_P07
TR_OUT_CTL8	CTB1_P00
TR_OUT_CTL9	CTB1_P01
TR_OUT_CTL10	CTB1_P02
TR_OUT_CTL11	CTB1_P03
TR_OUT_CTL12	CTB1_P04
TR_OUT_CTL13	CTB1_P05
TR_OUT_CTL14	CTB1_P06
TR_OUT_CTL15	CTB1_P07
TR_OUT_CTL16	CTB0_G32
TR_OUT_CTL17	CTB0_G41
TR_OUT_CTL18	CTB0_G52
TR_OUT_CTL19	CTB0_G61
TR_OUT_CTL20	CTB1_G32
TR_OUT_CTL21	CTB1_G41
TR_OUT_CTL22	CTB1_G52
TR_OUT_CTL23	CTB1_G61

Figure 18-3 shows the connection from trigger source to the CTB switch.

Figure 18-3. Trigger Mux Connection



18.2.10 Low-Power Behavior

The CTB can operate in Sleep and Deep-Sleep modes. CTBs are fully functional in Sleep mode, similar to Active mode. There are, however, some restrictions in Deep-Sleep mode. Following are the key considerations when the CTB is operated in Deep-Sleep mode:

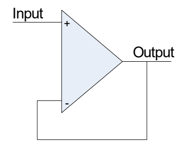
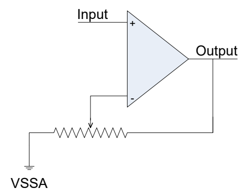
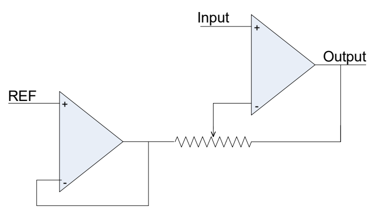
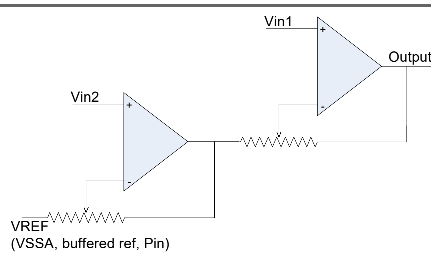
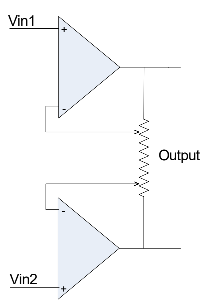
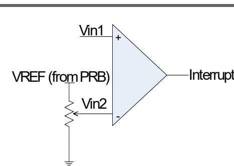
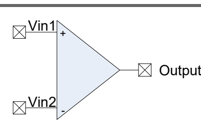
- Only the 10x driver is functional.
- Resistor arrays can be used. However, this requires closing the D81 or D82 switch (see Figure 18-2) because the 1x driver is not functional.
- All CTB switches are functional given that the CTB is enabled. Connections to other CTBs, SARMUX, and AROUTE can also be enabled for Deep-Sleep operation.
- Connections via AMUXBUS are not available.
- Charge pump is not active because of the absence of the clock. Thus, the input to the opamp is limited to (VDDA –1.5 V).
- For VDDA > 2.7 V, CTB is functional across the device operating temperature range specification with opamp high, medium, and low power modes.

  For 2.4 V < VDDA < 2.7 V, CTB functionality is limited to 0 to 85 °C temperature range with only low-power mode.
- Comparator interrupt can be used to wake up the device.
- The performance is reduced in Deep-Sleep mode. Switch resistances will be relatively higher in Deep-Sleep mode when compared to Sleep or Active mode.

18.3 Example Configurations

CTBs can be configured for multiple functions; some of these functions are described in Table 18-14.

Table 18-14. CTB Configurations

Configuration	Schematic	Details
Buffer		To achieve unity gain, the resistor array tap selection bits in the CTBx_OA_RESy_CTRL register is set to 0.
Non-Inverting Amplifier (GND referenced)		This configuration uses the internal resistor array to set the gain. The bottom end of the resistor array is connected to VSSA.
Non-Inverting Amplifier (non-GND referenced)		In this configuration, the non-inverting amplifier gets the buffered reference. The bottom end of the resistor array is connected to the output of another opamp.
Two Opamp Instrumentation Amplifier		This configuration uses two or three opamps depending on how the reference is connected. The bottom end of the resistor array of Vin2 opamp can be connected to VSSA, another opamp in buffer mode, or a pin. Note that in this configuration, the swap feature is used as discussed in Resistor Arrays on page 174.
Differential Amplifier (Differential Output)		In this configuration, the two resistor arrays of a CTB are connected together. The differential output can be directly connected to differential inputs of the SAR ADC for measurement.
Comparator		In this mode, the reference from the PRB can be used as a trip point in the comparator.
Opamps with inputs and output connected to pins		In this configuration, opamp terminals are brought out to pins for external connections. This configuration can be used to implement functions using external passive components such as TIA, integrator, filter, and window comparator.

18.4 Registers Summary

Table 18-15. CTB Registers

Register	Description
CTBx_CTB_CTRL	Global CTB Power Control Register
CTBx_OA_RESy_CTRL	CTBx - Opamp y and Resistor y Control Register
CTBx_COMP_STAT	CTBx - Comparator Status
CTBx_INTR	CTBx - Interrupt Request Register
CTBx_INTR_MASK	CTBx - Interrupt Request Mask Register
CTBx_OAy_SW	CTBx - Opamp y Switch Control Register
CTBx_OAy_SW_CLEAR	CTBx - Opamp y Switch Control Clear Register
CTBx_CTBBUS_SW	CTBx - CTB Bus Switch Control Register
CTBx_CTBBUS_SW_CLEAR	CTBx - CTB Bus Switch Control Clear Register
CTBx_CTB_SW_HW_CTRL	CTBx - CTB Bus Switch HW-SW Control Register
CTBx_CTB_SW_STATUS	CTBx - CTB Bus Switch Status Register
CTBx_OAy_OFFSET_TRIM	CTBx - Opamp y Offset Trim Control Register
CTBx_OAy_SLOPE_OFFSET_TRIM	CTBx - Opamp y Offset Drift Trim Control Register
CTBx_OAy_COMP_TRIM	CTBx - Opamp y Compensation Trim Register

Note that x is the CTB number and can be 0 or 1; y is the opamps number and can be 0 or 1.

19. Universal Analog Block (UAB)

This chapter presents the universal analog blocks (UAB) and associated applications. The UAB is highly configurable and can realize either single-ended or pseudo-differential transfer functions.

19.1 Features

- Single-ended or pseudo-differential mode of operation
- Five-bit and six-bit programmable capacitor arrays for high-resolution applications, including 12-bit DAC
- UAB chaining for higher resolution and higher order functions
- Auto-zero capable integrator for low-offset and low-1/f noise performance
- Three switched or continuous paths to enable complex filter functions (high-Q notch)
- Reference buffers to prevent kickback noise into the sensitive reference signals
- A block-internal flexible switch routing fabric that enables many transfer functions

19.2 Architecture

Figure 19-1 shows a simplified UAB block diagram. It is built with two completely symmetrical half-circuits configurable as either one pseudo-differential or two single-ended functions. Each half circuit requires a control logic for autonomous operation. The UAB is a switched capacitor integrator with three input capacitor branches and one feedback capacitor branch. The UAB contains opamps, buffers, comparators, various switched capacitor arrays, and basic logic to enable a rich set of functions. In addition to these analog resources, a flexible UAB clocking/timing controller is provided to control all switching activity in real time.

Figure 19-1. UAB Architecture

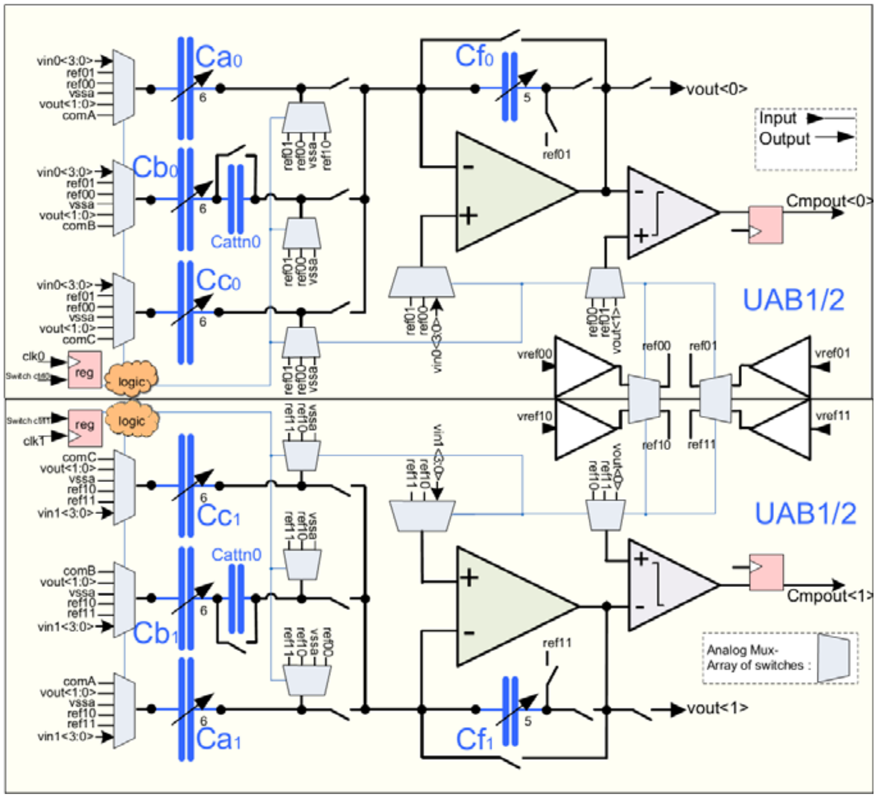
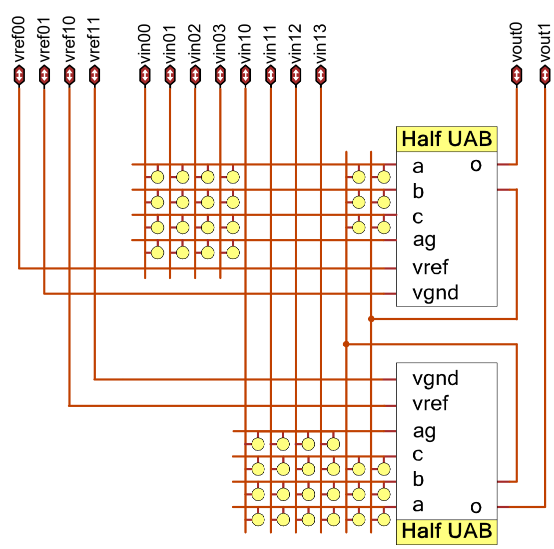


Figure 19-2 shows the simplified block diagram of UAB. Each half UAB has four fully-permutable inputs from AROUTE. The 4 input architecture is selected to allow implementation of (A0–A1)+B+C function. A0–A1 is achieved by time multiplexing two inputs on the A-capacitor. Each half UAB has an independent reference and analog ground input and all four are locally buffered. The four inputs can select from all four reference bus lines in the PSoC Analog Coprocessor; however, those switches are part of the AROUTE and are not shown in Figure 19-2.

Figure 19-2. Simplified UAB Block Diagram



19.3 UAB Controller

The UAB controller generates the control and clock signals required for UAB. It has the following features:

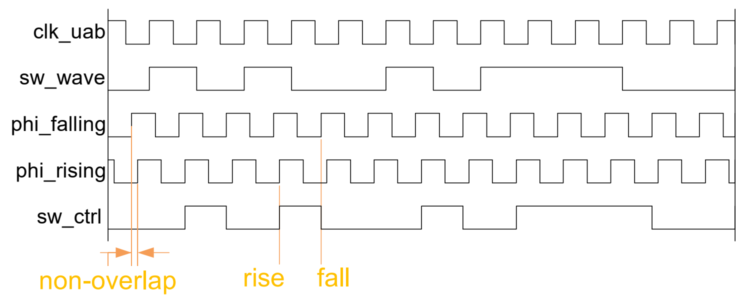
- A 16x16 SRAM containing the definition of up to 12 clocking waveforms with up to 16 steps
  - A step can be a phase or a part of a phase
  - Every digital clock one step from the clocking waveform is executed
  - Maximum analog clock speed is 2 MHz
- A mux for each switch to select the clocking waveform (or fixed value) to be used
  - Some switches can be controlled by an extra control signal (modbit)
  - The control signal can either be a comparator output from any UAB half
  - The control signal can also be a firmware-controlled bit or the sign bit in the DAC use case
- Precise timing for non-overlap and early/regular clocks is fully contained within the analog block
  - Static switch controls and other static controls do not require precise timing
  - Control for the A and B capacitance values require precise timing because they are not static for a waveform generating DAC use case

19.4 UAB Clocking Scheme

The most commonly-used clocking scheme is a clock with two non-overlapping phases. The phases should be non-overlapping to avoid creating an ephemeral short as one switch transitions to open while at the same time another switch transitions to closed (break-before-make). The logic cloud in Figure 19-1 combines the switch control bits and the clocking signals to create a single wire control for each clocked switch.

When the UAB half is running each digital clock cycle, the next bit defining the clocking waveform is read from the SRAM. Figure 19-3 shows a random switch control signal (sw_ctrl) generated from the pattern read from SRAM of the UAB controller.

Figure 19-3. Example Clocking Waveform



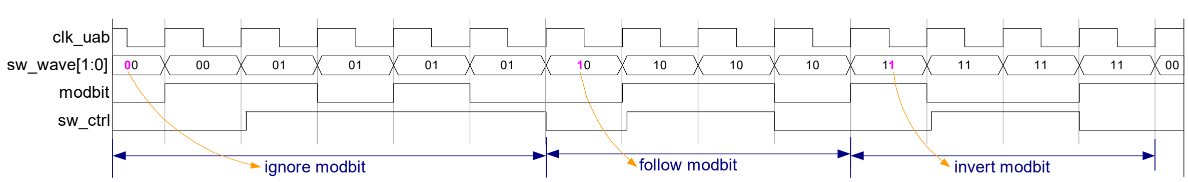
19.4.1 Clocking Waveforms with Extra Control Signal (modbit)

For some use cases, switches need to be closed or opened based on a dynamic control signal in addition to the analog clock phase. For example, the output of the UAB comparator (compout) for a sigma delta ADC use case or a firmware controlled sign bit in the VDAC use case.

To accommodate this, an extra control bit is needed. This extra bit determines if the additional control signal (commonly called 'modbit') is ignored (0) or followed (1). This allows the clocking waveform to be defined with hard open and closed phases (ignore modbit) or to open or close the switch based on the modbit. When the modbit is ignored then the other bit simply defines the waveform. When the modbit is followed then the modbit defines the waveform, and the other bit defines if the modbit should be inverted (1) or not (0).

Figure 19-4 demonstrates some possibilities of the 2-bit code and the modbit. The 2-bit clocking waveforms are also referred to as modbit-able waveforms.

Figure 19-4. Example Clocking Waveform with 2-bit Code



19.5 UAB Synchronization

19.5.1 UAB to SAR Synchronization

It is common for the output of UAB to go to the SAR. The SAR has two modes of operation – Scheduled and Unscheduled – to ensure that it samples a valid UAB output.

19.5.1.1 Scheduled UAB to SAR Synchronization

In this mode, UAB synchronizes to the SAR output trigger. The SAR generates a trigger at the beginning of each scan. Each UAB should start exactly 'startup delay' clk_hf cycles after the SAR output trigger. However, this is only possible if there is a positive clock edge on the UAB clock (clk_uab) precisely at that moment. In most cases the UAB clock generated with the signals from the clock divider will not be properly aligned; therefore, it is necessary to locally generate a 'delayed' UAB clock, which is properly aligned. This delayed UAB clock is generated such that it will have a positive clock edge right after the startup delay.

After the startup delay, the delayed UAB clock needs to continue running at the speed at which the clock divider for the UAB was set. To do that, the local 'Align' logic that generates the delayed UAB clock should know the period of the UAB clock from the clock divider. To achieve that, the Align logic first measures that UAB clock period on the signal from the clock divider.

The startup delay is configured by the firmware in the STARTUP_DELAY field. Note that if the first channel of the SAR scan is a UAB then the UAB may need to be started near the end of the previous iteration of the periodic SAR scan. And because the startup delay is expressed in clk_hf cycles, the STARTUP_DELAY field needs to be able to hold large values, which is why the STARTUP_DELAY field is 16 bits wide.

The Align logic is disabled by making STARTUP_DELAY=0 (the default). When startup delay is disabled the UAB runs on the UAB clock from the clock divider; that is, it bypasses the Align logic. When the startup delay is used (set to a non-zero value) and the UAB RUN bit is set, the clock period can be measured. Note that after the UAB RUN bit is set all of the UAB configurations, including the STARTUP_DELAY and UAB clock setting should be stable.

After the UAB clock period is measured, the Align logic waits for a trigger to start the STARTUP_DELAY count down. After the startup delay the first delayed UAB clock is generated and the measured UAB period value is loaded in the down counter. From this point, a new UAB clock is generated and new period value is loaded in down counter whenever the down counter reaches zero. Note that before the RUN bit is set and during the time that the UAB clock period is being measured, triggers are ignored.

When the UAB clock is switched over from the PCLK provided clock to the internally generated clock (or vice versa), there will be one clock period that is either (much) longer or shorter than configured.

The Align logic will go back to the idle state when the UAB is stopped (RUN bit cleared) or when Align logic is disabled (STARTUP_DELAY=0).

The formula to calculate the startup delay for a UAB is as follows:

dly = {(offset + st + 1) * clk_sar - 2 - (last_valid_step+1) * clk_uab} mod period

dly = UAB startup delay (must be $\geq$ 1)

offset = channel offset in the SAR schedule in clk_sar cycles (= sum of preceding channel conversion times)

st = SAR sample time for this UAB channel in clk_sar cycles

clk_sar = SAR clock expressed in clk_hf cycles (must be $\geq$2)

clk_uab = UAB clock expressed in clk_hf cycles (must be $\geq$ 2)

last_valid_step = last UAB SRAM address that the valid clocking waveform is high

mod = modulo operator

period = minimum of the SAR Trigger and UAB period time in clk_hf cycles

Note that in this mode it is imperative that the SAR scan is periodic and the SAR scan period is a integer multiple of the UAB period. This is because when it is synchronized, the synchronization by definition remains locked.

19.5.1.2 Unscheduled UAB to SAR Synchronization

In 'Unscheduled' mode, the UABs are not synchronized to the SAR and the SAR scan may even be ad hoc–not periodic. Instead, the SAR will synchronize to the positive edge of the output trigger of the UAB selected by the SAR channel to sample a valid UAB output. This means that when the SAR selects a UAB it will wait an undetermined amount of time for the UAB trigger to arrive. Moreover, if due to subsampling the UAB Valid output was suppressed, the SAR will take another sample of the same UAB after waiting for another trigger, until a Valid sample is received.

When a UAB runs at its maximum frequency of 2 MHz, the output phase is 250 ns long (assuming two phases). This is long enough to meet the SAR's minimum input sample time requirement of 194 ns.

However, the UAB output will take some time to settle in the beginning of the valid phase. Therefore, it is desirable to align the end of the UAB valid phase with the end of the SAR sample window.

With the UAB running at 2 MHz, the 500 ns time is convenient because it is equal to one full analog clock cycle. Therefore, it is possible that the UAB already has clocking waveform defined, which can also be used for the UAB output trigger. In other use-cases with, for example, a UAB running at a lower frequency, it is possible that a special clocking waveform will need to be created to generate an output trigger at 500 ns before the end of the SAR sample window. This may require the UAB clock to run at 2 MHz or more to create sufficient step granularity. Note that another solution for fine grained control over the timing from the trigger to sample window end is adjusting the "SAMPLE_TIME" in the SAR.

19.6 Application Description

Some basic functions supported by UAB are listed in Table 19-1.

Table 19-1. Functions Supported by UAB

Function	Comment
Delta-Sigma Modulator	First order in one UAB. Chainable to build second order (SE and Diff)
Incremental ADC	14+ bits at low rate 100 sps (SE and Diff)
MDAC	10-bit INL but-12 bit numerical result (SE and Diff)
Filter	Gregorain and Themes Low/High Q biquad with one full UAB (SE and Diff)
Summer	Add up to three analog signals (SE and Diff)
Subtractor	Subtract up to three analog signals (SE and Diff)
Integrator	Resettable correlated double sample (CDS) capable (SE and Diff)
Programmable Gain Amplifier	Inverting and non-inverting supported (x1, 2, 4, 8, 16) (SE and Diff)
Mixer	External signal required to modulate the input signal (SE and Diff)
Comparator	Sampled data comparator (SE and Diff)
Sample and Hold	Programmable aperture and hold (SE and Diff)

19.7 Power Modes

The block remains functional during Active and Sleep modes. In Deep-Sleep mode, all clocks are off but all the power remains on. Therefore, all digital controls to the UAB remain valid; however, they will be static and the block is not functional.

19.8 Registers Summary

Register	Description
UAB_CTRL	Global UAB control
INTR	Interrupt request register
INTR_SET	Interrupt request set register
INTR_MASK	Interrupt request mask
INTR_MASKED	Interrupt request masked
OA0_CTRL	Opamp, comparator, and buffer controls
CAP_CTRL0	Capacitance controls
CAP_ABCF0_VAL	Capacitance values for CA0, CB0, CC0, and CF0
CAP_AB0_VAL_NXT	Next capacitance values for CA0 and CB0
CAP_CF0_VAL_NXT	Next capacitance values for CC0 and CF0
STARTUP_DELAY0	Startup delay
SUBSAMPLE_CTRL0	Subsample control
SW_STATIC0	Static switches for UAB half 0
SW_MODBIT_SRC0	Select source of Modbit for A, B, and C branches of half 0

Register	Description
SW_CA0_IN0	Cap A0 input switches set 0
SW_CA0_IN1	Cap A0 input switches set 1
SW_CA0_TOP	Cap A0 top plate switches
SW_CB0_IN0	Cap B0 input switches set 0
SW_CB0_IN1	Cap B0 input switches set 1
SW_CB0_TOP	Cap A0 top plate switches
SW_CC0_IN0	Cap C0 input switches set 0
SW_CC0_IN1	Cap C0 input switches set 1
SW_CC0_TOP	Cap A0 top plate switches
SW_CF0_BOT	Cap F0 bottom plate and output switches
SW_OTHER0	Other clocked controls
SW_BOOST_CTRL0	Bootstrap clock control
SRAM0_CTRL	SRAM programmed size
STAT0	Status current SRAM counter and comparator
SRAM0	Waveform SRAM for UAB half 0
OA1_CTRL	Opamp, comparator, and buffer controls
CAP_CTRL1	Capacitance controls
CAP_ABCF1_VAL	Capacitance values for CA1, CB1, CC1, and CF1
CAP_AB1_VAL_NXT	Next capacitance values for CA1 and CB1
CAP_CF1_VAL_NXT	Next capacitance values for CC1 and CF1
STARTUP_DELAY1	Startup delay
SUBSAMPLE_CTRL1	Subsample control
SW_STATIC1	Static switches for UAB half 1
SW_MODBIT_SRC1	Select source of Modbit for A, B, and C branches of half1
SW_CA1_IN0	Cap A1 input switches set 0
SW_CA1_IN1	Cap A1 input switches set 1
SW_CA1_TOP	Cap A1 top plate switches
SW_CB1_IN0	Cap B1 input switches set 0
SW_CB1_IN1	Cap B1 input switches set 1
SW_CB1_TOP	Cap A1 top plate switches
SW_CC1_IN0	Cap C1 input switches set 0
SW_CC1_IN1	Cap C1 input switches set 1
SW_CC1_TOP	Cap A1 top plate switches
SW_CF1_BOT	Cap F1 bottom plate and output switches
SW_OTHER1	Other clocked controls
SW_BOOST_CTRL1	Bootstrap clock control
SRAM1_CTRL	SRAM programmed size
STAT1	Status current SRAM counter and comparator
SRAM1	Waveform SRAM for UAB half 1
CAP_TRIM0	Trim for attenuation cap half0
CAP_TRIM1	Trim for attenuation cap half1
OA_TRIM0	Trim for opamp and buffers half0
OA_TRIM1	Trim for opamp and buffers half1

20. SAR ADC

The PSoC® Analog Coprocessor has one successive approximation register analog-to-digital converter (SAR ADC). It has the following features:

- Selectable resolution of 8-, 10-, and 12-bit
- Aggregate sample rate up to 1 Msps
- Sixteen input channels multiplexed by an hardware sequencer
- Single-ended and Differential input modes
- Multiple options for input sources – external (GPIOs), continuous timer block (CTB) output, universal analog block (UAB), output, on-chip temperature sensor, and analog mux buses
- Optional second order switched-capacitance filter (implemented in UAB) on channel 0
- Programmable signal acquisition time
- Selectable averaging mode – Sequential-fixed resolution, Sequential-accumulate, Interleaved accumulation mode
- Selectable reference – VDDA, VDDA/2, bandgap voltage of 1.2 V, voltage reference from programmable reference block, and an external pin
- Interrupt generation

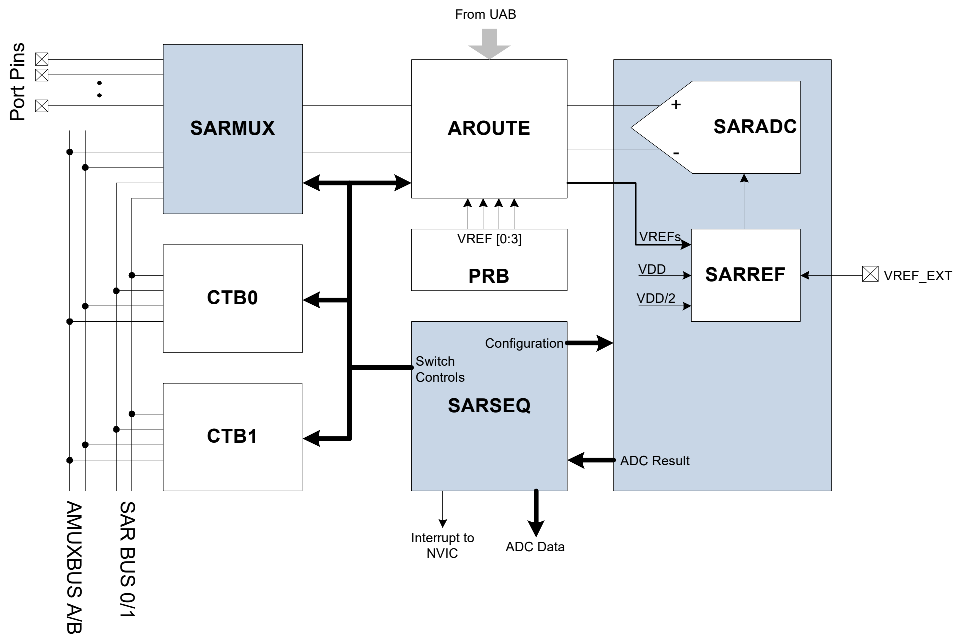
20.1 Architecture Overview

SAR ADC consists of the following blocks:

- SARADC – This is the SAR core consisting of an engine to translate the input analog to a digital count.
- SARREF – This block consists of a multiplexer to select a reference signal for the SARADC. It receives the reference signals from the bandgap, programmable reference block (PRB), analog route (AROUTE), and an external pin.
- SARMUX – This is the 16-channel input multiplexer for the SARADC. The channel selection is controlled in the hardware by the SAR sequencer (SARSEQ). This enables multi-channel signal conversion in a round-robin fashion without CPU intervention.
- SARSEQ – This block is the control unit for SAR ADC. It is responsible for selecting the input channel, configuration of SARADC and SARREF block depending on the selected channel, controlling the sequence of events throughout the conversion process, output averaging, and generating the interrupt.

Figure 20-1 shows the interconnections of these blocks. Extensive analog routing with the use of AROUTE, AMUXBUS A/B, and SARBUS 0/1 enables the SAR ADC to connect to other peripherals in the device such as the CTB and UAB. This is useful in many analog signal chain applications where the amplifier is built using CTB or the filter with UAB and their its output are required to be measured by the SAR ADC.

Figure 20-1. SAR ADC Block Diagram



Note: This is an overview diagram; the details of interconnection are in the Analog Routing [AROUTE] chapter on page 208.

20.2 SARADC

This is the SAR core providing 8-bit, 10-bit, or 12-bit resolution. It has a maximum sample rate of 1 Msps. It works in differential mode, but can support single-ended mode by connecting one end of the input to a fixed reference.

SARADC is enabled/disabled by writing to the SAR_CTRL[31] register bit.

Table 20-1. Enabling/Disabling the SAR ADC

SAR_CTRL[31]	Description
0	Disables the SAR ADC
1	Enables the SAR ADC

The SARADC block is responsible for analog to digital conversion, but the channel is selected by the sequencer block. The sequencer connects the input of the SARADC to the source under measurement, applies the configuration to the SARADC block, and does post processing. Configuration of the SAR ADC core is unique to the channel and comes from the sequencer. The configuration parameters include measurement resolution (8-bit/10-bit/12-bit), differential/single input mode, and the signal acquisition time. These parameters are set in the configuration registers. As the parameters are unique to the channel, there are 16 configuration registers (SAR_CHAN_CONFIG) – one for each channel.

20.2.1 Setting the Resolution

The ADC resolution is set using the SAR_CHAN_CONFIGx[9] register (where x is the channel number) and the SAR_SAMPLE_CTRL[0] bit as given in the following table.

Table 20-2. Setting SAR ADC Resolution

RESOLUTION (SAR_CHAN_CONFIGx[9])	SUB_RESOLUTION (SAR_SAMPLE_CTRL[0]	SAR ADC Resolution
0	0	12-bit
0	1	12-bit
1	0	8-bit
1	1	10-bit

The SAR_CHAN_CONFIG register is available for every channel, but the SAR_SAMPLE_CTRL is a global register. Thus, if the RESOLUTION bit is set for a set of channels, then all those channels can have either 8-bit or 10-bit resolution.

20.2.2 Setting the Input Mode

SARADC can be configured to single or differential input mode using the SAR_CHAN_CONFIGx[8] bit.

Table 20-3. Setting Input Mode of SAR ADC

DIFFERENTIAL_EN (SAR_CHAN_CONFIGx[8])	Input Mode
0	Single Ended
1	Differential

When the channel is configured to single-ended mode, the negative input connection has several options. This is configured by the SAR_CTRL register.

Table 20-4. Selecting Negative Input of SAR ADC in Single-Ended Mode

NEG_SEL (SAR_CTRL[11:9])	Negative Input	Description
000	VSSA_KELVIN	Connected to the device analog ground VSSA
001	ART_VSSA	Connected to the analog ground VSSA in AROUTE (close to SARADC)
010	P1	Connected to pin 1 of SARMUX port
011	P3	Connected to pin 3 of SARMUX port
100	P5	Connected to pin 5 of SARMUX port
101	P7	Connected to pin 7 of SARMUX port
110	ACORE	Connected to ACORE line in AROUTE
111	VREF	Connected to VREF input of SARADC

Note that the SAR_CTRL register is a global register and is applicable to all channels that are configured in the single-ended mode.

The input range of the SAR ADC depends on the negative input voltage and reference voltage. If the negative input voltage is Vn and the reference voltage is VREF, then the input range is Vn ± VREF. This holds true for both single-ended and differential mode. Note that Vn ± VREF should be in the range of VSSA to VDDA. If the negative input is connected to VSSA, the input range is limited to 0 to VREF and not –VREF to VREF. This is because the input signal cannot be less than VSSA. This effectively generates only a 11-bit result.

The reference voltage is set by configuring SARMUX as mentioned in SARREF on page 193.

20.2.3 SAR ADC Clock

A clock is required to run the SAR ADC. Its frequency must be between 1 MHz and 18 MHz, which can come from the IMO or the external source via a clock divider. Note that SAR ADC does not support a fractional divider. This clock is one of the parameters which decide the sample rate.

20.2.3.1 Setting Signal Acquisition Time

Acquisition time is the time taken by the sample and hold (S/H) circuit to settle. After the acquisition time, the input signal source is disconnected from the SARADC core and the output of the S/H circuit is used for conversion. Four programmable signal acquisition time configurations are provided for use with all channels. Each configuration provides 10-bit control, that is, 0 to 1023 SAR ADC clocks delay. For example, at 1 MHz SAR ADC clock, the acquisition time can be 0 µs to 1023 µs. These four configurations are packed into two registers SAR_SAMPLE_TIME01 and SAR_SAMPLE_TIME23, as shown in the following tables.

Table 20-5. Setting Signal Acquisition Time – Configuration 0 and 1

SAR_SAMPLE_TIME01	Description
[9:0]	Sample Time Configuration 0
[25:16]	Sample Time Configuration 1

Table 20-6. Setting Signal Acquisition Time – Configuration 2 and 3

SAR_SAMPLE_TIME23	Description
[9:0]	Sample Time Configuration 2
[25:16]	Sample Time Configuration 3

To select a particular configuration, use the SAMPLE_TIME_SEL bits provided in the SAR_CHAN_CONFIGx register as shown in Table 20-7.

Table 20-7. Selecting the Signal Acquisition Time Configuration

SAMPLE_TIME_SEL (SAR_CHAN_CONFIGx[13:12])	Description
00	Sample Time Configuration 0 is selected
01	Sample Time Configuration 1 is selected
02	Sample Time Configuration 2 is selected
03	Sample Time Configuration 3 is selected

20.2.4 Conversion Time

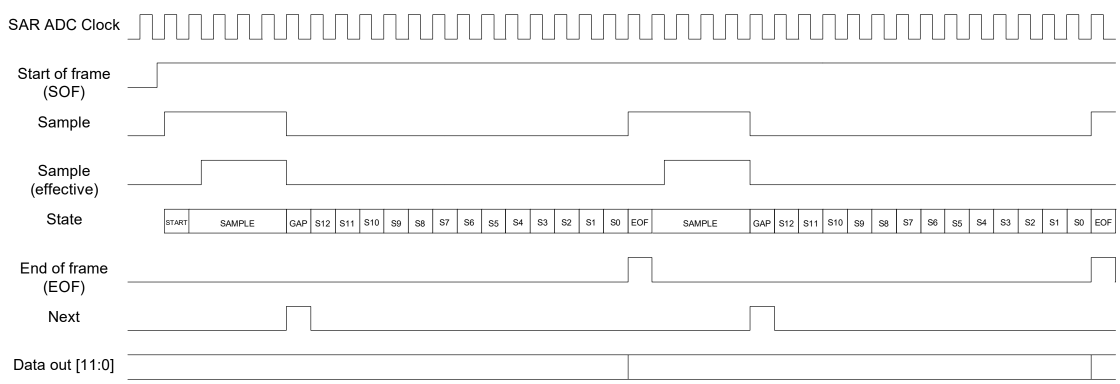
The conversion time of the SAR ADC depends on the SAR ADC clock, resolution, and signal acquisition time as given in the following equation:

Conversion time (# SAR ADC Clocks) = Resolution (bits) + 2 + Acquisition time (count)

Thus, for a resolution of 12 bits with acquisition time (count) of four, 18 SAR ADC clocks are required. To achieve sample rate of 1 Msps, an 18-MHz SAR ADC clock is required. The details of the conversion stages are explained in 20.2.5 SAR ADC Timing.

20.2.5 SAR ADC Timing

Figure 20-2. SAR ADC Timing Diagram



Start of frame (SOF) is generated on the trigger by the SARSEQ block (see Trigger on page 199 for more details). It is sampled at the rising edge of the SAR ADC clock. In the next one and a half SAR ADC clocks, auto-zeroing action takes place. Then the input is sampled for (n–1/2) clock cycles where n is the SAMPLE_TIME set for the signal acquisition time. If the SAMPLE_TIME is configured to 4, then the input is sampled for three and a half clock cycles. There is a one clock delay (GAP) before the conversion is started. It then takes one clock for every bit of the result; for example, it takes 12 cycles for 12-bit resolution. After conversion is complete, one clock is taken for end of frame (EOF). Thus, for a 12-bit result, SAR ADC requires 18 clock cycles. Note that this is applicable in continuous conversion mode. In one shot mode, an extra clock is required.

SARADC generates a "next" signal, which indicates the completion of channel sampling. This signal can be used by other peripherals in the device.

20.2.6 Result Data Format

Result data format is configurable from two aspects:

- Signed/Unsigned
- Left-/Right-Aligned

When the result is signed, the most significant bit of the conversion is used for sign extension to 16 bits. For an unsigned conversion, the result is zero extended to 16 bits. Selection of the signed or the unsigned result can be independently made to a differential input mode and single-ended input mode configured channels using the SAR_SAMPLE_CTRL register. Note that this is a global register and the settings made in this register applies to all channels.

Table 20-8. Selecting Single-Ended/Differential and Unsigned/Signed Result Format

DIFFERENTIAL_EN (SAR_CHAN_CONFIGx[8])	SINGLE_ENDED_SIGNED (SAR_SAMPLE_CTRL[2])	DIFFERENTIAL_SIGNED (SAR_SAMPLE_CTRL[3])	Result Data Format
0	0	X	Single-ended, unsigned result
0	1	X	Signed-ended, signed result
1	X	0	Differential, unsigned result
1	X	1	Differential, signed result

The result can either be right-aligned or left-aligned within the 16 bits of the result register. By default, data is right-aligned in data[11:0] with sign extension to 16 bits. A lower resolution combined with left alignment will cause lower significant bits to be made zero. Alignment is selected using the SAR_SAMPLE_CTRL register.

Table 20-9. Selecting Right-Aligned/Left-Aligned Result Format

LEFT_ALIGN (SAR_SAMPLE_CTRL[1])	Result Data Alignment
0	Right aligned (default)
1	Left aligned

Combined with signed and unsigned formats, and left and right alignment for 12-, 10-, and 8-bit conversion, the result data format is as follows.

Table 20-10. Result Data Format Summary

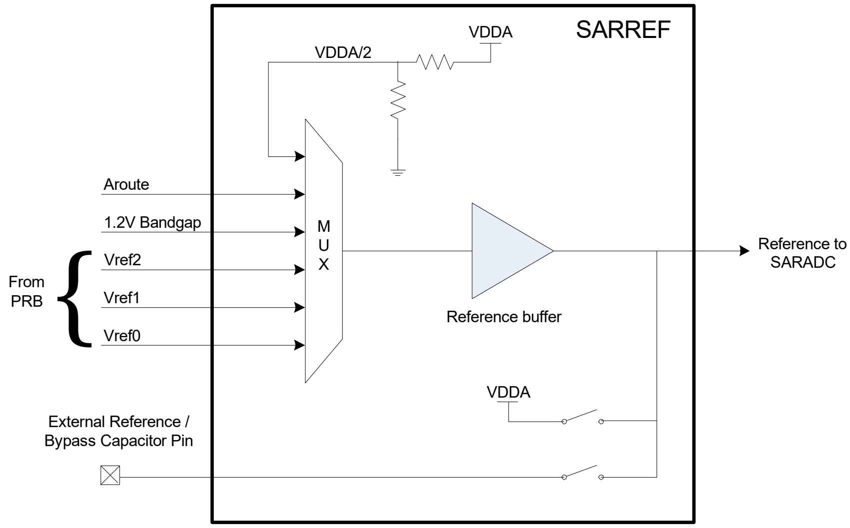
<table>
<tr><th rowspan="2">Signed/Unsigned</th><th rowspan="2">Alignment</th><th rowspan="2">Resolution</th><th colspan="16">Result Register</th></tr>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="3">Unsigned</td><td rowspan="3">Right</td><td>12</td><td>–</td><td>–</td><td>–</td><td>–</td><td colspan="12">Result Magnitude</td></tr>
<tr><td>10</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td colspan="10">Result Magnitude</td></tr>
<tr><td>8</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td colspan="8">Result Magnitude</td></tr>
<tr><td rowspan="3">Signed</td><td rowspan="3">Right</td><td>12</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td colspan="11">Result Magnitude</td></tr>
<tr><td>10</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td colspan="9">Result Magnitude</td></tr>
<tr><td>8</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td><td colspan="8">Result Magnitude</td></tr>
<tr><td rowspan="3">Unsigned</td><td rowspan="3">Left</td><td>12</td><td colspan="12">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>10</td><td colspan="10">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>8</td><td colspan="7">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td rowspan="3">Signed</td><td rowspan="3">Left</td><td>12</td><td>S</td><td colspan="11">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>10</td><td>S</td><td colspan="9">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td>8</td><td>S</td><td colspan="7">Result Magnitude</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
</table>

'S': sign bit (0 - positive, 1 - negative)

20.3 SARREF

This block is used to provide reference voltage to the SARADC and also enable reference bypass for external filtering. Figure 20-3 shows the block diagram.

Figure 20-3. SARREF Block Diagram



There are multiple options for reference voltage, selected using the SAR_CTRL register as given in the following table.

Table 20-11. Selecting Reference for SAR ADC

VREF_SEL (SAR_CTRL[6:4])	Reference	Reference Buffer
000	VREF0 from PRB	ON
001	VREF1 from PRB	ON
010	VREF2 from PRB	ON
011	VREF from AROUTE	ON
100	1.2V from Bandgap	ON
101	External reference from pin	OFF
110	VDDA/2	ON
111	VDDA	OFF

20.3.1 Bypass Capacitors

The selected reference from the mux can be routed to the external pin (bypass) where an external capacitor can be used to filter the noise that may exist on the reference signal. It is enabled using the SAR_CTRL register.

Table 20-12. Enabling Bypass Capacitor Connection

VREF_BYP_CAP_EN (SAR_CTRL[7])	Bypass
0	Disabled
1	Enabled

The SAR ADC clock has some limitations if the bypass is not enabled. This impacts the maximum sample rate that can be obtained. Table 20-13 summarizes the maximum SAR ADC clock frequency for different use cases.

Table 20-13. Maximum SAR ADC Clock Frequency and Sample Rate in Different Use Cases

Reference	Buffer	Bypass	Maximum Frequency	Maximum Sample Rate (12 bit, 4 ADC Clocks Acquisition Time))
Reference from PRB/AROUTE/Bandgap	ON	No	1.6 MHz	100 ksps
Reference from PRB/AROUTE/Bandgap	ON	Yes	18 MHz	1 Msps
External Reference[a]	OFF	X	18 MHz	1 Msps
VDDA/2	ON	No	1.6 MHz	100 ksps
VDDA/2	ON	Yes	18 MHz	1 Msps
VDDA[a]	OFF	X	9 MHz	500 ksps

a. It is recommended to enable the bypass capacitors.

The internal reference voltage startup time varies with different bypass capacitors. The following table lists two common values of the bypass capacitor and the startup time. If the reference voltage selection is changed between the scans, make sure the reference is settled when the SAR ADC starts sampling.

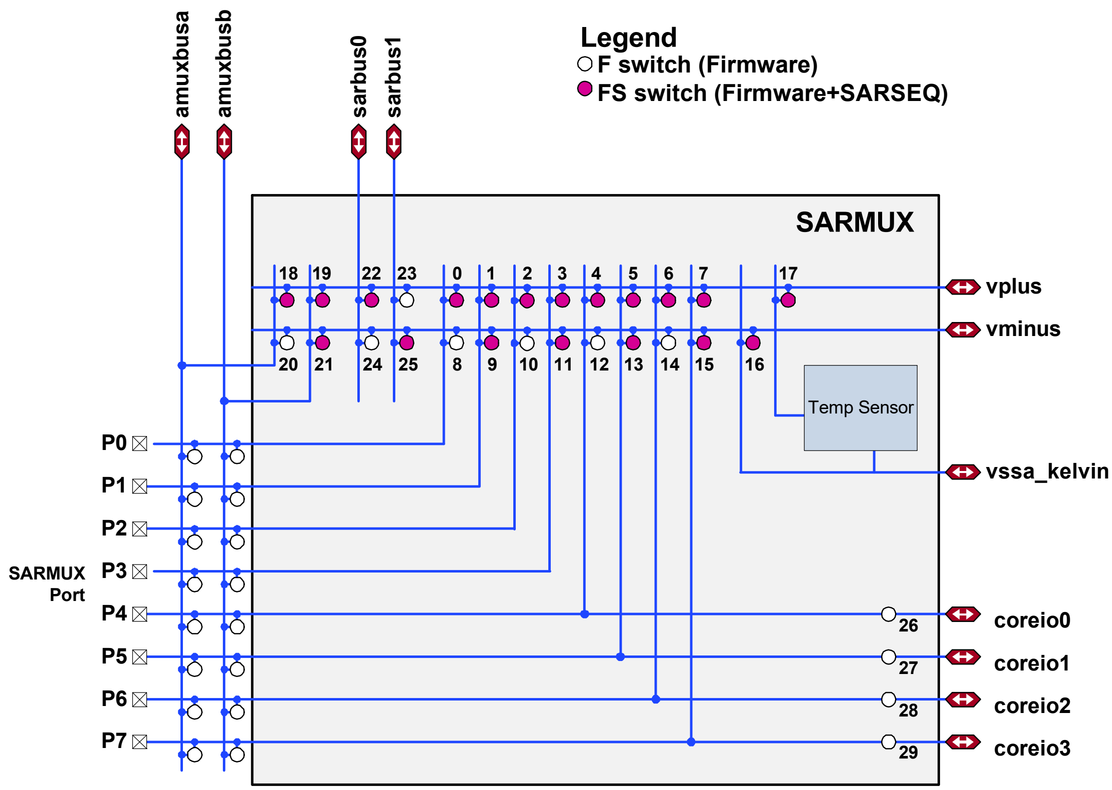
Table 20-14. Bypass Capacitor and Reference Start Up Time

External Bypass Capacitor	Maximum Specification (Internal VREF startup time)
1 uF	2 ms
100 nF	200 us

20.4 SARMUX

SARMUX is an analog dedicated programmable multiplexer primarily used for flexible SAR ADC input routing. Figure 20-4 shows the SARMUX architecture. The vplus and vminus lines shown in the figure are routed to the SAR ADC inputs through the AROUTE fabric.

Figure 20-4. SARMUX Architecture



SARMUX has two types of switches as shown in Figure 20-4. Some switches (F switches) are controlled only in firmware and some (FS switches) are controlled in both firmware and the sequencer (SARSEQ). When SARSEQ is controlling the switches, it configures the switches based on the channel to be measured without CPU intervention. When switches are operated in firmware, the user must configure the switches depending on the channel to be selected.

A dedicated SARMUX port, consisting of eight pins, can be used as an input channel to the SAR ADC. SARMUX also connects to AMUXBUS A/B, SARBUS 0/1, and coreio[3:0] lines. SARBUS and coreio[3:0] connect to other analog peripherals such as CTB, UAB, VDAC, and PRB through AROUTE. Note that AROUTE and CTB also have the F and FS switches. The input to SARADC is selected by configuring the SARMUX, AROUTE, and CTB switches. The details of AROUTE and CTB can be obtained in the Analog Routing [AROUTE] chapter on page 208 and Continuous Time Block (CTB) chapter on page 171.

The following register configurations are used to control the switches:

- POS_PIN_ADDR, POS_PORT_ADDR, NEG_PIN_ADDR, and NEG_PORT_ADDR in the SAR_CHAN_CONFIGx register – selects the source for the channel (positive and negative input), one register for each channel
- SAR_MUX_SWITCH0 register – enables firmware control of the switch, one bit for each switch
- SAR_MUX_HW_CTRL register – enables hardware control of the switch, one bit for each switch

A switch is closed or opened depending on the following conditions on the setting in these registers:

- F switch is closed if the corresponding bit in SAR_MUX_SWITCH0 is set to '1'.
- FS switch (in firmware mode) is closed if the corresponding bit in SAR_MUX_SWITCH0 is set to '1' and SAR_MUX_HW_CTRL is set to '0'.
- FS switch (in sequencer mode) requires the corresponding bit in SAR_MUX_SWITCH0 and SAR_MUX_HW_CTRL set to '1', and the channel selected by SAR_CHAN_CONFIGx. Note that the user configures these registers, but the switch is controlled only by the control signals sent by the sequencer.

SAR_MUX_SWITCH0 register is used to control the switches present in the SARMUX through firmware.

The following table shows the details of this register.

Table 20-15. SAR_MUX_SWITCH0 - Firmware Control of SARMUX Switches

SAR_MUX_SWITCH0 [29:0] Bit	Name	Description	Switch Number in Figure 20-4
0	MUX_FW_P0_VPLUS	Connects SARMUX Port pin P0 and VPLUS	0
1	MUX_FW_P1_VPLUS	Connects SARMUX Port pin P1 and VPLUS	1
2	MUX_FW_P2_VPLUS	Connects SARMUX Port pin P2 and VPLUS	2
3	MUX_FW_P3_VPLUS	Connects SARMUX Port pin P3 and VPLUS	3
4	MUX_FW_P4_VPLUS	Connects SARMUX Port pin P4 and VPLUS	4
5	MUX_FW_P5_VPLUS	Connects SARMUX Port pin P5 and VPLUS	5
6	MUX_FW_P6_VPLUS	Connects SARMUX Port pin P6 and VPLUS	6
7	MUX_FW_P7_VPLUS	Connects SARMUX Port pin P7 and VPLUS	7
8	MUX_FW_P0_VMINUS	Connects SARMUX Port pin P0 and VMINUS	8
9	MUX_FW_P1_VMINUS	Connects SARMUX Port pin P1 and VMINUS	9
10	MUX_FW_P2_VMINUS	Connects SARMUX Port pin P2 and VMINUS	10
11	MUX_FW_P3_VMINUS	Connects SARMUX Port pin P3 and VMINUS	11
12	MUX_FW_P4_VMINUS	Connects SARMUX Port pin P4 and VMINUS	12
13	MUX_FW_P5_VMINUS	Connects SARMUX Port pin P5 and VMINUS	13
14	MUX_FW_P6_VMINUS	Connects SARMUX Port pin P6 and VMINUS	14
15	MUX_FW_P7_VMINUS	Connects SARMUX Port pin P7 and VMINUS	15
16	MUX_FW_VSSA_VMINUS	Connects VMINUS to VSSA	16
17	MUX_FW_TEMP_VPLUS	Connects temperature sensor output to VPLUS	17
18	MUX_FW_AMUXBUSA_VPLUS	Connects AMUXBUSA to VPLUS	18
19	MUX_FW_AMUXBUSB_VPLUS	Connects AMUXBUSB to VPLUS	19
20	MUX_FW_AMUXBUSA_VMINUS	Connects AMUXBUSA to VMINUS	20
21	MUX_FW_AMUXBUSB_VMINUS	Connects AMUXBUSB to VMINUS	21
22	MUX_FW_SARBUS0_VPLUS	Connects SARBUS0 to VPLUS	22
23	MUX_FW_SARBUS1_VPLUS	Connects SARBUS1 to VPLUS	23
24	MUX_FW_SARBUS0_VMINUS	Connects SARBUS0 to VMINUS	24
25	MUX_FW_SARBUS1_VMINUS	Connects SARBUS1 to VMINUS	25
26	MUX_FW_P4_COREIO0	Connects SARMUX Port pin P4 to COREIO0	26
27	MUX_FW_P5_COREIO1	Connects SARMUX Port pin P5 to COREIO1	27
28	MUX_FW_P6_COREIO2	Connects SARMUX Port pin P6 to COREIO2	28
29	MUX_FW_P7_COREIO3	Connects SARMUX Port pin P7 to COREIO3	29

The FS switches can be configured to be firmware-controlled or hardware sequencer-controlled by writing into the SAR_MUX_SWITCH_HW_CTRL register.

Table 20-16. SAR_MUX_SWITCH_HW_CTRL - Selection of Hardware or Firmware Control

SAR_MUX_SWITCH_HW_CTRL [29:0] Bit	Name	Switch Number in Figure 20-4
0	MUX_HW_CTRL_P0	0
1	MUX_HW_CTRL_P1	1
2	MUX_HW_CTRL_P2	2
3	MUX_HW_CTRL_P3	3
4	MUX_HW_CTRL_P4	4
5	MUX_HW_CTRL_P5	5

Table 20-16. SAR_MUX_SWITCH_HW_CTRL - Selection of Hardware or Firmware Control

SAR_MUX_SWITCH_HW_CTRL [29:0] Bit	Name	Switch Number in Figure 20-4
6	MUX_HW_CTRL_P6	6
7	MUX_HW_CTRL_P7	7
16	MUX_HW_CTRL_VSSA	8
17	MUX_HW_CTRL_TEMP	9
18	MUX_HW_CTRL_AMUXBUSA	10
19	MUX_HW_CTRL_AMUXBUSB	11
22	MUX_HW_CTRL_SARBUS0	12
23	MUX_HW_CTRL_SARBUS1	13

Note that this register configures FS switches present in SARMUX. However, there are FS switches also available in the CTB and AROUTE. See the Continuous Time Block (CTB) chapter on page 171 and Analog Routing [AROUTE] chapter on page 208 for details about the registers to be used.

POS_PIN_ADDR and POS_PORT_ADDR in the SAR_CHAN_CONFIGx register select the source for the positive input of channel 'x'.

Table 20-17. Positive Input Selection for SAR ADC

POS_PORT_ADDR SAR_CHAN_CONFIGx[18:16]	POS_PIN_ADDR SAR_CHAN_CONFIGx[2:0]	Location	Source to Positive Input of SAR ADC
000	000 to 111	SARMUX	SARMUX port pin 0 to pin 7
001	000 to 111	CTB0	CTB0 port pin 0 to pin 7
010	000 to 111	CTB1	CTB1 port pin 0 to pin 7
110	000	AROUTE	Output of opamp OA0 of CTB0
	001	AROUTE	Output of opamp OA1 of CTB0
	010	AROUTE	Output 0 of UAB
	011	AROUTE	Output 1 of UAB
	100	AROUTE	Output of opamp OA0 of CTB1
	101	AROUTE	Output of opamp OA1 of CTB1
111	000	SARMUX	Temperature sensor
	010	SARMUX	AMUXBUS A
	011	SARMUX	AMUXBUS B

The following are limitations of the positive input while using the sequencer with differential mode:

- Odd numbered pins (1, 3, 5, and 7) of the SARMUX port and the CTB port cannot be used.
- For POS_PORT_ADDR of 110, POS_PIN_ADDR of 001, 011, and 101 cannot be used.

Similar to the positive input, NEG_PIN_ADDR and NEG_PORT_ADDR in the SAR_CHAN_CONFIGx register select the source for negative input of the channel 'x'. Note that these registers are applicable only in differential mode.

Table 20-18. Negative Input Selection for SAR ADC

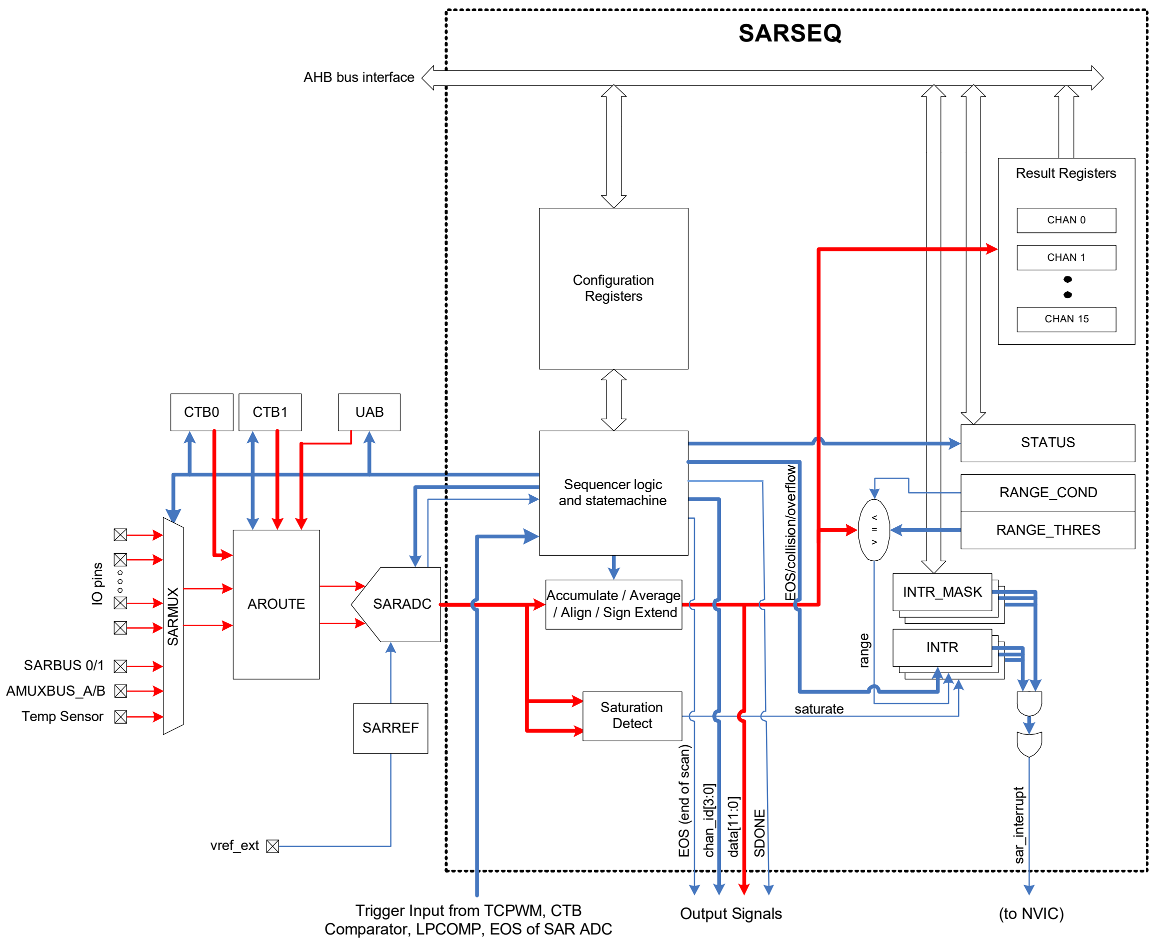
NEG_PORT_ADDR SAR_CHAN_CONFIGx[18:16]	NEG_PIN_ADDR SAR_CHAN_CONFIGx[2:0]	Location	Source to Negative Input of SAR ADC
000	000	SARMUX	SARMUX port pin 1
	010	SARMUX	SARMUX port pin 3
	100	SARMUX	SARMUX port pin 5
	110	SARMUX	SARMUX port pin 7
110	001	AROUTE	Output of OpAmp OA1 of CTB0
	011	AROUTE	Output 1 of UAB
	101	AROUTE	Output of OpAmp OA1 of CTB1

20.5 SARSEQ

SARSEQ is a dedicated sequencer that automatically sequences the SAR ADC input from one channel to the next while placing the result in an array of registers. It has the following main tasks:

- Control the FS switches to set the input channel. Switches are present in SARMUX, AROUTE, and CTB
- Control the sequence of events for every channel, which involves:
  - Configuring the resolution (8, 10, and 12 bit)
  - Setting the negative input (decides the input mode – single-ended or differential)
  - Setting the acquisition time
  - Starting the scan by triggering the SARADC core and placing the result in the appropriate register
- Running the scan for all enabled channels on an external trigger, firmware trigger, or continuously scan without any trigger
- Generating pulse signals on the completion of sampling a channel and the end of conversion of all the enabled channels
- Averaging the result
- Detecting range with two programmable thresholds and saturation detect
- Generating interrupt on various conditions

Figure 20-5. SARSEQ Block Diagram



20.5.1 Enabling a Channel

The channels are enabled/disabled using the SAR_CHAN_EN[15:0] register. Each bit corresponds to a particular channel. For example, writing '1' at the SAR_CHAN_EN[0] bit, enables channel 0; writing '0' disables it.

20.5.2 Trigger

There are two ways to start scanning all enabled channels:

- Firmware trigger (one shot) – In this mode, the firmware starts the scan of the enabled channels. Scanning happens only once and if required, the firmware must start the scan again. Scan is started by writing '1' to the FW_TRIGGER bit in SAR_START_CTRL. When the conversion is complete, SARSEQ clears the FW_TRIGGER bit and waits for the user to set this bit again to start the next scan. This bit is also cleared if the SAR ADC is disabled, given that it was enabled before.

  This mode is always available for use unless the CONTINUOUS bit in the SAR_SAMPLE_CTRL register is set, which causes continuous scanning of all the enabled channels.
- Hardware Trigger (from on-chip signals) – In this mode, TCPWM's overflow, compare match and underflow, SARADC's end of conversion, CTB comparator outputs, and low-power comparator outputs are used to trigger the start of scan. This mode is enabled by writing '1' to the DSI_TRIGGER_EN bit of the SAR_SAMPLE_CTRL register as provided in the following table.

Table 20-19. Enabling Hardware Trigger

DSI_TRIGGER_EN (SAR_SAMPLE_CTRL[17])	Description
0	Firmware trigger only
1	Hardware trigger is enabled; however, firmware trigger also remains enabled

Trigger can be rising edge or level type. In rising edge, one scan happens on every rising edge. In level trigger, the scan continuously happens as long as signal is logic 1. The trigger type is selected using the DSI_TRIGGER_LEVEL bit of the SAR_SAMPLE_CTRL register as shown in the following table.

Table 20-20. Setting Hardware Trigger Type

DSI_TRIGGER_LEVEL (SAR_SAMPLE_CTRL[18])	Trigger Type
0	Rising Edge
1	Level

Note that it is still possible to use firmware trigger by writing FW_TRIGGER bit in the hardware trigger mode. If both firmware trigger and hardware trigger happens at the same time, priority is given to the hardware trigger; after the completion, scan is started again for the firmware trigger.

Hardware trigger is selected using TR_OUT_CTLx register.

Table 20-21. Selecting Hardware Trigger Source

TR_OUT_CTLx[6:0]	Trigger Source
0	Hardwired to 0 (for firmware trigger)
1	TCPWM 0 Overflow
2	TCPWM 1 Overflow
3	TCPWM 2 Overflow
4	TCPWM 3 Overflow
5	TCPWM 4 Overflow
6	TCPWM 5 Overflow
7	TCPWM 6 Overflow
8	TCPWM 7 Overflow
9	TCPWM 0 Compare Match
10	TCPWM 1 Compare Match
11	TCPWM 2 Compare Match
12	TCPWM 3 Compare Match

Table 20-21. Selecting Hardware Trigger Source

TR_OUT_CTLx[6:0]	Trigger Source
13	TCPWM 4 Compare Match
14	TCPWM 5 Compare Match
15	TCPWM 6 Compare Match
16	TCPWM 7 Compare Match
17	TCPWM 0 Underflow
18	TCPWM 1 Underflow
19	TCPWM 2 Underflow
20	TCPWM 3 Underflow
21	TCPWM 4 Underflow
22	TCPWM 5 Underflow
23	TCPWM 6 Underflow
24	TCPWM 7 Underflow
25	SAR ADC EOS (End of Scan)
26	CTB0 Comparator 0
27	CTB0 Comparator 1
28	CTB1 Comparator 0
29	CTB1 Comparator 1
30	LPCOMP 0
31	LPCOMP 1

- Continuous trigger – When the CONTINUOUS bit in the SAR_SAMPLE_CTRL register is set to '1', SARSEQ periodically scans the enabled channels without any trigger. The next scan is started immediately after the present scan completion. All other triggers are ignored in this mode. FW_TRIGGER bit is cleared by the SARSEQ at the end of conversion.

Table 20-22. Enabling Continuous Trigger and One-Shot Mode

CONTINUOUS (SAR_SAMPLE_CTRL[16])	Description
0	SAR ADC waits for FW_TRIGGER (one shot) or hardware trigger
1	Continuously scans enabled channels ignoring the triggers

20.5.3 Double Buffer

Each channel is provided with two storage registers for the result. During the scan, the result of the converted channels is stored in one set of registers called work registers, SAR_CHAN_WORKx[15:0]. When all channels are scanned, the data is moved from the working registers to the result registers, SAR_CHAN_RESULTx[15:0].

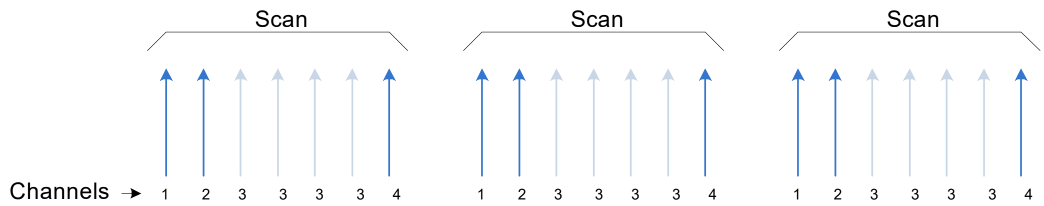
20.5.4 Averaging

SARSEQ provide two averaging modes:

- Sequential
- Interleaved

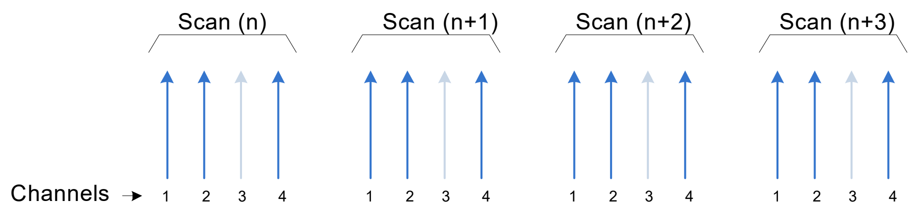
In Sequential mode, SARSEQ accumulates N consecutive samples of the specified channel in every scan. The average result is then calculated and stored in the result register. For example, Figure 20-6 shows SARSEQ taking four consecutive samples of channel 3 in every scan. The result is available at the end of every scan.

Figure 20-6. Sequential Mode



In interleaved mode, one sample is taken for every triggered scan and accumulated, but after the predefined number of samples, the average is calculated and the result register is updated. In all other scans, the result register will not be updated; instead, the intermediate accumulated value is stored in the work register. For example, Figure 20-7 shows SARSEQ taking one sample of channel 3 in every scan. The result is available after predefined number of samples.

Figure 20-7. Interleaved Mode



Averaging can be enabled on a channel basis by writing into the AVG_EN bit of the SAR_CHAN_CONFIGx register.

Table 20-23. Enabling Averaging

AVG_EN (SAR_CHAN_CONFIGx[10])	Averaging
0	Disabled
1	Enabled

Averaging mode is selected using the AVG_MODE bit of the SAMPLE_CTRL register. Note that this is a global setting; this means, if averaging is required, all channels (with averaging enabled) can have either sequential mode or interleaved mode.

Table 20-24. Setting Averaging Mode

AVG_MODE (SAR_SAMPLE_CTRL[8])	Averaging Mode
0	Sequential
1	Interleaved

A 3-bit AVG_CNT in the SAR_SAMPLE_CTRL register specifies the number of samples N to be accumulated based on the following formula:

$N = (1 << (AVGCOUNT[2:0]+1))$

Thus, N ranges from 2 to 256.

Before the accumulated count is loaded into the result register, it can be shifted to get the average. Shifting is enabled using AVG_SHIFT bit in the SAR_SAMPLE_CTRL register.

Table 20-25. Configuring the Result Shift Parameter

AVG_SHIFT (SAR_SAMPLE_CTRL[7])	Shift
0	Accumulated result is shifted right to fit in the 16-bit resolution (limited by the size of the result register). Shifting is not required when the channel resolution is set to 8-bit as the accumulated result will not overflow even at the maximum AVG_CNT of 256. This is the Sequential - Accumulate mode.
1	Accumulated result is shifted right by AVG_CNT+1 bits to get the channel resolution (8/10/12-bit). This is the Sequential - Fixed Resolution mode.

It is important that the AVG_CNT should not exceed 16 for the channel resolution of 12-bit because it will cause the accumulated value in the work register (16-bit) to overflow.

20.5.5 Interrupts

SAR ADC provides interrupts for the following events:

- Range Detection
- Saturate Detect
- End of Scan Interrupt
- Overflow Interrupt
- Collision Interrupt

20.5.5.1 Range Detection

Range detection is provided to allow comparison of sample values with two programmable thresholds and generate the interrupt. CPU is not involved in the comparison; it is done in hardware.

The two programmable thresholds (HIGH and LOW) are stored in the SAR_RANGE_THRES register.

Table 20-26. Selecting the Range Detection Thresholds

SAR_RANGE_THRES	Description
15:0	Low threshold
31:16	High threshold

The interrupt can be triggered by four conditions. One of them is selected using the SAR_RANGE_COND register.

Table 20-27. Selecting Range Detection Condition

SAR_RANGE_COND[31:30]	Condition	Description
00	BELOW	Result < Low threshold
01	INSIDE	Low threshold <= Result < High threshold
10	ABOVE	Result >= High threshold
11	OUTSIDE	Result < Low threshold or Result >= High threshold

The interrupt can be enabled/disabled on a channel basis using the SAR_RANGE_INTR_MASK[15:0] register, with each bit corresponding to a channel. Set the bit to enable the interrupt.

Note that SAR_RANGE_THRES and SAR_RANGE_COND are global registers. Range detect is done for all the enabled channels after averaging, alignment, and sign extension. Thus, the threshold values should be in the same format as the result. Range detection is done after averaging, if enabled.

SAR_RANGE_INTR[15:0] provides the status of the interrupt for each channel. Writing '1' to this register clears the corresponding bit, thus, clearing the interrupt. Note that the bits in this register are ANDed with the corresponding MASK bits and then routed to the NVIC. The ANDed version can be seen in the SAR_RANGE_INTR_MASKED[15:0] register. The interrupt can be triggered in firmware by writing to the the SAR_RANGE_INTR_SET[15:0] register.

20.5.5.2 Saturate Detect

SARSEQ detects if the sample value is equal to the minimum (0x000) or the maximum (0xFFF) value and generates an interrupt if enabled. Similar to the range detect feature, saturate detect can also be enabled on a channel basis using the interrupt mask register, SAR_SATURATE_INTR_MASK. SAR_SATURATE_INTR[15:0] provides status of the interrupt and SAR_SATURATE_INTR_MASKED[15:0] provides the AND result of the interrupt status register and the MASK register. The interrupt can be triggered in firmware by writing to the SAR_SATURATE_INTR_SET[15:0] register. The SAR_SATURATE_INTR bit should be set to ‘1’ in the firmware to clear the interrupt.

20.5.5.3 End of Scan (EOS) Interrupt

This interrupt is triggered after scanning all the enabled channels. The interrupt is enabled by writing ‘1’ to the EOS_MASK bit in the SAR_INTR_MASK register. EOS_INTR in the SAR_INTR register provides the interrupt status. Note that, EOS_INTR is ANDed with EOS_MASK and then routed to the NVIC for interrupt. The AND version is available as EOS_MASKED bit in the SAR_INTR_MASKED register. EOS_INTR should be set to ‘1’ in the firmware after reading the result register to clear the interrupt. Interrupt can also be triggered in firmware by writing to the EOS_SET bit in the SAR_INTR_SET register.

20.5.5.4 Overflow Interrupt

If the scanning of all the enabled channels is complete, hardware writes to the EOS_INTR bit. If this bit is already high, OVERFLOW_INTR bit in the SAR_INTR register is set, indicating that the previous unread result has been overwritten with the new value. This can generate an interrupt if the OVERFLOW_INTR_MASK in the SAR_INTR_MASK register is set to ‘1’. Similar to other interrupts, the AND result of OVERFLOW_INTR and OVERFLOW_INTR_MASK is available in the OVERFLOW_INTR_MASKED bit in the SAR_INTR_MASKED register. The interrupt is cleared by writing ‘1’ to the OVERFLOW_INTR bit in the SAR_INTR register. The interrupt can be triggered in firmware by writing ‘1’ to the OVERFLOW_SET bit in the SAR_INTR_SET register.

20.5.5.5 Collision Interrupt

If a new trigger is given while the SARSEQ is still busy with the scan started by the previous trigger, a collision interrupt is generated. This is to indicate that the result available from the current scan is for the previous trigger. As explained earlier, there are two ways to trigger an interrupt, through hardware and by firmware. Thus, there are corresponding two types of collision interrupts. FW_COLLISION_INTR bit in the SAR_INTR register is set if the firmware trigger is asserted while the SAR ADC is busy in the conversion process. Writing ‘1’ to the FW_COLLISION_MASK bit in the SAR_INTR_MASK register enables the interrupt. The logical AND of the FW_COLLISION_INTR and the FW_COLLISION_MASK bits are available as the FW_COLLISION_MASKED bit in the SAR_INTR_MASKED register. Similarly, there are DSI_INTR, DSI_INTR_MASK, and DSI_INTR_MASKED bits in the corresponding registers for the hardware trigger.

20.5.6 Low Power Behavior

SAR ADC works in the Active and Sleep mode of the device. It does not work in the Deep-Sleep mode as high-frequency clock sources are shut down.

20.6 Registers

Register	Description
SAR_CTRL	SAR Control Register
SAR_SAMPLE_CTRL	SAR Sample Control Register
SAR_SAMPLE_TIME01 and SAR_SAMPLE_TIME23	Sample time specification register
SAR_RANGE_THRES	Global range detect threshold register
SAR_RANGE_COND	Global range detect mode register
SAR_CHAN_EN	Channel enable control register
SAR_START_CTRL	Firmware trigger control register
SAR_CHAN_CONFIG	Channel configuration register
SAR_CHAN_WORK	Channel working data register
SAR_CHAN_RESULT	Channel result data register
SAR_CHAN_WORK_UPDATED	Channel working data status register
SAR_CHAN_RESULT_UPDATED	Channel result data status register
SAR_INTR	Interrupt request register
SAR_INTR_SET	Interrupt set request register
SAR_INTR_MASK	Interrupt mask register
SAR_INTR_MASKED	Interrupt masked request register
SAR_SATURATE_INTR	Saturate interrupt request register
SAR_SATURATE_INTR_SET	Saturate interrupt set request register
SAR_SATURATE_INTR_MASK	Saturate interrupt mask register
SAR_SATURATE_INTR_MASKED	Saturate interrupt masked request register
SAR_RANGE_INTR	Range interrupt request register
SAR_RANGE _INTR_SET	Range interrupt set request register
SAR_RANGE _INTR_MASK	Range interrupt mask register
SAR_RANGE _INTR_MASKED	Range interrupt masked request register
SAR_INTR_CAUSE	Interrupt cause register
SAR_STATUS	Status register
SAR_AVG_STAT	Averaging status register
SAR_MUX_SWITCH0	SARMUX firmware switch controls
SAR_MUX_SWITCH_CLEAR0	SARMUX firmware switch clear control
SAR_MUX_SWITCH1	SARMUX firmware switch controls
SAR_MUX_SWITCH_CLEAR1	SARMUX firmware switch clear control
SAR_MUX_SWITCH_HW_CTRL	SARMUX switch hardware control
SAR_MUX_SWITCH_STATUS	SARMUX switch status

21. Temperature Sensor

PSoC® Analog Coprocessor has an on-chip temperature sensor that is used to measure the internal die temperature. The sensor consists of a transistor connected in diode configuration.

21.1 Features

The temperature sensor has the following features:

- ± 5° Celsius accuracy over temperature range –40 °C to +85 °C
- 0.5° Celsius/LSB resolution (without amplification) when using a 12-bit SAR ADC with a 1.2-V reference
- 10 µs settling time

21.2 How it Works

The temperature sensor consists of a single bipolar junction transistor (BJT) in the form of a diode. Its base-to-emitter voltage (V_{BE}) has a strong dependence on temperature at a constant collector current and zero collector-base voltage. This property is used to calculate the die temperature by measuring the V_{BE} of the transistor using SAR ADC, as shown in Figure 21-1.

Figure 21-1. Temperature Sensing Mechanism

The analog output from the sensor (V_{BE}) is measured using the SAR ADC. Die temperature in °C can be calculated from the ADC results as given in the following equation:

$$\text{Temp} = (A \times SAR_{out} + 2^{10} xB) + T_{adjust}$$

Equation 21-1

- Temp is the slope compensated temperature in °C represented as Q16.16 fixed point number format.
- 'A' is the 16-bit multiplier constant. The value of A is determined using the PSoC Analog Coprocessor family characterization data of two point slope calculation. It is calculated as given in the following equation.

$$A = (\text{signed int})\left(2^{16}\left(\frac{100°C - (-40°C)}{SAR_{100°C} - SAR_{-40°C}}\right)\right)$$

Equation 21-2

  Where,

  SAR_{100C} = ADC counts at 100°C

  SAR_{-40C} = ADC counts at –40°C

  Constant 'A' is stored in a register SFLASH_SAR_TEMP_MULTIPLIER.

- 'B' is the 16-bit offset value. The value of B is determined on a per die basis by taking care of all the process variations and the actual bias current (I_{bias}) present in the chip. It is calculated as given in the following equation.

$$B = (\text{unsigned int})\left(2^{6} x 100°C - \left(\frac{A \times SAR_{100C}}{2^{10}}\right)\right)$$

Equation 21-3

  Where,

SAR_{100C} = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH_SAR_TEMP_OFFSET.

■ T_{adjust} is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating $T_{initial} = (A \times SAR_{out} + 2^{10} \times B)$. If it is greater than the center value (15°C), then T_{adjust} is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{100°C - 15°C} \times (100°C \times 2^{16} - T_{initial})\right)$$

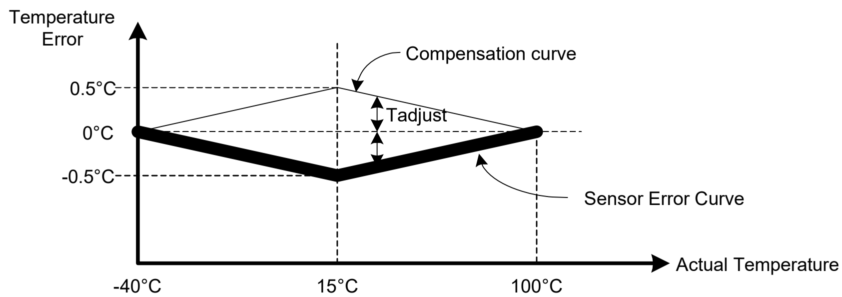
Equation 21-4

If less than center value, then Tadjust is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{40°C + 15°C} \times (40°C \times 2^{16} - T_{initial})\right)$$

Equation 21-5

Figure 21-2. Temperature Error Compensation



Note A and B are 16-bit constants stored in flash during factory calibration. Note that these constants are valid only when the SAR ADC is running at 12-bit resolution with a 1.2-V reference.

21.3 Temperature Sensor Configuration

The temperature sensor output is routed to the positive input of SAR ADC via dedicated switches, which can be controlled by sequencer, or firmware. See the SAR ADC chapter on page 183 for details on how to read the temperature sensor output using the ADC.

21.4 Algorithm

1. Enable the SARMUX and SAR ADC.
2. Configure SAR ADC in single-ended mode with $V_{NEG} = V_{SS}$, V_{REF} = 1.2 V, 12-bit resolution, and right-aligned result.
3. Enable the temperature sensor.
4. Get the digital output from the SAR ADC.
5. Fetch ‘A’ from SFLASH_SAR_TEMP_MULTIPLIER and ‘B’ from SFLASH_SAR_TEMP_OFFSET.
6. Calculate the die temperature using the linear equation (Equation 21-1).

   For example, let A = 0xBC4B and B = 0x65B4. Assume that the output of SAR ADC (V_{BE}) is 0x595 at a given temperature.

   Firmware does the following calculations:

   a. Multiply A and V_{BE}: 0xBC4B × 0x595 = $(–17333)_{10} \times (1429)_{10} = (–24768857)_{10}$
   b. Multiply B and 1024: 0x65B4 × 0x400 = $(26036)_{10} \times (1024)_{10} = (26660864)_{10}$
   c. Add the result of steps 1 and 2 to get $T_{initial}$: $(–24768857)_{10} + (26660864)_{10} = (1892007)_{10}$ = 0x1CDEA7
   d. Calculate T_{adjust} using $T_{initial}$ value: $T_{initial}$ is the upper 16 bits multiplied by 2^{16}, that is, 0x1C00 = $(1835008)_{10}$. It is greater than 15°C (0x1C - upper 16 bits). Use Equation 4 to calculate T_{adjust}. It comes to 0x6C6C = $(27756)_{10}$

e. Add T_{adjust} to $T_{initial}$: $(1892007)_{10} + (27756)_{10} = (1919763)_{10}$ = 0x1D4B13

f. The integer part of temperature is the upper 16 bits = 0x001D = $(29)_{10}$

g. The decimal part of temperature is the lower 16 bits = 0x4B13 = $(0.19219)_{10}$

h. Combining the result of steps f and g, Temp = 29.19219 °C ~ 29.2°C

21.5 Registers

Name	Description
SAR_MUX_SWITCH0	This register has the SAR_MUX_FW_TEMP_VPLUS field to connect the temperature sensor to the SAR MUX terminal.
SAR_MUX_SWITCH_STATUS	This register provides the status of the temperature sensor switch connection to SAR MUX.
SFLASH_SAR_TEMP_MULTIPLIER	Multiplier constant 'A' as defined in Equation 21-1.
SFLASH_SAR_TEMP_OFFSET	Constant 'B' as defined in Equation 21-1.

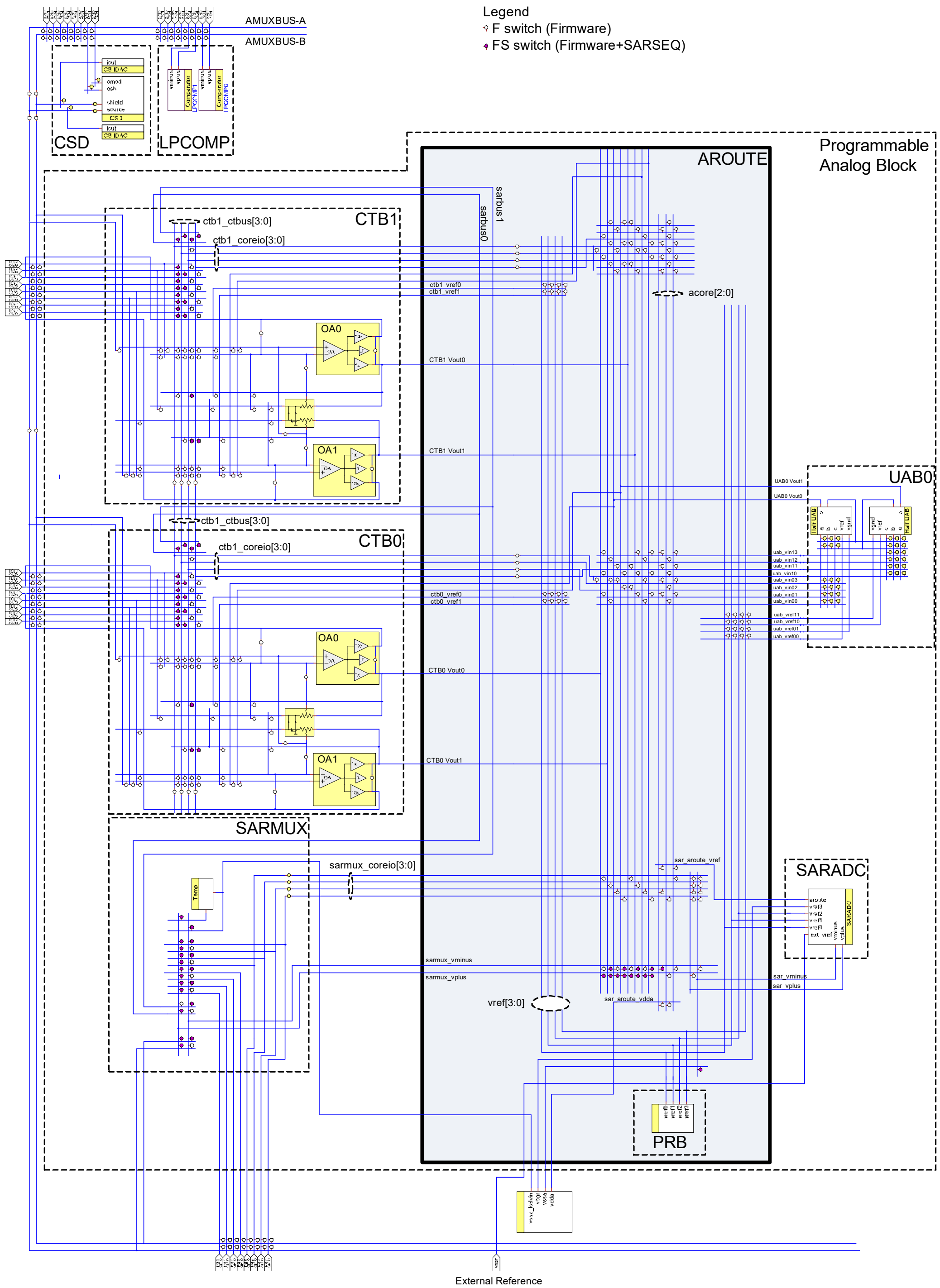
22. Analog Routing [AROUTE]

AROUTE consists of routing lines and analog switches that connect the analog peripherals – CTB, SARMUX, SAR, UAB, and PRB. Figure 22-1 shows how the AROUTE interconnects the analog peripherals. With extensive routing, it is possible to implement a variety of analog signal chains with minimal external wiring. For example, in an interface to an analog sensor, its output can be amplified using the CTB and fed to the UAB for filtering. The UAB output can then be wired to the SAR ADC for measurement. The CTB, UAB, and SAR ADC can receive the reference voltage from the PRB. All these interconnections are made possible within the device by the AROUTE fabric.

Some of the important features of AROUTE are as follows:

- Interconnections between any two analog blocks happen through AROUTE; an exception is the interconnection of the CTB opamps through ctbus[3:0] lines.
- SAR ADC inputs are always taken from AROUTE.
- The F switches in AROUTE are controlled only by firmware and the FS switches are controllable by firmware and the SAR sequencer (see Figure 22-1).
- AROUTE is capable of operating in all device power modes including the Deep-Sleep mode.

Figure 22-1. Analog Routing Block Diagram



22.1 Enabling AROUTE

The global AROUTE control register ART_CTRL is used to enable and disable the AROUTE fabric.

Table 22-1. Global AROUTE Control Register – ART_CTRL

ENABLED ART_CTRL[31]	DEEPSLEEP_ON ART_CTRL[30]	Description
0	0	AROUTE is disabled. All switches are opened.
0	1	AROUTE is disabled. All switches are opened.
1	0	AROUTE is enabled only in active and sleep mode.
1	1	AROUTE is enabled in all device power modes

The charge pump circuit needs to be enabled if VDDA is less than 4 V. This is to keep the ON state resistance of the switches to a low value. The charge pump is enabled using PUMP_SEL bits of the CLK_SELECT register. Note that in Deep-Sleep mode, the charge pump is inactive and thus causes higher ON state switch resistance for VDDA less than 4 V.

22.1.1 Switch Control

AROUTE has two types of switches F (firmware only) and FS (firmware and sequencer). Each switch has a control bit. Table 22-2 summarizes the registers that connect point 1 and point 2. Details of each register can be obtained from the *PSoC Analog Coprocessor Family Registers TRM*.

Table 22-2. AROUTE Connections

Register	Point 1	Point 2
ART_SARMUXVPLUS_SW	sarmux_vplus	ctb0_vout0, ctb0_vout1, ctb1_vout0, ctb1_vout1, uab0_vout0, uab0_vout1, acore[1] or sar_vplus
ART_SARMUXVMINUS_SW	sarmux_vminus	ctb0_vout0, ctb0_vout1, ctb1_vout0, ctb1_vout1, uab0_vout0, uab0_vout1, acore[0], acore[2], sar_vminus or VSSA
ART_SARMUXCOREIO0_SW	sarmux_coreio[0]	ctb0_vout0, ctb1_vout0, acore[2], sar_vplus or sar_vminus
ART_SARMUXCOREIO1_SW	sarmux_coreio[1]	ctb0_vout1, ctb1_vout1, sar_vplus or sar_vminus
ART_SARMUXCOREIO2_SW	sarmux_coreio[2]	uab0_vout0, acore[0], sar_vplus or sar_vminus
ART_SARMUXCOREIO3_SW	sarmux_coreio[3]	uab0_vout1, acore[1], sar_vplus or sar_vminus
ART_SARAROUTEVREF_SW	sar_aroute_vref	acore[0] or acore[2]
ART_SARAROUTEVDDA_SW	sar_aroute_vdda	acore[0] or acore[1]
ART_UAB0VIN00_SW	uab0_vin00	ctb0_vout1 or acore[1]
ART_UAB0VIN01_SW	uab0_vin01	ctb0_vout0 , ctb1_vout1, acore[0] or acore[2]
ART_UAB0VIN02_SW	uab0_vin02	uab0_vout1, ctb1_vout0, acore[1] or ctb0_ctbus[2]
ART_UAB0VIN03_SW	uab0_vin03	ctb0_vout1, uab0_vout0, acore[0], acore[2], uab0_vin10 or ctb0_ctbus1
ART_UAB0VIN10_SW	uab0_vin10	ctb0_vout0, ctb1_vout1, uab0_vout1, acore[1] or ctb0_ctbus[0]
ART_UAB0VIN11_SW	uab0_vin11	ctb1_vout0, uab0_vout0, acore[0], acore[2] or ctb0_ctbus3
ART_UAB0VIN12_SW	uab0_vin12	ctb0_vout1, uab1_vout0, or acore[1]

Table 22-2. AROUTE Connections

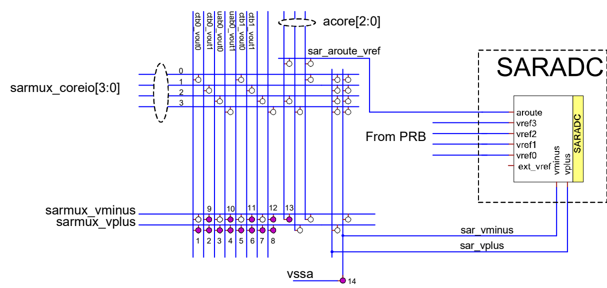
Register	Point 1	Point 2
ART_UAB0VIN13_SW	uab0_vin13	ctb0_vout0, ctb1_vout1, acore[0] or acore[2]
ART_UAB0VREF_SW	uab0_vref00, uab0_vref01 uab0_vref10, uab0_vref11	vref0, vref1, vref2, vref3
ART_CTB0VREF_SW	ctb0_vref0, ctb0_vref1	vref0, vref1, vref2, vref3
ART_CTB1VREF_SW	ctb1_vref0, ctb1_vref1	vref0, vref1, vref2, vref3

Note that each of these signals is named in Figure 22-1.

22.1.2 Sequencer Control

The FS switches in the AROUTE can be controlled by both firmware and hardware. Switches 1 to 13 are the FS switches. Figure 22-2 shows the relevant section of the AROUTE architecture diagram.

Figure 22-2. FS Switches



The FS switches are used to connect the SAR ADC inputs (sar_plus and sar_minus) to the analog resources. These switches are controlled by firmware using registers ART_SARMUXVPLUS_SW and ART_SARMUXVMINUS_SW. To allow these switches to be controlled by the SAR sequencer, the ART_SARMUX_SW_HW_CTRL register is used.

Table 22-3. ART_SARMUX_SW_HW_CTRL Register

ART_SARMUX_SW_HW_CTRL bit	Name	Switch number	Description
0	SW_C0O0	1	Connects sarmux_vplus to ctb0_vout0
1	SW_C0O1	2 and 9	Connects sarmux_vplus or sarmux_vminus to ctb0_vout1
2	SW_U0O0	3	Connects sarmux_vplus to uab0_vout0
3	SW_U0O1	4 and 10	Connects sarmux_vplus or sarmux_vminus to uab0_vout0
4	SW_C1O0	5	Connects sarmux_vplus to ctb1_vout0
5	SW_C1O1	6 and 11	Connects sarmux_vplus or sarmux_vminus to ctb1_vout1
16	SW_AU0	13	Connects sarmux_vminus to acore[0]
17	SW_VSSA	14	Connects sarmux_vminus to VSSA

Note: The corresponding firmware control bits in ART_SARMUXVPLUS_SW and ART_SARMUXVMINUS_SW act as mask bits when FS switch is configured for hardware control. Thus, the controls for the sarmux_vminus switches are shared with the sarmux_vplus switches.

23. Low-Power Comparator

PSoC® Analog Coprocessor devices have two low-power comparators. These comparators can perform fast analog signal comparison in all system power modes. Refer to the Power Modes chapter on page 88 for details on various device power modes. The positive and negative inputs can be connected to dedicated GPIO pins or to AMUXBUS-A/AMUXBUS-B. The comparator output can be read by the CPU through a status register, used as an interrupt or wakeup source or routed to a GPIO.

23.1 Features

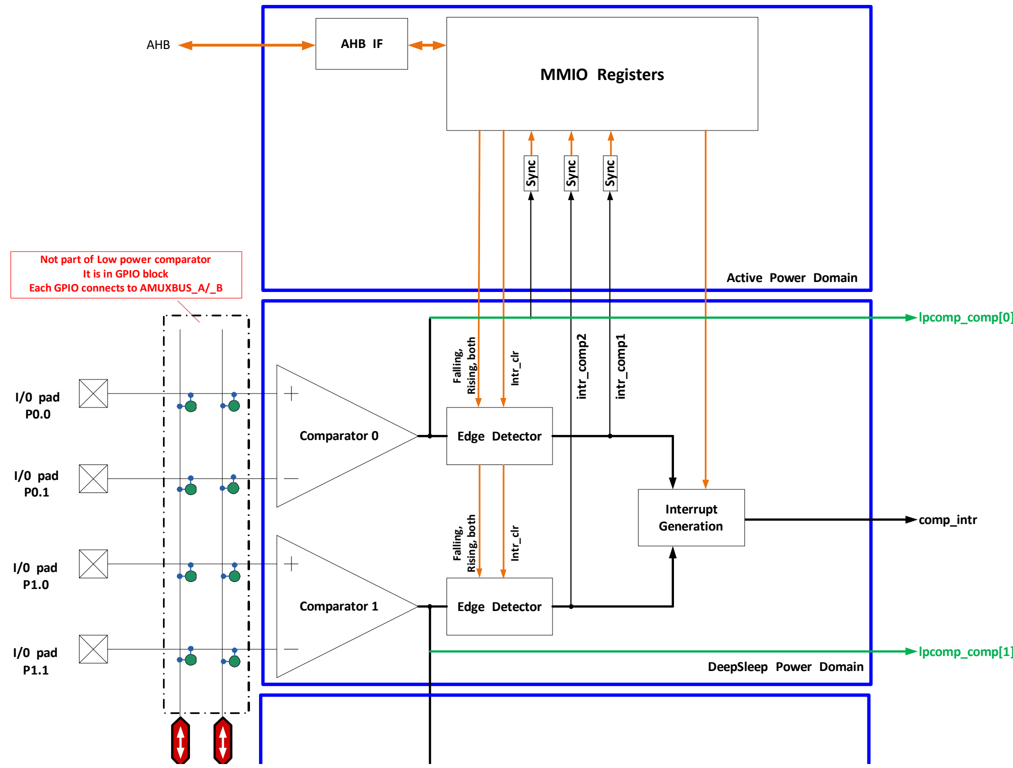
PSoC comparators have the following features:

- Configurable positive and negative inputs
- Programmable power and speed
- Ultra low-power mode support (<4 µA)
- Optional 10-mV input hysteresis
- Low-input offset voltage (<4 mV after trim)
- Wakeup source in Deep-Sleep mode

23.2 Block Diagram

Figure 23-1 shows the block diagram for the low-power comparator.

Figure 23-1. Low-Power Comparator Block Diagram



23.3 How It Works

The following sections describe the operation of the PSoC low-power comparator, including input configuration, power and speed mode, output and interrupt configuration, hysteresis, wake up from low-power modes, comparator clock, and offset trim.

23.3.1 Input Configuration

Inputs to the comparators can be as follows:

- Both positive and negative inputs from dedicated input pins.
- Both positive and negative inputs from any pin through AMUXBUS (not available in Deep-Sleep mode).
- One input from an external pin and another input from an internally-generated signal. Both inputs can be connected to either positive or negative inputs of the comparator. The internally-generated signal is connected to the comparator input through the analog AMUXBUS.
- Both positive and negative inputs from internally-generated signals. The internally-generated signals are connected to the comparator input through AMUXBUS-A/AMUXBUS-B.

From Figure 23-1, note that P0.0 and P0.1 connect to positive and negative inputs of Comparator 0; P1.0 and P1.1 connect to the inputs of Comparator 1. Also, note that the AMUXBUS nets do not have a direct connection to the comparator inputs. Therefore, the comparator connection is routed to the AMUXBUS nets through the corresponding input pin. These input pins will not be available for other purposes when using AMUXBUS for comparator connections. They should be left open in designs that use AMUXBUS for comparator input connection. Note that AMUXBUS connections are not available in Deep-Sleep mode. If Deep-Sleep operation is required, the low-power comparator must be connected to the dedicated pins. This restriction also includes routing of any internally-generated signal, which uses the AMUXBUS for the connection. See the I/O System chapter on page 57 for more details on connecting the GPIO to AMUXBUS A/B or setting up the GPIO for comparator input.

23.3.2 Output and Interrupt Configuration

The output of Comparator0 and Comparator1 are available in the OUT1 bit [6] and OUT2 bit [14], respectively, in the LPCOMP_CONFIG register (Table 23-1). The comparator outputs are synchronized to SYSCLK before latching them to the OUTx bits in the LPCOMP_CONFIG register. The output of each comparator is connected to a corresponding edge detector block. This block determines the edge that triggers the interrupt. The edge selection and interrupt enable is configured using the INTTYPE1 bits [5:4] and INTTYPE2 bits [13:12] in the LPCOMP_CONFIG register. Using the INTTYPEx bits, the interrupt type can be selected to disabled, rising edge, falling edge, or both edges, as described in Table 23-1.

Each comparator's output can be routed directly to a GPIO pin through the HSIOM. The comparator outputs are available as Deep-Sleep source 2 connection in the HSIOM. See High-Speed I/O Matrix on page 62 for details on HSIOM. For details on the pins that support the low-power comparator output, refer to the device datasheet. The output on these pins are direct output from the comparator and are not synchronized. Because they act as Deep-Sleep source for the pins, the comparator output is available in Deep-Sleep power mode as well.

During an edge event, the comparator will trigger an interrupt (intr_comp1/intr_comp2 signals in Figure 23-1). The interrupt request is registered in the COMP1 bit [0] and COMP2 bit [1] of the LPCOMP_INTR register for Comparator0 and Comparator1, respectively. Both Comparator0 and Comparator1 share a common interrupt (comp_intr signal in Figure 23-1), which is a logical OR of the two interrupts and mapped as the low-power comparator block's interrupt in the CPU NVIC. Refer to the Interrupts chapter on page 47 for details. If both the comparators are used in a design, the COMP1 and/or COMP2 bits of the LPCOMP_INTR register need to be read in the interrupt service routine to know which one triggered the interrupt. Alternatively, COMP1_MASK bit [0] and COMP2_MASK bit [1] of the LPCOMP_INTR_MASK register can be used to mask the Comparator0 and Comparator1 interrupts to the CPU. Only the masked interrupts will be serviced by the CPU. After the interrupt is processed, the interrupt should be cleared by writing a '1' to the COMP1 and COMP2 bits of the LPCOMP_INTR register in firmware. If the interrupt is not cleared, the next compare event will not trigger an interrupt and the CPU will not be able to process the event..

The LPCOMP interrupt (comp1_intr/comp2_intr) is synchronous with SYSCLK. Clearing comp1_intr/comp2_intr are all synchronous.

LPCOMP_INTR_SET register bits [1:0] can be used to assert an interrupt for software debugging.

In Deep-Sleep mode, the wakeup interrupt controller (WIC) can be activated by a comparator edge event, which then wakes up the CPU. Thus, the LPCOMP has the capability to monitor a specified signal in low-power modes.

Table 23-1. Output and Interrupt Configuration in LPCOMP_CONFIG Register

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[6]	OUT1	Current/Instantaneous output value of Comparator0
LPCOMP_CONFIG[14]	OUT2	Current/Instantaneous output value of Comparator1
LPCOMP_CONFIG[5:4]	INTTYPE1	Sets on which edge Comparator0 will trigger an IRQ 00: Disabled 01: Rising Edge 10: Falling Edge 11: Both rising and falling edges
LPCOMP_CONFIG[13:12]	INTTYPE2	Sets on which edge Comparator1 will trigger an IRQ 00: Disabled 01: Rising Edge 10: Falling Edge 11: Both rising and falling edges
LPCOMP_INTR[0]	COMP1	Comparator0 Interrupt: hardware sets this interrupt when Comparator0 triggers. Write a '1' to clear the interrupt
LPCOMP_INTR[1]	COMP2	Comparator2 Interrupt: hardware sets this interrupt when Comparator1 triggers. Write a '1' to clear the interrupt
LPCOMP_INTR_SET[0]	COMP1	Write a '1' to trigger the software interrupt for Comparator0
LPCOMP_INTR_SET[1]	COMP2	Write a 1 to trigger the software interrupt for Comparator1

23.3.3 Power Mode and Speed Configuration

The low-power comparators can operate in three power modes:

- Fast
- Slow
- Ultra low-power

The power or speed setting for Comparator0 is configured using MODE1 bits [1:0] in the LPCOMP_CONFIG register. The power or speed setting for Comparator1 is configured using MODE2 bits [9:8] in the same register. The power consumption and response time vary depending on the selected power mode; power consumption is highest in fast mode and lowest in ultra-low-power mode, response time is fastest in fast mode and slowest in ultra-low-power mode. Refer to the device datasheet for specifications for the response time and power consumption for various power settings.

The comparators are enabled/disabled using ENABLE1 bit [7] and ENABLE2 bit [15] in the LPCOMP_CONFIG register, as described in Table 23-2.

Note The output of the comparator may glitch when the power mode is changed while comparator is enabled. To avoid this, disable the comparator before changing the power mode.

Table 23-2. Comparator Power Mode Selection Bits MODE1 and MODE2

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[1:0]	MODE1	Compartor0 power mode selection 00: Slow operating mode (uses less power) 01: Fast operating mode (uses more power) 10: Ultra low-power operating mode (uses lowest possible power)
LPCOMP_CONFIG[9:8]	MODE2	Compartor1 power mode selection 00: Slow operating mode (uses less power) 01: Fast operating mode (uses more power) 10: Ultra low-power operating mode (uses lowest possible power)
LPCOMP_CONFIG[7]	ENABLE1	Comparator0 enable bit 0: Disables Comparator0 1: Enables Comparator0
LPCOMP_CONFIG[15]	ENABLE2	Comparator1 enable bit 0: Disables Comparator1 1: Enables Comparator1

23.3.4 Hysteresis

For applications that compare signals close to each other and slow changing signals, hysteresis helps to avoid oscillations at the comparator output when the signals are noisy. For such applications, a fixed 10-mV hysteresis may be enabled in the comparator block.

The 10-mV hysteresis level is enabled/disabled by using the HYST1 bit [2] and HYST2 bit [10] in the LPCOMP_CONFIG register, as described in Table 23-3.

Table 23-3. Hysteresis Control Bits HYST1 and HYST2

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[2]	HYST1	Enable/Disable 10 mV hysteresis to Comparator0 - 0: Enable Hysteresis - 1: Disable Hysteresis
LPCOMP_CONFIG[10]	HYST2	Enable/Disable 10 mV hysteresis to Comparator1 - 0: Enable Hysteresis - 1: Disable Hysteresis

23.3.5 Wakeup from Low-Power Modes

The comparator is operational in the device's low-power modes, including Sleep and Deep-Sleep modes. The comparator output interrupt can wake the device from Sleep and Deep-Sleep modes. The comparator should be enabled in the LPCOMP_CONFIG register, the INTTYPEx bits in the LPCOMP_CONFIG register should not be set to disabled, and the INTR_MASKx bit should be set in the LPCOMP_INTR_MASK register for the corresponding comparator to wake the device from low-power modes. Comparisons involving AMUXBUS connections are not available in DeepSleep mode.

In the Deep-Sleep power mode, a compare event on either Comparator0 or Comparator1 output will generate a wakeup interrupt. The INTTYPEx bits in the LPCOMP_CONFIG register should be configured, as required, for the corresponding comparator to wake the device from low-power modes. The mask bits in the LPCOMP_INTR_MASK register is used to select whether one or both of the comparator's interrupt is serviced by the CPU.

23.3.6 Comparator Clock

The comparator uses the system main clock SYSCLK as the clock for interrupt synchronization.

23.3.7 Offset Trim

The comparator offset is trimmed at the factory to less than 4.0 mV. The trim is a two-step process, trimmed first at common mode voltage equal to 0.1 V, then at common mode voltage equal to V_{DD}–0.1 V. Offset voltage is guaranteed to be less than 10.0 mV over the input voltage range of 0.1 V to V_{DD}–0.1 V. For normal operation, further adjustment of trim values is not recommended.

If a tighter trim is required at a specific input common mode voltage, a trim may be performed at the desired input common mode voltage. The comparator offset trim is performed using the LPCOMP_TRIM1/2/3/4 registers. LPCOMP_TRIM1 and LPCOMP_TRIM2 are used to trim comparator 0. LPCOMP_TRIM3 and LPCOMP_TRIM4 are used to trim comparator 1. The bit fields that change the trim values are TRIMA bits [4:0] in LPCOMP_TRIM1 and LPCOMP_TRIM3, and TRIMB bits [3:0] in LPCOMP_TRIM2 and LPCOMP_TRIM4. TRIMA bits are used to coarse tune the offset; TRIMB bits are used to fine tune. The use of TRIMB bits for offset correction is restricted to slow mode of comparator operation.

Any standard comparator offset trim procedure can be used to perform the trimming. The following method can be used to improve the offset at a given reference/common mode voltage input.

1. Short the comparator inputs externally and connect the voltage reference, V_{ref}, to the input.
2. Set up the comparator for comparison, turn off hysteresis, and check the output.
3. If the output is high, the offset is positive. Otherwise, the offset is negative. Follow these steps to tune the offset:
    a. Tune the TRIMA bits[4:0] until the output switches direction. TRIMA bits[3:0] control the amount of offset and TRIMA bit[4] controls the polarity of offset ('1' indicates positive offset and '0' indicates negative offset).
    b. When the tuning of TRIMA bits is complete, tune the TRIMB bits[3:0] until the output switches direction again. The TRIMB bit tuning is valid only for slow mode of comparator operation. TRIMB bit[3] controls the polarity of offset. Increasing TRIMB bits [2:0] reduces the offset.
    c. After completing step 3-b, the values available in the TRIMA and TRIMB bits will be the closest possible trim value for that particular V_{ref}.

23.4 Register Summary

Table 23-4. Low-Power Comparator Register Summary

Register	Function
LPCOMP_ID	Includes the information of LPCOMP controller ID and revision number
LPCOMP_CONFIG	LPCOMP configuration register
LPCOMP_INTR	LPCOMP interrupt register
LPCOMP_INTR_SET	LPCOMP interrupt set register
LPCOMP_INTR_MASK	LPCOMP interrupt request mask register
LPCOMP_INTR_MASKED	LPCOMP masked interrupt output register
LPCOMP_TRIM1	Trim fields for comparator 0
LPCOMP_TRIM2	Trim fields for comparator 0
LPCOMP_TRIM3	Trim fields for comparator 1
LPCOMP_TRIM4	Trim fields for comparator 1

24. LCD Direct Drive

The PSoC® Analog Coprocessor Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PSoC device to directly drive STN and TN segment LCDs.

24.1 Features

The PSoC LCD segment drive block has the following features:

- Supports up to 30 segments and eight commons
- Supports Type A (standard) and Type B (low-power) drive waveforms
- Any GPIO can be configured as a common or segment
- Supports five drive methods:
  - Digital correlation
  - PWM at 1/2 bias
  - PWM at 1/3 bias
  - PWM at 1/4 bias
  - PWM at 1/5 bias
- Ability to drive 3-V displays from 1.8 V V_{DD} in Digital Correlation mode
- Operates in active, sleep, and deep-sleep modes
- Digital contrast control

24.2 LCD Segment Drive Overview

A segmented LCD panel has the liquid crystal material between two sets of electrodes and various polarization and reflector layers. The two electrodes of an individual segment are called commons (COM) or backplanes and segment electrodes (SEG). From an electrical perspective, an LCD segment can be considered as a capacitive load; the COM/SEG electrodes can be considered as the rows and columns in a matrix of segments. The opacity of an LCD segment is controlled by varying the root-mean-square (RMS) voltage across the corresponding COM/SEG pair.

The following terms/voltages are used in this chapter to describe LCD drive:

- $\mathbf{V_{RMSOFF}}$: The voltage that the LCD driver can realize on segments that are intended to be off.
- $\mathbf{V_{RMSON}}$: The voltage that the LCD driver can realize on segments that are intended to be on.
- **Discrimination Ratio (D)**: The ratio of V_{RMSON} and V_{RMSOFF} that the LCD driver can realize. This depends on the type of waveforms applied to the LCD panel. Higher discrimination ratio results in higher contrast.

Liquid crystal material does not tolerate long term exposure to DC voltage. Therefore, any waveforms applied to the panel must produce a 0-V DC component on every segment (on or off). Typically, LCD drivers apply waveforms to the COM and SEG electrodes that are generated by switching between multiple voltages. The following terms are used to define these waveforms:

- **Duty**: A driver is said to operate in 1/M duty when it drives 'M' number of COM electrodes. Each COM electrode is effectively driven 1/M of the time.
- **Bias**: A driver is said to use 1/B bias when its waveforms use voltage steps of (1/B) × VDRV. VDRV is the highest drive voltage in the system (equals to V_{DD} in PSoC). PSoC supports 1/2, 1/3, 1/4, and 1/5 biases in PWM drive modes.
- **Frame**: A frame is the length of time required to drive all the segments. During a frame, the driver cycles through the commons in sequence. All segments receive 0-V DC (but non-zero RMS voltage) when measured over the entire frame.

PSoC supports two different types of drive waveforms in all drive modes. These are:

- **Type-A Waveform**: In this type of waveform, the driver structures a frame into M sub-frames. 'M' is the number of COM electrodes. Each COM is addressed only once during a frame. For example, COM[i] is addressed in sub-frame i.
- **Type-B Waveform**: The driver structures a frame into 2M sub-frames. The two sub-frames are inverses of each other. Each COM is addressed twice during a frame. For example, COM[i] is addressed in sub-frames i and M+i. Type-B waveforms are slightly more power efficient because it contains fewer transitions per frame.

24.2.1 Drive Modes

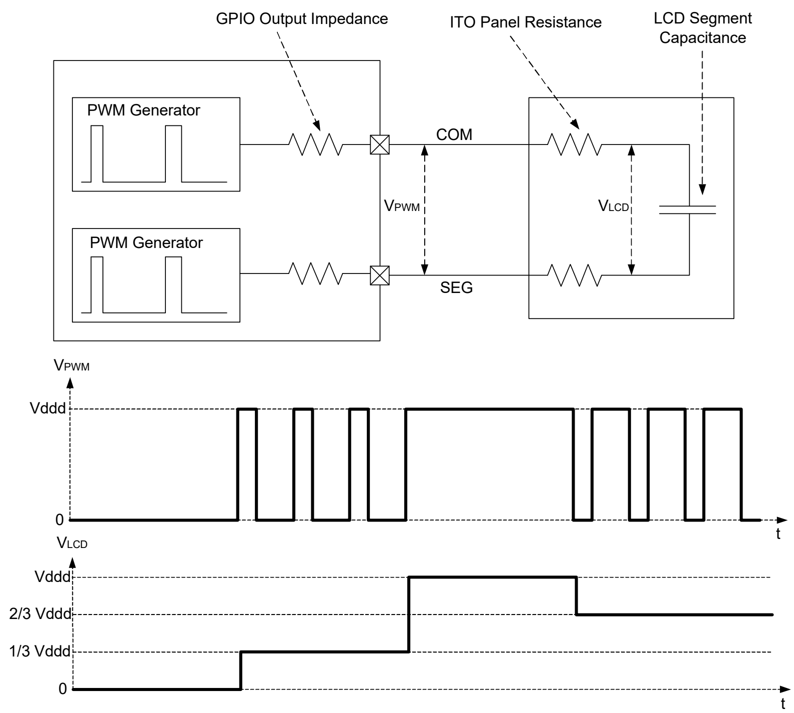
PSoC supports the following drive modes.

- PWM drive at 1/2 bias
- PWM drive at 1/3 bias
- PWM drive at 1/4 bias with high-frequency clock input
- PWM drive at 1/5 bias with high-frequency clock input
- Digital correlation

24.2.1.1 PWM Drive

In PWM drive mode, multi-voltage drive signals are generated using a PWM output signal together with the intrinsic resistance and capacitance of the LCD. Figure 24-1 illustrates this.

Figure 24-1. PWM Drive (at 1/3 Bias)



The output waveform of the drive electronics is a PWM waveform. With the Indium Tin Oxide (ITO) panel resistance and the segment capacitance to filter the PWM, the voltage across the LCD segment is an analog voltage, as shown in Figure 24-1. This figure illustrates the generation of a 1/3 bias waveform (four commons and voltage steps of $V_{DD}/3$).

The PWM is derived from either ILO (32 kHz, low-speed operation) or IMO (high-speed operation). The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 24-2 and Figure 24-3 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/2 bias and 1/4 duty. Only COM0/COM1 and SEG0/SEG1 are drawn for demonstration purpose. Similarly, Figure 24-4 and Figure 24-5 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/3 bias and 1/4 duty.

Figure 24-2. PWM1/2 Type-A Waveform Example

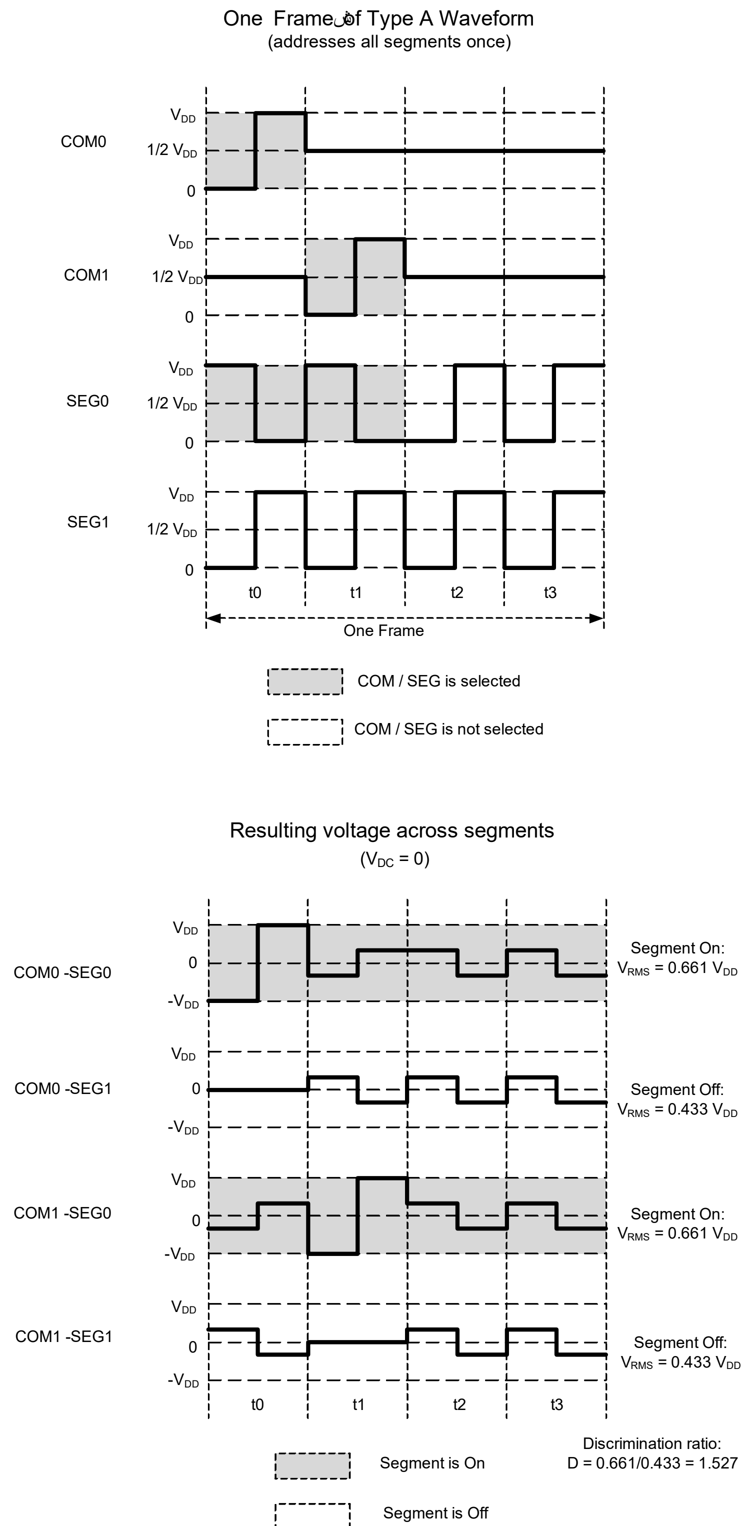


Figure 24-3. PWM1/2 Type-B Waveform Example

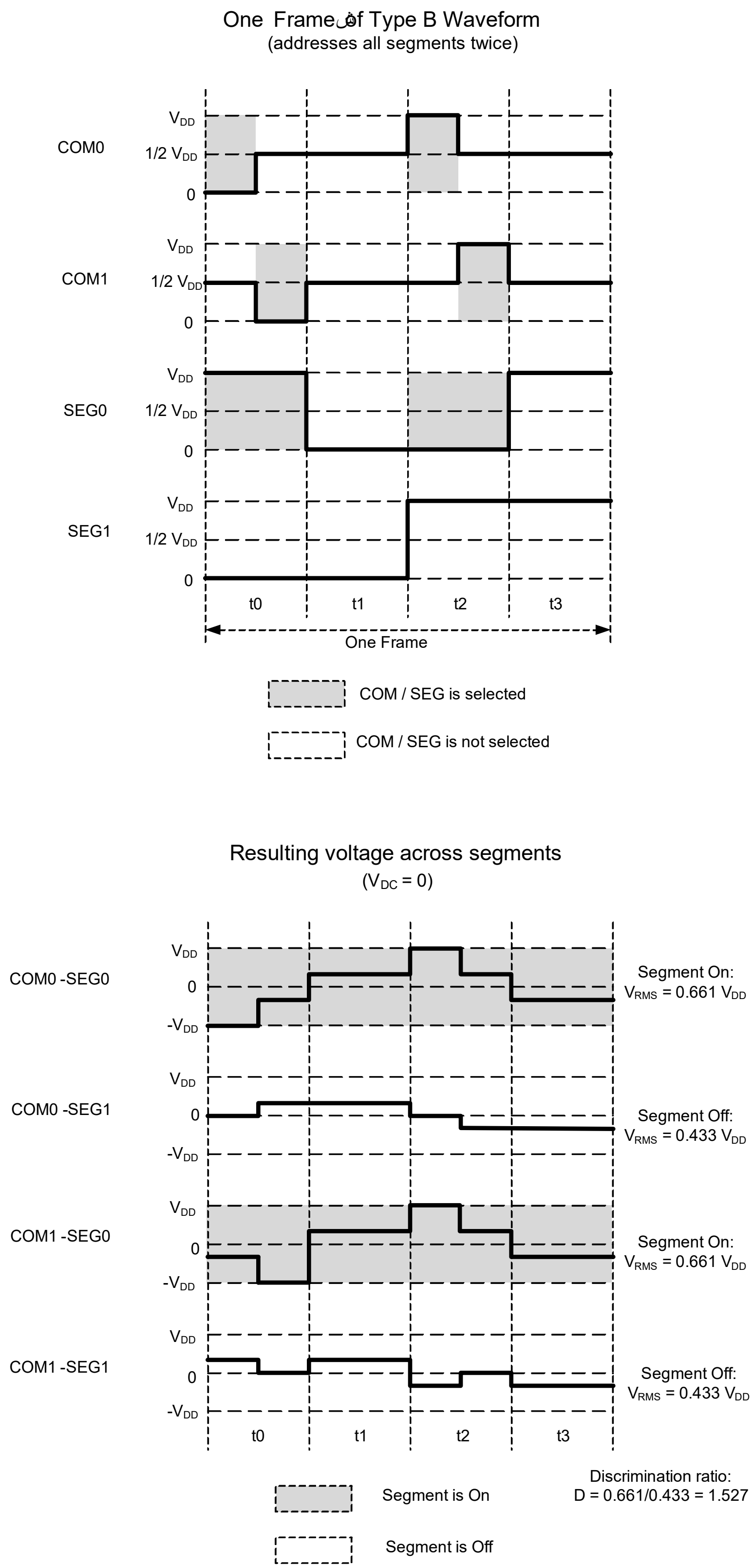


Figure 24-4. PWM1/3 Type-A Waveform Example

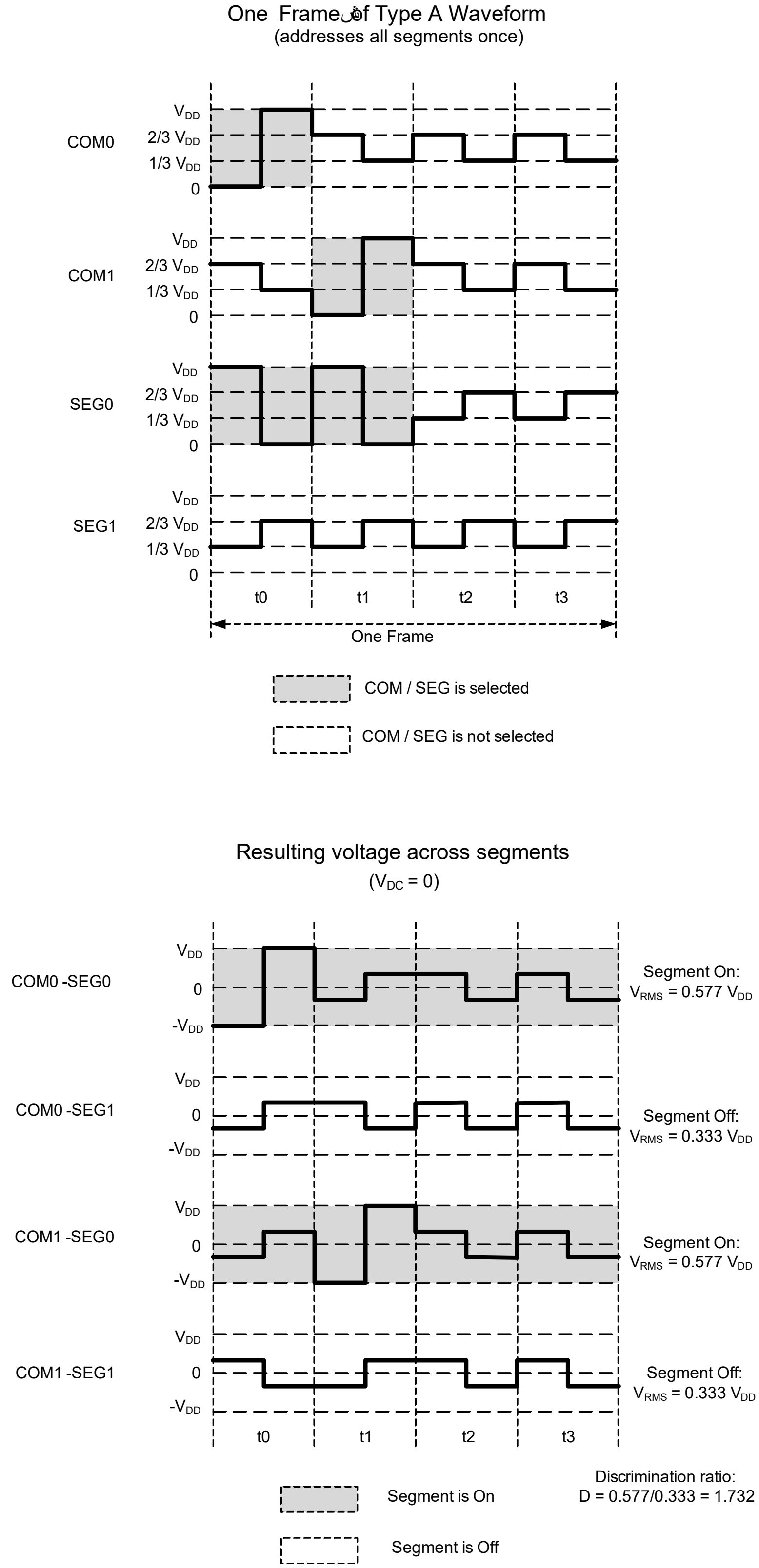
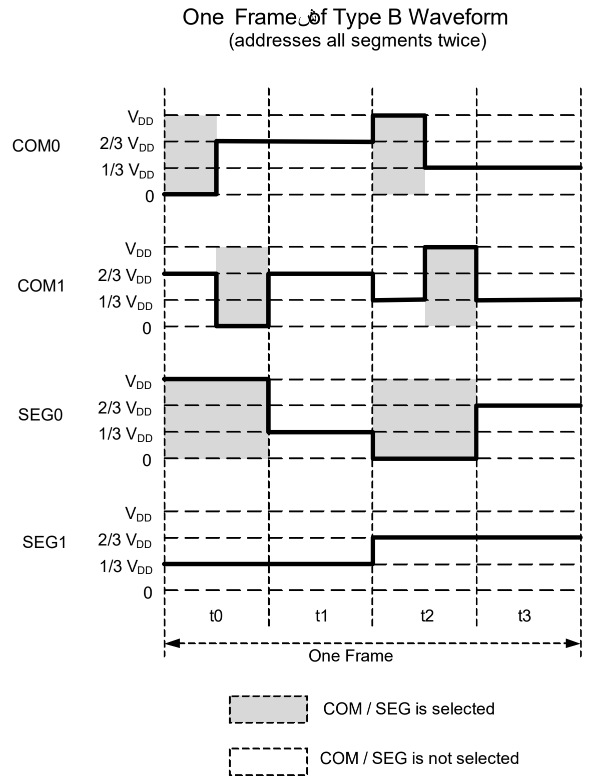
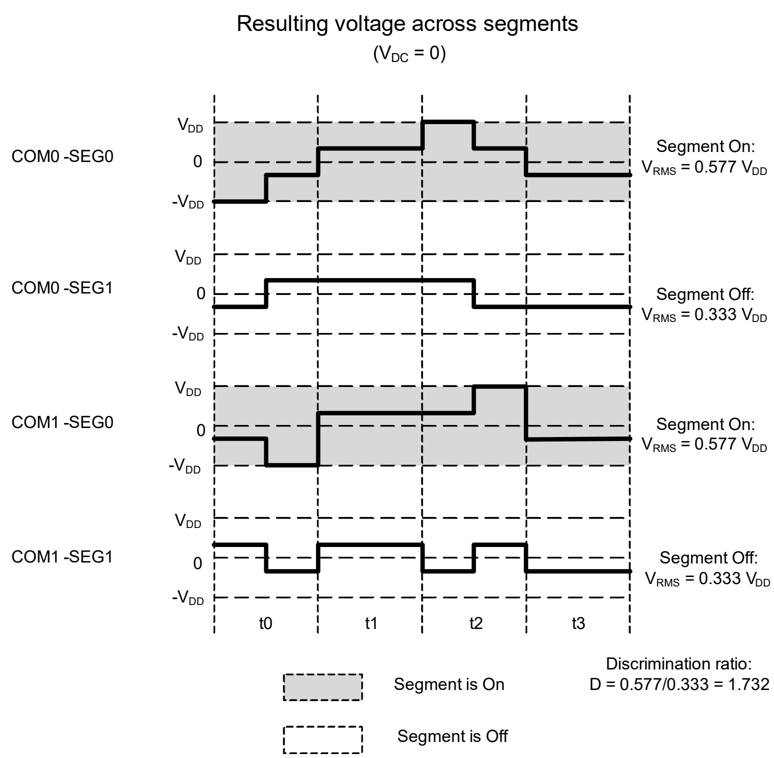


Figure 24-5. PWM1/3 Type-B Waveform Example





The effective RMS voltage for ON and OFF segments can be calculated easily using these equations:

$$V_{RMS(OFF)} = \sqrt{\frac{2(B-2)^2 + 2(M-1)}{2M}} x \left(\frac{V_{DRV}}{B}\right)$$

Equation 24-1

$$V_{RMS(ON)} = \sqrt{\frac{2B^2 + 2(M-1)}{2M}} x \left(\frac{V_{DRV}}{B}\right)$$

Equation 24-2

Where B is the bias and M is the duty (number of COMs).

For example, if the number of COMs is four, the resulting discrimination ratios (D) for 1/2 and 1/3 biases are 1.528 and 1.732, respectively. 1/3 bias offers better discrimination ratio in two and three COM drives also. Therefore, 1/3 bias offers better contrast than 1/2 bias and is recommended for most applications. 1/4 and 1/5 biases are available only in high-speed operation of the LCD. They offer better discrimination ratio especially when used with high COM designs (more than four COMs).

When the low-speed operation of LCD is used, the PWM signal is derived from the ILO. To drive a low-capacitance display with acceptable ripple and rise/fall times using a 32-kHz PWM, additional external series resistances of 100 k-1 MΩ should be used. External resistors are not required for PWM frequencies greater than ~1 MHz. The ideal PWM frequency depends on the capacitance of the display and the internal ITO resistance of the ITO routing traces.

The 1/2 bias mode has the advantage that PWM is only required on the COM signals; the SEG signals use only logic levels, as shown in Figure 24-2 and Figure 24-3.

24.2.1.2 Digital Correlation

The digital correlation mode, instead of generating bias voltages between the rails, takes advantage of the characteristic of LCDs that the contrast of LCD segments is determined by the RMS voltage across the segments. In this approach, the correlation coefficient between any given pair of COM and SEG signals determines whether the corresponding LCD segment is on or off. Thus, by doubling the base drive frequency of the COM signals in their inactive sub-frame intervals, the phase relationship of the COM and SEG drive signals can be varied to turn segments on and off. This is different from varying the DC levels of the signals as in the PWM drive approach. Figure 24-8 and Figure 24-9 are example waveforms that illustrate the principles of operation.

Figure 24-6. Digital Correlation Type-A Waveform

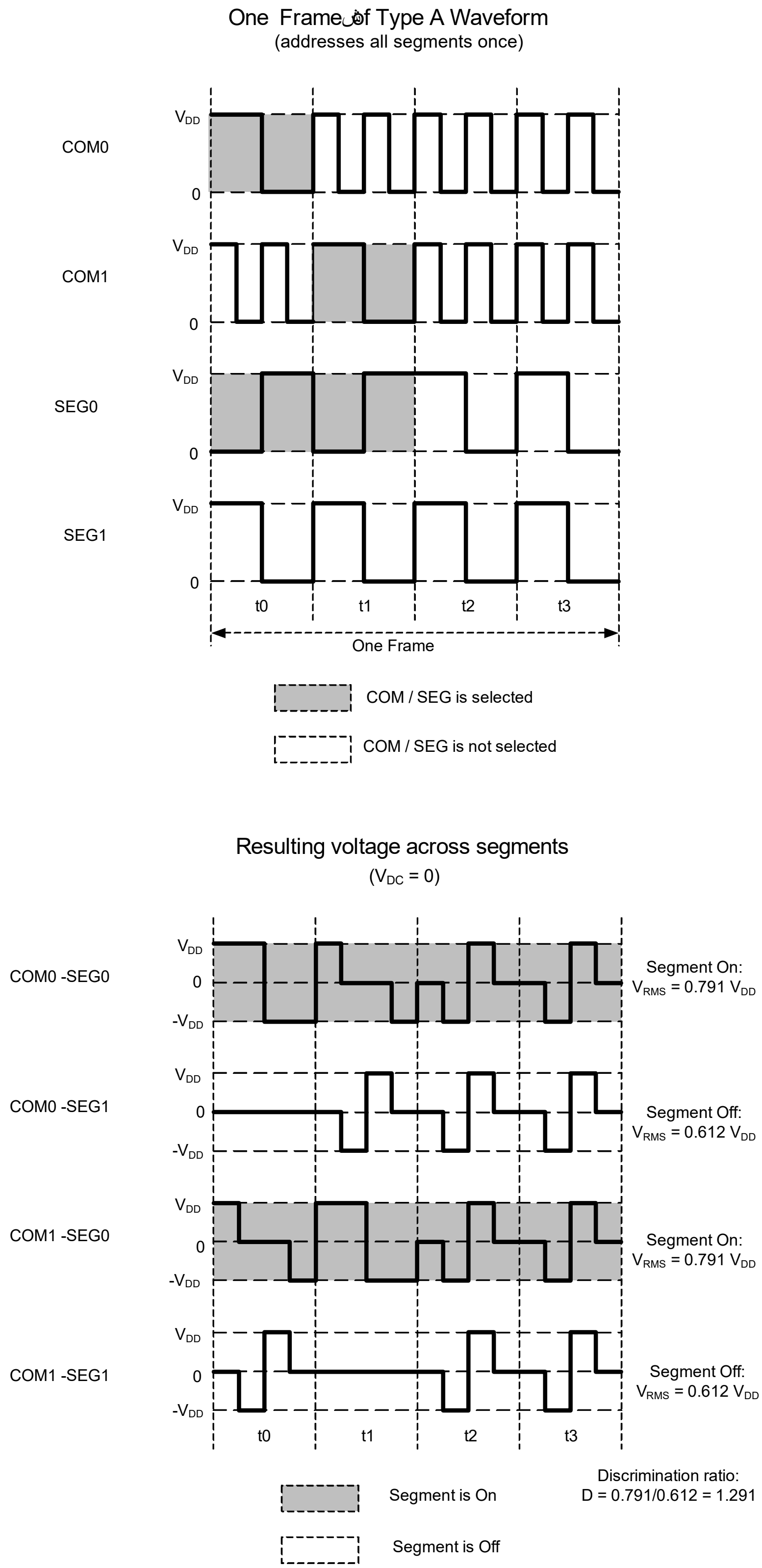
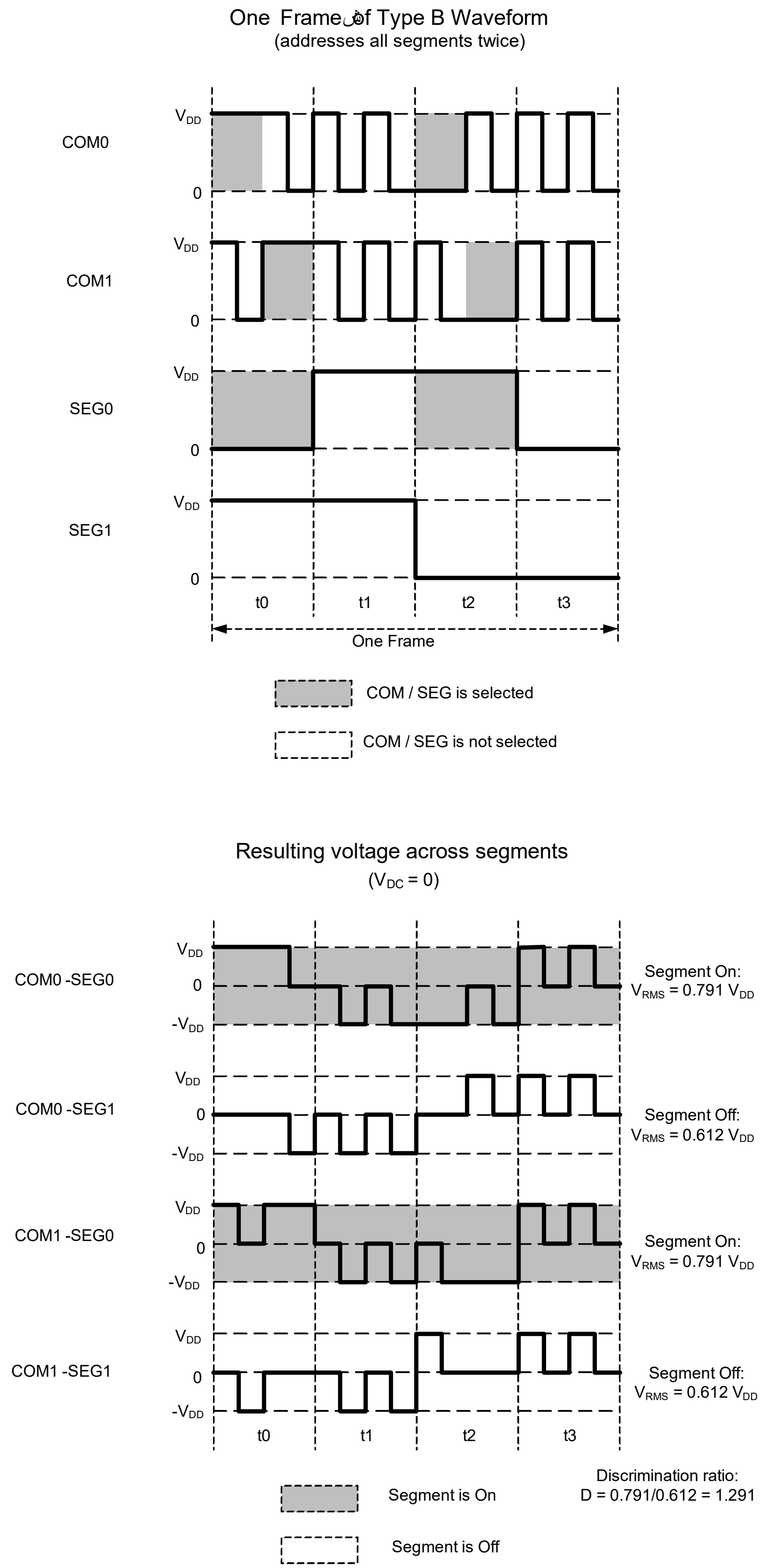


Figure 24-7. Digital Correlation Type-B Waveform



The RMS voltage applied to on and off segments can be calculated as follows:

$$V_{RMS(OFF)} = \sqrt{\frac{(M-1)}{2M}} x(V_{DD})$$

$$V_{RMS(ON)} = \sqrt{\frac{2+(M-1)}{2M}} x(V_{DD})$$

Where B is the bias and M is the duty (number of COMs). This leads to a discrimination ratio (D) of 1.291 for four COMs. Digital correlation mode also has the ability to drive 3-V displays from 1.8-V V_{DD}.

24.2.2 Recommended Usage of Drive Modes

The PWM drive mode has higher discrimination ratios compared to the digital correlation mode, as explained in 24.2.1.1 PWM Drive and 24.2.1.2 Digital Correlation. Therefore, the contrast in digital correlation method is lower than PWM method but digital correlation has lower power consumption because its waveforms toggle at low frequencies.

The digital correlation mode creates reduced, but acceptable contrast on TN displays, but no noticeable difference in contrast or viewing angle on higher contrast STN displays. Because each mode has strengths and weaknesses, recommended usage is as follows.

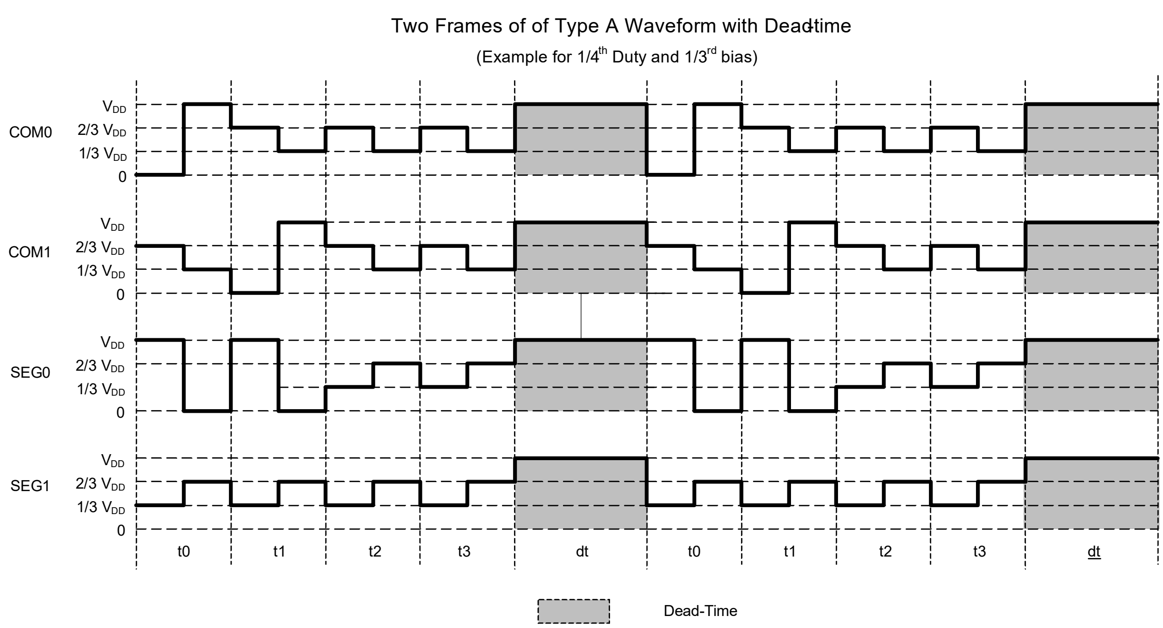
Table 24-1. Recommended Usage of Drive Modes

Display Type	Deep-Sleep Mode	Sleep/Active Mode	Notes
Four COM TN Glass	Digital correlation	PWM 1/3 bias	Firmware must switch between LCD drive modes before going to deep sleep or waking up.
Four COM STN Glass	Digital correlation		No contrast advantage for PWM drive with STN glass.
Eight COM, STN	Not supported	PWM 1/4 bias and 1/5 bias	Supported only in the high-speed LCD mode. The low-speed clock is not fast enough to make the PWM work at high multiplex ratios.

24.2.3 Digital Contrast Control

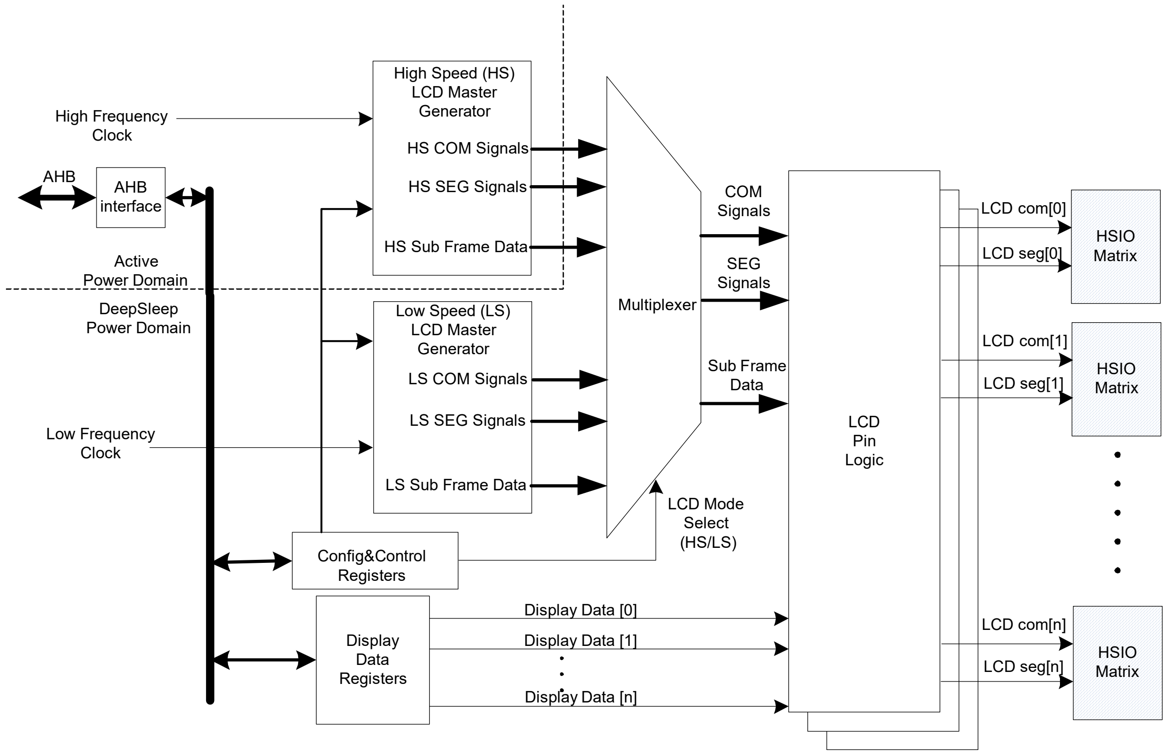
In all drive modes, digital contrast control can be used to change the contrast level of the segments. This method reduces contrast by reducing the driving time of the segments. This is done by inserting a ‘Dead-Time’ interval after each frame. During dead time, all COM and SEG signals are driven to a logic 1 state. The dead time can be controlled in fine resolution. Figure 24-8 illustrates the dead-time contrast control method for 1/3 bias and 1/4 duty implementation.

Figure 24-8. Dead-Time’ Contrast Control



24.3 Block Diagram

Figure 24-9. Block Diagram of LCD Direct Drive System



24.3.1 How it Works

The LCD controller block contains two generators; one with a high-speed clock source HFCLK and the other with a low-speed clock source derived from the ILO. These are called high-speed LCD master generator and low-speed LCD master generator, respectively. Both the generators support PWM and digital correlation drive modes. PWM drive mode with low-speed generator requires external resistors, as explained in PWM Drive on page 218.

The multiplexer selects one of these two generator outputs to drive LCD, as configured by the firmware. The LCD pin logic block routes the COM and SEG outputs from the generators to the corresponding I/O matrices. Any GPIO can be used as either COM or SEG. This configurable pin assignment for COM or SEG is implemented in GPIO and I/O matrix; see High-Speed I/O Matrix on page 62. These two generators share the same configuration registers. These memory mapped I/O registers are connected to the system bus (AHB) using an AHB interface.

The LCD controller works in three device power modes: active, sleep, and deep-sleep. High-speed operation is supported in active and sleep modes. Low-speed operation is supported in active, sleep, and deep-sleep modes. The LCD controller is unpowered in hibernate and stop modes.

24.3.2 High-Speed and Low-Speed Master Generators

The high-speed and low-speed master generators are similar to each other. The only exception is that the high-speed version has larger frequency dividers to generate the frame and sub-frame periods. This is because the clock of the high-speed block (HFCLK) is derived from the IMO, which is typically at 30 to 100 times the frequency of the ILO clock fed to the low-speed block. The high-speed generator is in the active power domain and the low-speed generator is in the deep-sleep power domain. A single set of configuration registers is provided to control both high-speed and low-speed blocks. Each master generator has the following features and characteristics:

- Register bit configuring the block for either Type A or Type B drive waveforms (LCD_MODE bit in LCD_CONTROL register).
- Register bits to select the number of COMs (COM_NUM field in LCD_CONTROL register). The available values are 2, 3, and 4.
- Operating mode configuration bits enabled to select one of the following:
  - Digital correlation
  - PWM 1/2 bias

- PWM 1/3 bias
- PWM 1/4 bias (not supported in low-speed generator)
- PWM 1/5 bias (not supported in low-speed generator)
- Off/disabled. Typically, one of the two generators will be configured to be Off

OP_MODE and BIAS fields in LCD_CONTROL bits select the drive mode.

- A counter to generate the sub-frame timing. The SUBFR_DIV field in the LCD_DIVIDER register determines the duration of each sub-frame. If the divide value written into this counter is C, the sub-frame period is 4 × (C+1). The low-speed generator has an 8-bit counter. This counter generates a maximum half sub-frame period of 8 ms from the ILO clock. The high-speed generator has a 16-bit counter.
- A counter to generate the dead time period. These counters have the same number of bits as the sub-frame period counters and use the same clocks. DEAD_DIV field in the LCD_DIVIDER register controls the dead time period.

24.3.3 Multiplexer and LCD Pin Logic

The multiplexer selects the output signals of either high-speed or low-speed master generator blocks and feeds it to the LCD pin logic. This selection is controlled by the configuration and control register. The LCD pin logic uses the sub-frame signal from the multiplexer to choose the display data. This pin logic will be replicated for each LCD pin.

24.3.4 Display Data Registers

Each LCD segment pin is part of an LCD port with its own display data register, LCD_DATAnx. The device has eight such LCD ports. Note that these ports are not real pin ports but the ports/connections available in the LCD hardware for mapping the segments to commons. Each LCD segment configured is considered as a pin in these LCD ports. The LCD_DATAnx registers are 32-bit wide and store the ON/OFF data for all SEG-COM combination enabled in the design. LCD_DATA0x holds SEG-COM data for COM0 to COM3 and LCD_DATA1x holds SEG-COM data for COM4 to COM7. The bits [4i+3:4i] (where 'i' is the pin number) of each LCD_DATA0x register represent the ON/OFF data for Pin[i] in Port[x] and COM[3,2,1,0] combinations, as shown in Table 24-2. The LCD_DATAnx register should be programmed according to the display data of each frame. The display data registers are Memory Mapped I/O (MMIO) and accessed through the AHB slave interface.

Table 24-2. SEG-COM Mapping in LCD_DATA0x Registers (each SEG is a pin of the LCD port)

BITS[31:28] = PIN_7[3:0]				BITS[27:24] = PIN_6[3:0]			
PIN_7-COM3	PIN_7-COM2	PIN_7-COM1	PIN_7-COM0	PIN_6-COM3	PIN_6-COM2	PIN_6-COM1	PIN_6-COM0
BITS[23:20] = PIN_5[3:0]				**BITS[19:16] = PIN_4[3:0]**			
PIN_5-COM3	PIN_5-COM2	PIN_5-COM1	PIN_5-COM0	PIN_4-COM3	PIN_4-COM2	PIN_4-COM1	PIN_4-COM0
BITS[15:12] = PIN_3[3:0]				**BITS[11:8] = PIN_2[3:0]**			
PIN_3-COM3	PIN_3-COM2	PIN_3-COM1	PIN_3-COM0	PIN_2-COM3	PIN_2-COM2	PIN_2-COM1	PIN_2-COM0
BITS[7:3] = PIN_1[3:0]				**BITS[3:0] = PIN_0[3:0]**			
PIN_1-COM3	PIN_1-COM2	PIN_1-COM1	PIN_1-COM0	PIN_0-COM3	PIN_0-COM2	PIN_0-COM1	PIN_0-COM0

24.4 Register List

Table 24-3. LCD Direct Drive Register List

Register Name	Description
LCD_ID	This register includes the information of LCD controller' ID and revision number
LCD_DIVIDER	This register controls the sub-frame and dead-time period
LCD_CONTROL	This register is used to configure high-speed and low-speed generators
LCD_DATA0x	LCD port pin data register for COM0 to COM3; x = port number, eight ports are available
LCD_DATA1x	LCD port pin data register for COM4 to COM7; x = port number, eight ports are available

25. CapSense

The PSoC Analog Coprocessor device uses the fourth generation capacitive touch sensing system (CapSense). The CapSense system can measure the change in self-capacitance of a single electrode or the mutual capacitance between a pair of electrodes. In addition to the capacitive sensing feature, the system can also function as an ADC to measure voltage on any GPIO pin that supports CapSense functionality.

PSoC Analog Coprocessor uses Cypress's patented capacitive touch sensing methods – CapSense Sigma Delta (CSD) for self-capacitance sensing and CapSense Crosspoint (CSX) for mutual-capacitance scanning. The CSD and CSX touch sensing methods provide the industry's best-in-class signal-to-noise ratio (SNR), high touch sensitivity, low-power operation, and superior EMI performance. The CapSense system is a combination of hardware and firmware techniques.

See the *Getting Started with CapSense* design guide for basics of capacitive sensing and the *PSoC 4 CapSense Design Guide* for more details on the basics of CapSense operation.

25.1 Features

The fourth-generation CapSense in the PSoC 4 device has the following features:

- Supports self-capacitance and mutual-capacitance-based touch sensing
- Supports voltage measurement on any GPIO pin
- Provides superior SNR with CSD-based touch sensing method and with programmable voltage reference
- Provides robust touch sensing using spread spectrum scanning method
- Supports spread spectrum, pseudo-random sequence (PRS) clock source and programmable resistance switches to reduce electromagnetic interference (EMI)
- Provides high touch sensitivity to detect touch across a variety of overlay materials and thickness
- Provides low-power CapSense operation
- Allows any GPIO pin to be used for capacitive sensing and shielding
- Supports large proximity-sensing distance
- Supports liquid tolerant operation using driven shield signal
- Supports split-IDAC operation for improved scan speed and SNR
- Reduces overhead on CPU during CapSense scanning by offloading the initialization process to the CapSense sequencer
- PSoC Creator CapSense Component supports SmartSense™ auto-tuning to automatically tune all the CapSense parameters
- Allows general-purpose use of CapSense comparator and IDAC

25.2 Block Diagram

Figure 25-1 shows the block diagram of the CapSense system.

Figure 25-1. CapSense System Block Diagram

25.3 How It Works

The following are the main blocks of the CapSense system:

- GPIO Cell
- Analog Mux Bus (AMUXBUS)
- CapSense Delta Sigma (CSD) Modulator
- IDAC
- Digital Sequencers and Counter
- Clock Generation

25.3.1 GPIO Cell

In a CapSense system, the GPIO cells are used for the following purposes:

- Sensing self-capacitance
- Driving and sensing mutual-capacitance
- Driving shield signal
- Connecting external capacitors to the CapSense system

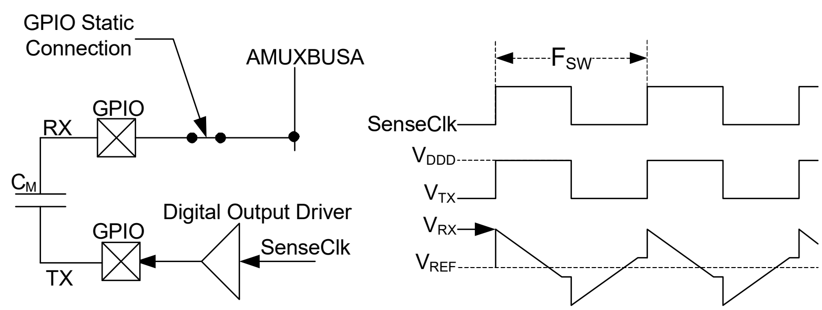
25.3.1.1 Sensing Self-Capacitance

25.3.1.2 Driving and Sensing Mutual-capacitance

Mutual-capacitance sensing measures the mutual-capacitance between two electrodes connected to the GPIO pins. One of the GPIOs is used as a transmit (TX) pin and the other as a receive (RX) pin.

Figure 25-2 shows the configuration of GPIO in mutual-capacitance sensing mode. The TX pin drives a digital signal to the TX electrode using the digital output driver of the GPIO pin. The SenseClk generated from the clock generator section is the input clock to the digital output driver. The RX pin is statically connected to the AMUXBUSA to measure the charge coupled from TX electrode to the RX electrode.

Figure 25-2. GPIO Configuration for Mutual-Capacitance Sensing



When the SenseClk is high, charge is coupled from TX electrode to RX electrode. The voltage on the RX electrode, which was at V_{REF} increases above V_{REF} due to charge coupling from TX electrode to RX electrode. A sinking IDAC is connected to AMUXBUSA to bring the voltage on AMUXBUSA back to V_{REF}.

Similarly, when the SenseClk is low, charge flows from RX electrode to TX electrode and the voltage on RX electrode, which was at V_{REF} drops below V_{REF}. A sourcing IDAC is connected to AMUXBUSA to bring the voltage back to V_{REF}.

25.3.1.3 Driving Shield Signal

Shield signal is used to switch the shield electrode between V_{REF} and ground to reduce sensor C_P, implement liquid tolerance, and improve the proximity-sensing distance

25.3.1.4 Connecting External Capacitors

The CapSense system requires external capacitors (C_{MOD}, C_{INTA}, and C_{INTB}) to be connected to dedicated pins for proper CapSense operation.

25.3.2 Analog Mux Bus

The Analog Mux Bus (AMUXBUS) in the device provides the path for the signal from GPIO pin to the CapSense block. The PSoC 4 device has two analog mux buses, AMUXBUSA and AMUXBUSB. These mux buses allow any GPIO to be used as sensor pin or as a shield pin.

The AMUXBUSA is used to connect the sensor pin to the CapSense block. The AMUXBUSB is used to drive shield signal from the CapSense block to the shield pin.

25.3.3 CapSense Delta Sigma Modulator

25.3.4 Digital Sequencer

To reduce CPU overhead during the CapSense and ADC operations, the CapSense system has two sequencers - CSD sequencer and ADC sequencer, which control CapSense hardware during initialization and capacitance or input voltage measurements.

The CapSense system implements a slope ADC to measure the input voltage. The ADC sequencer automatically samples the input voltage and performs the voltage measurement without CPU intervention.

25.3.5 Clock Generation

The clock generation block generates the clock signal required for the switches in the GPIO and the clock for the CapSense and ADC counter.

Figure 25-3 shows the CapSense clock architecture. The SampleClk is generated using the PERI divider. The SampleClk is used as the input clock for CSD and ADC counters.

The SenseClk is used to control the switches in the GPIO. It can be generated using the following methods:

- Direct Clock
- Direct Clock with PWM
- Spread Spectrum Clock

Figure 25-3. Clock Generation for CapSense

25.3.5.1 Direct Clock

In the direct clock configuration, the SenseClk is a divided version of SampleClk.

25.3.5.2 Direct Clock with PWM

In this configuration, the duty cycle of the SenseClk can be varied

25.3.5.3 Spread Spectrum Clock

The SenseClk can also be generated as a pseudo-random sequence signal.

25.4 General-Purpose Resources

If the CapSense block is not used for touch sensing, the CSDCOMP and the two IDACs can be used as general-purpose analog blocks.

You can use AMUXBUSA/B to connect any CapSense-supported GPIO to the positive input of the CSDCOMP. The negative input of the CSDCOMP is connected to the REFGEN. The AMUXBUSA/B can also be used as an analog multiplexer at the comparator input.

If AMUXBUS is required for other uses, the positive terminal of the CSDCOMP to the fixed CEXT1, CEXT2, and CEXT3 pins. The output of the comparator is connected to a dedicated pin CSD.COMP. See thet device datasheet for details on the pin number.

Both the IDAC-A/B can operate in the general-purpose mode.

The output of IDAC can be connected to GPIOs using either AMUXBUSA or AMUXBUSB. It is also possible to connect the both IDACs to a single AMUXBUS.

Section F: Program and Debug

This section encompasses the following chapters:

Top Level Architecture

Program and Debug Block Diagram

26. Program and Debug Interface

The PSoC® Analog Coprocessor Program and Debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be a Cypress-supplied programmer and debugger, or a third-party device that supports programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and the PSoC Analog Coprocessor.

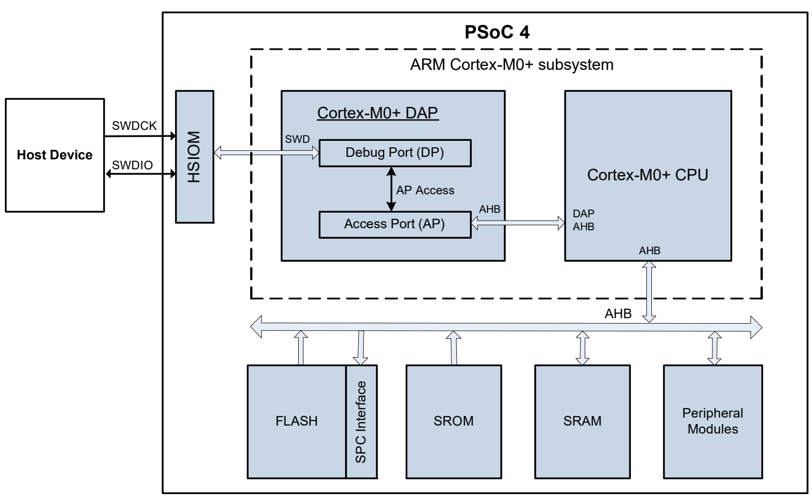
26.1 Features

- Programming and debugging through the SWD interface
- Four hardware breakpoints and two hardware watchpoints while debugging
- Read and write access to all memory and registers in the system while debugging, including the Cortex-M0+ register bank when the core is running or halted

26.2 Functional Description

Figure 26-1 shows the block diagram of the program and debug interface in the PSoC Analog Coprocessor. The Cortex-M0+ debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the "host", communicates with the DAP of the PSoC Analog Coprocessor "target" using the two pins of the SWD interface - the bidirectional data pin (SWDIO) and the host-driven clock pin (SWDCK). The SWD physical port pins (SWDIO and SWDCK) communicate with the DAP through the high-speed I/O matrix (HSIOM). See the I/O System chapter on page 57 for details on HSIOM.

Figure 26-1. Program and Debug Interface



The DAP communicates with the Cortex-M0+ CPU using the ARM-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside the device, which facilitates memory and peripheral register access by the AHB master. The device has two AHB masters – ARM CM0 CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

26.3 Serial Wire Debug (SWD) Interface

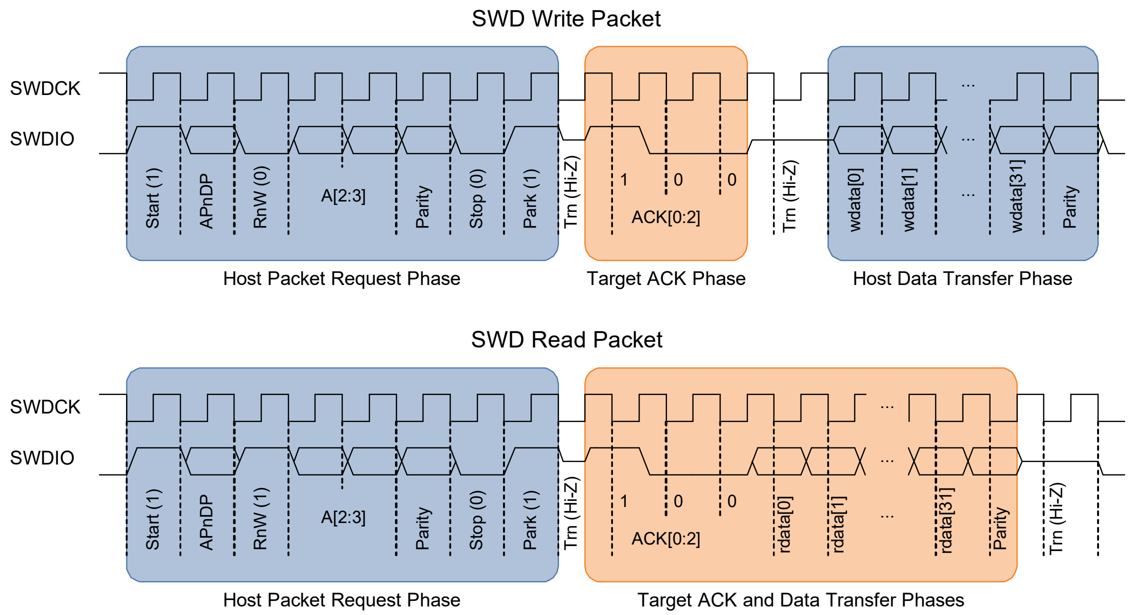
PSoC Analog Coprocessor's Cortex-M0+ supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete data transfer (one SWD packet) requires 46 clocks and consists of three phases:

- **Host Packet Request Phase** – The host issues a request to the PSoC Analog Coprocessor target.
- **Target Acknowledge Response Phase** – The PSoC Analog Coprocessor target sends an acknowledgement to the host.
- **Data Transfer Phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

Figure 26-2 shows the timing diagrams of read and write SWD packets.

Figure 26-2. SWD Write and Read Packet Timing Diagrams



The sequence to transmit SWD read and write packets are as follows:

1. Host Packet Request Phase: SWDIO driven by the host
   a. The start bit initiates a transfer; it is always logic 1.
   b. The "AP not DP" (APnDP) bit determines whether the transfer is an AP access – 1b1 or a DP access – 1b0.
   c. The "Read not Write" bit (RnW) controls which direction the data transfer is in. 1b1 represents a 'read from' the target, or 1b0 for a 'write to' the target.
   d. The Address bits (A[3:2]) are register select bits for AP or DP, depending on the APnDP bit value. See Table 26-3 and Table 26-4 for definitions.
   Note Address bits are transmitted with the LSB first.
   e. The parity bit contains the parity of APnDP, RnW, and ADDR bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.
   If the parity bit is not correct, the header is ignored by the PSoC Analog Coprocessor; there is no ACK response (ACK = 3b111). The programming operation should be aborted and retried again by following a device reset.
   f. The stop bit is always logic 0.
   g. The park bit is always logic 1.
2. Target Acknowledge Response Phase: SWDIO driven by the target
   a. The ACK[2:0] bits represent the target to host response, indicating failure or success, among other results. See Table 26-1 for definitions.
   Note ACK bits are transmitted with the LSB first.
3. Data Transfer Phase: SWDIO driven by either target or host depending on direction
   a. The data for read or write is written to the bus, LSB first.

b. The data parity bit indicates the parity of the data read or written. It is an even parity; this means when XORed with the data bits, the result will be 0.

   If the parity bit indicates a data error, corrective action should be taken. For a read packet, if the host detects a parity error, it must abort the programming operation and restart. For a write packet, if the target detects a parity error, it generates a FAULT ACK response in the next packet.

According to the SWD protocol, the host can generate any number of SWDCK clock cycles between two packets with SWDIO low. It is recommended to generate three or more dummy clock cycles between two SWD packets if the clock is not free-running or to make the clock free-running in IDLE mode.

The SWD interface can be reset by clocking the SWDCK line for 50 or more cycles with SWDIO high. To return to the idle state, clock the SWDIO low once.

26.3.1 SWD Timing Details

The SWDIO line is written to and read at different times depending on the direction of communication. The host drives the SWDIO line during the Host Packet Request Phase and, if the host is writing data to the target, during the Data Transfer phase as well. When the host is driving the SWDIO line, each new bit is written by the host on falling SWDCK edges, and read by the target on rising SWDCK edges. The target drives the SWDIO line during the Target Acknowledge Response Phase and, if the target is reading out data, during the Data Transfer Phase as well. When the target is driving the SWDIO line, each new bit is written by the target on rising SWDCK edges, and read by the host on falling SWDCK edges.

Table 26-1 and Figure 26-2 illustrate the timing of SWDIO bit writes and reads.

Table 26-1. SWDIO Bit Write and Read Timing

SWD Packet Phase	SWDIO Edge	
	Falling	Rising
Host Packet Request	Host Write	Target Read
Host Data Transfer	Host Write	Target Read
Target Ack Response	Host Read	Target Write
Target Data Transfer	Host Read	Target Write

26.3.2 ACK Details

The acknowledge (ACK) bit-field is used to communicate the status of the previous transfer. OK ACK means that previous packet was successful. A WAIT response requires a data phase. For a FAULT status, the programming operation should be aborted immediately. Table 26-2 shows the ACK bit-field decoding details.

Table 26-2. SWD Transfer ACK Response Decoding

Response	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	3b111

Details on WAIT and FAULT response behaviors are as follows:

- For a WAIT response, if the transaction is a read, the host should ignore the data read in the data phase. The target does not drive the line and the host must not check the parity bit as well.
- For a WAIT response, if the transaction is a write, the data phase is ignored by the PSoC Analog Coprocessor. But, the host must still send the data to be written to complete the packet. The parity bit corresponding to the data should also be sent by the host.
- For a WAIT response, it means that the PSoC Analog Coprocessor is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received. If it fails, then the programming operation should be aborted and retried again.
- For a FAULT response, the programming operation should be aborted and retried again by doing a device reset.

26.3.3 Turnaround (Trn) Period Details

There is a turnaround period between the packet request and the ACK phases, as well as between the ACK and the data phases for host write transfers, as shown in Figure 26-2. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This action ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor the PSoC Analog Coprocessor should drive the SWDIO line during the Trn period.

26.4 Cortex-M0+ Debug and Access Port (DAP)

The Cortex-M0+ program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex-M0+ DAP-AHB interface. Typically, the DP registers are used for a one time configuration or for error detection purposes, and the AP registers are used to perform the programming and debugging operations. Complete architecture details of the DAP is available in the *ARM® Debug Interface v5 Architecture Specification*.

26.4.1 Debug Port (DP) Registers

Table 26-3 shows the Cortex-M0+ DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the *ARM® Debug Interface v5 Architecture Specification* for details on all of the DP registers.

Table 26-3. Main Debug Port (DP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex-M0+ CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In PSoC Analog Coprocessor, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

26.4.2 Access Port (AP) Registers

Table 26-4 lists the main Cortex-M0+ AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 26-4. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the PSoC Analog Coprocessor Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register

26.5 Programming the PSoC Analog Coprocessor Device

PSoC 4 is programmed using the following sequence. Refer to the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4700S Device Programming Specifications for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in the PSoC Analog Coprocessor.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

26.5.1 SWD Port Acquisition

26.5.1.1 Primary and Secondary SWD Pin Pairs

The first step in device programming is to acquire the SWD port in the PSoC Analog Coprocessor. Refer to the device datasheet for information on SWD pins.

If two SWD pin pairs are available in the device, the SWD_CONFIG register in the supervisory flash region is used to select between one of the two SWD pin pairs that can be used for programming and debugging. Note that only one of the SWD pin pairs can be used during any programming or debugging session. The default selection for devices coming from the factory is the primary SWD pin pair. To select the secondary SWD pin pair, it is necessary to program the device using the primary pair with the hex file that enables the secondary pin pair configuration. Afterwards, the secondary SWD pin pair may be used.

26.5.1.2 SWD Port Acquire Sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After removing the XRES signal, the host must send an SWD connect sequence for the device within the acquire window to connect to the SWD interface in the DAP. The pseudo code for the sequence is given here.

Code 1. SWD Port Acquire Pseudo Code

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute ARM's connection sequence to acquire SWD-port
do
{
	SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
	ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register
}while ((ack != OK) && time_elapsed < ms); // retry connection until OK ACK or timeout

if (time_elapsed >= ms) return FAIL; //check for acquire time out

if (ID != CM0P_ID) return FAIL; //confirm SWD ID of Cortex-M0+ CPU. (0x0BC11477)
```

In this pseudo code, SWD_LineReset() is the standard ARM command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex-M0+ DAP.

26.5.2 SWD Programming Mode Entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudo code for entering the programming mode are detailed in the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4700S Device Programming Specifications document. The minimum required clock frequency for the Port Acquire step and this step to succeed is 1.5 MHz.

26.5.3 SWD Programming Routines Executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in the Nonvolatile Memory Programming chapter on page 241. The exact sequence of calling the programming routines is given in thePSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4700S Device Programming Specifications.

26.6 PSoC Analog Coprocessor SWD Debug Interface

Cortex-M0+ DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex-M0+ Debug.

See the *ARMv6-M Architecture Reference Manual* for complete details on the debug architecture.

26.6.1 Debug Control and Configuration Registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the *ARMv6-M Architecture Reference Manual* for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0P_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0P_DFSR) – This register describes the reason a debug event has occurred and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0P_DCRSR) – This register is used to select the general-purpose register in the Cortex-M0+ CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0P_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0P_DCRSR register.
- Debug Exception and Monitor Control Register (CM0P_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

26.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0+ DAP in the PSoC Analog Coprocessor supports up to four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction in the Cortex-M0+. The BPU has two types of registers.

- The breakpoint control register (CM0P_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in the PSoC Analog Coprocessor).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0P_BP_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

26.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows.

- The watchpoint compare (CM0P_DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint mask (CM0P_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0P_DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event has occurred.
- The watchpoint comparator PC sample register (CM0P_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, non-invasive profiling of the program counter register.

26.6.4 Debugging the PSoC Analog Coprocessor Device

The host debugs the target PSoC Analog Coprocessor by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0+ DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PSoC Analog Coprocessor is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the

DAP SWDID through the SWD interface. The SWD port is acquired when the correct CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the I/O System chapter on page 57 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as LCD and GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

Debugging the target device is also affected by the overall device protection setting, which is explained in the Device Security chapter on page 99. Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

26.7 Registers

Table 26-5. List of Registers

Register Name	Description
CM0P_DHCSR	Debug Halting Control and Status Register
CM0P_DFSR	Debug Fault Status Register
CM0P_DCRSR	Debug Core Register Selector Register
CM0P_DCRDR	Debug Core Register Data Register
CM0P_DEMCR	Debug Exception and Monitor Control Register
CM0P_BP_CTRL	Breakpoint control register
CM0P_BP_COMPx	Breakpoint Compare Register
CM0P_DWT_COMPx	Watchpoint Compare Register
CM0P_DWT_MASKx	Watchpoint Mask Register
CM0P_DWT_FUNCTIONx	Watchpoint Function Register
CM0P_DWT_PCSR	Watchpoint Comparator PC Sample Register

27. Nonvolatile Memory Programming

Nonvolatile memory programming refers to the programming of flash memory in the PSoC® Analog Coprocessor device. This chapter explains the different functions that are part of device programming, such as erase, write, program, and checksum calculation. Cypress-supplied programmers and other third-party programmers can use these functions to program the PSoC device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

27.1 Features

- Supports programming through the debug and access port (DAP) and Cortex-M0+ CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex-M0+ CPU

27.2 Functional Description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0+ CPU requests the system call by writing the function opcode and parameters to the System Performance Controller Interface (SPCIF) input registers, and then requesting the SROM to execute the function. Based on the function opcode, the System Performance Controller (SPC) executes the corresponding system call from SROM and updates the SPCIF status register. The DAP or the CPU should read this status register for the pass/fail result of the function execution. As part of function execution, the code in SROM interacts with the SPCIF to do the actual flash programming operations.

PSoC flash is programmed using a Program Erase Program (PEP) sequence. The flash cells are all programmed to a known state, erased, and then the selected bits are programmed. This sequence increases the life of the flash by balancing the stored charge. When writing to flash the data is first copied to a page latch buffer. The flash write functions are then used to transfer this data to flash.

External programmers program the flash memory in PSoC using the SWD protocol by sending the commands to the Debug and Access Port (DAP). The programming sequence for the PSoC device with an external programmer is given in the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4100S, PSoC 4700S Programming Specifications. Flash memory can also be programmed by the CM0+ CPU by accessing the relevant registers through the AHB interface. This type of programming is typically used to update a portion of the flash memory as part of a bootload operation, or other application requirements, such as updating a lookup table stored in the flash memory. All write operations to flash memory, whether from the DAP or from the CPU, are done through the SPCIF.

Note It can take as much as 20 milliseconds to write to flash. During this time, the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources (see the Reset System chapter on page 97) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

Note PSoC implements a User Supervisory Flash (SFlash), which can be used to store application-specific information. These rows are not part of the hex file; their programming is optional.

27.3 System Call Implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: Two 8-bit parameters are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows:

  key1 = 0xB6

  key2 = 0xD3 + Opcode

  The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, additional parameters may be required depending on the specific function being called.
- Return Values: Some system calls also return a value on completion of their execution, such as the silicon ID or a checksum.
- Completion Status: Each system call returns a 32-bit status that the CPU or DAP can read to verify success or determine the reason for failure.

27.4 Blocking and Non-Blocking System Calls

System call functions can be categorized as blocking or non-blocking based on the nature of their execution. Blocking system calls are those where the CPU cannot execute any other task in parallel other than the execution of the system call. When a blocking system call is called from a process, the CPU jumps to the code corresponding in SROM. When the execution is complete, the original thread execution resumes. Non-blocking system calls allow the CPU to execute some other code in parallel and communicate the completion of interim system call tasks to the CPU through an interrupt.

Non-blocking system calls are only used when the CPU initiates the system call. The DAP will only use system calls during the programming mode and the CPU is halted during this process.

The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

Because the CPU cannot execute code from flash while doing an erase or program operation on the flash, the non-blocking system calls can only be called from a code executing out of SRAM. If the non-blocking functions are called from flash memory, the result is undefined and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

The System Performance Controller (SPC) is the block that generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called from SRAM, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure that the subsequent steps in the system call are completed. Because the CPU can execute code only from the SRAM when a non-blocking write or program operation is being done, the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. The Resume Non-Blocking function call done in the SPC ISR is called once in a non-blocking program operation and thrice in a non-blocking write operation.

The pseudo code for using a non-blocking write system call and executing user code out of SRAM is given later in this chapter.

27.4.1 Performing a System Call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, additional parameters) are:
   a. Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b. Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The

maximum execution time is one second. If these two bits are not cleared after one second, the operation should be considered a failure and aborted without executing the following steps. Note that unlike the DAP, the CPU application code cannot poll these bits during system call execution. This is because the CPU executes code out of the SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.

4. Check the completion status: After the PRIVILEGED and SYSCALL_REQ bits are cleared to indicate completion of the system call, the CPUSS_SYSARG register should be read to check for the status of the system call. If the 32-bit value read from the CPUSS_SYSARG register is 0xAXXXXXXX (where 'X' denotes don't care hex values), the system call was successfully executed. For a failed system call, the status code is 0xF00000YY where YY indicates the reason for failure. See Table 27-1 for the complete list of status codes and their description.
5. Retrieve the return values: For system calls that return values such as silicon ID and checksum, the CPU or DAP should read the CPUSS_SYSREG and CPUSS_SYSARG registers to fetch the values returned.

27.5 System Calls

Table 27-1 lists all the system calls supported in PSoC along with the function description and availability in device protection modes. See the Device Security chapter on page 99 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

Table 27-1. List of System Calls

System Call	Description	DAP Access			CPU Access
		Open	Protected	Kill	
Silicon ID	Returns the device Silicon ID, Family ID, and Revision ID	✔	✔	–	✔
Load Flash Bytes	Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 128 bytes	✔	–	–	✔
Write Row	Erases and then programs a row of flash with data in the page latch buffer	✔	–	–	✔
Program Row	Programs a row of flash with data in the page latch buffer	✔	–	–	✔
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	✔	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔	–	✔
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	✔	✔	–	
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	✔

27.5.1 Silicon ID

This function returns a 12-bit family ID, 16-bit silicon ID, and an 8-bit revision ID, and the current device protection mode. These values are returned to the CPUSS_SYSARG and CPUSS_SYSREQ registers. Parameters are passed through the CPUSS_SYSARG and CPUSS_SYSREQ registers.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD3	Key2

Address	Value to be Written	Description
Bits [31:16]	0x0000	Not used
CPUSS_SYSREQ register		
Bits [15:0]	0x0000	Silicon ID opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [7:0]	Silicon ID Lo	See the device datasheet for Silicon ID values for different part numbers
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision Id	See the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4000S, PSoC 4100S, PSoC 4700S Programming Specifications for these values
Bits [23:20]	Major Revision Id	
Bits [27:24]	0xXX	Not used (don't care)
Bits [31:28]	0xA	Success status code
CPUSS_SYSREQ register		
Bits [11:0]	Family ID	Family ID is 0xAC for PSoC Analog Coprocessor
Bits [15:12]	Chip Protection	See the Device Security chapter on page 99
Bits [31:16]	0xXXXX	Not used

27.5.2 Configure Clock

This function initializes the clock necessary for flash programming and erasing operations. This API is used to ensure that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz prior to calling the flash write and flash erase APIs. The flash write and erase APIs will exit without acting on the flash and return the "Invalid Pump Clock Frequency" status if the IMO is the source of the charge pump clock and is not 48 MHz.

27.5.3 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 128 bytes. Data is loaded into the page latch buffer starting at the location specified by the "Byte Addr" input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS_SYSARG register. Note that the starting parameter address should be a word-aligned address.

Parameters

Address	Value to be Written	Description
SRAM Address - 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD7	Key2
Bits [23:16]	Byte Addr	Start address of page latch buffer to write data 0x00 – Byte 0 of latch buffer 0x7F – Byte 127 of latch buffer
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 (Refer to the Cortex-M0+ CPU chapter on page 28 for the number of flash macros in the device)

Address	Value to be Written	Description
SRAM Address- 32'hYY + 0x04		
Bits [7:0]	Load Size	Number of bytes to be written to the page latch buffer. 0x00 – 1 byte 0x7F – 128 bytes
Bits [15:8]	0xXX	Don't care parameter
Bits [23:16]	0xXX	Don't care parameter
Bits [31:24]	0xXX	Don't care parameter
SRAM Address- From (32'hYY + 0x08) to (32'hYY + 0x08 + Load Size)		
Byte 0	Data Byte [0]	First data byte to be loaded
.	.	.
.	.	.
Byte (Load size –1)	Data Byte [Load size –1]	Last data byte to be loaded
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0004	Load Flash Bytes opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

27.5.4 Write Row

This function erases and then programs the addressed row of flash with the data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The parameters for this function are stored in SRAM. The start address of the stored parameters is written to the CPUSS_SYSARG register. This function clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write protected.

Refer to the CLK_IMO_CONFIG register in the *PSoC Analog Coprocessor Family Registers TRM* for more information.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		

Address	Value to be Written	Description
Bits [31:0]	32’hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0005	Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don’t care)

27.5.5 Program Row

This function programs the addressed row of the flash with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The row must be in an erased state before calling this function. It clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

Parameters

Address	Value to be Written	Description
SRAM Address: 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD9	Key2
Bits [31:16]	Row ID	Row number to program 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32’hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0006	Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don’t care)

27.5.6 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in supervisory flash row 0 of each flash macro.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. This API can be called only from the DAP in the programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDD	Key2
Bits [31:16]	0xXXXX	Don't care
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x000A	Erase All opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

27.5.7 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDE	Key2
Bits [31:16]	Row ID	Selects the flash row number on which the checksum operation is done Row number – 16 bit flash row number or 0x8000 – Checksum is performed on entire flash memory
CPUSS_SYSREQ register		
Bits [15:0]	0x000B	Checksum opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	Checksum	24-bit checksum value of the selected flash region

27.5.8 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by eight. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

Then, the Write Protection function is called, which programs the flash protection bytes from the page latch to be the corresponding flash macro's supervisory row. In flash macro zero, which also stores the device protection settings, the device level protection setting is passed as a parameter in the CPUSS_SYSARG register.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE0	Key2
Bits [23:16]	Device Protection Byte	Parameter applicable only for Flash Macro 0 0x01 – OPEN mode 0x02 – PROTECTED mode 0x04 – KILL mode
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1
CPUSS_SYSREQ register		
Bits [15:0]	0x000D	Write Protection opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	0x000000	

27.5.9 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0+ CPU in a non-blocking manner, so that the CPU can execute code from SRAM while the write operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 242.

The non-blocking write row system call has three phases: Pre-program, Erase, Program. Pre-program is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all of the bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code from SRAM. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After the completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

Note The device firmware must not attempt to put the device to sleep during a non-blocking write row. This action will reset the page latch buffer and the flash will be written with all zeroes.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0+ CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDA	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0007	Non-Blocking Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

27.5.10 Non-Blocking Program Row

This function is used when a flash row needs to be programmed by the CM0+ CPU in a non-blocking manner, so that the CPU can execute code from the SRAM when the program operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 242. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

Unlike the Non-Blocking Write Row system call, the Program system call only has a single phase. Therefore, the Resume Non-Blocking function only needs to be called once from the SPC interrupt when using the Non-Blocking Program Row system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0+ CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDB	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

27.5.11 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are explained in Blocking and Non-Blocking System Calls on page 242.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDC	Key2
Bits [31:16]	0xXXXX	Don't care. Not used by SROM
CPUSS_SYSARG register		

Address	Value to be Written	Description
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0009	Resume Non-Blocking opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

27.6 System Call Status

At the end of every system call, a status code is written over the arguments in the CPUSS_SYSARG register. A success status is 0xAXXXXXXX, where X indicates don't care values or return data in the case of the system calls that return a value. A failure status is indicated by 0xF00000XX, where XX is the failure code.

Table 27-2. System Call Status Codes

Status Code (32-bit value in CPUSS_SYSARG register)	Description
AXXXXXXXh	Success – The "X" denotes a don't care value, which has a value of '0' returned by the SROM, unless the API returns parameters directly to the CPUSS_SYSARG register.
F0000001h	Invalid Chip Protection Mode – This API is not available during the current chip protection mode.
F0000003h	Invalid Page Latch Address – The address within the page latch buffer is either out of bounds or the size provided is too large for the page address.
F0000004h	Invalid Address – The row ID or byte address provided is outside of the available memory.
F0000005h	Row Protected – The row ID provided is a protected row.
F0000007h	Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API.
F0000008h	Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other APIs may be called.
F0000009h	System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume.
F000000Ah	Checksum Zero Failed – The calculated checksum was not zero.
F000000Bh	Invalid Opcode – The opcode is not a valid API opcode.
F000000Ch	Key Opcode Mismatch – The opcode provided does not match key1 and key2.
F000000Eh	Invalid Start Address – The start address is greater than the end address provided.
F0000012h	Invalid Pump Clock Frequency - IMO must be set to 48 MHz and HF clock source to the IMO clock source before flash write/erase operations.

27.7 Non-Blocking System Call Pseudo Code

This section contains pseudo code to demonstrate how to set up a non-blocking system call and execute code out of SRAM during the flash programming operations.

```
#define REG(addr)          (*((volatile uint32 *) (addr)))
#define CM0_ISER_REG              REG( 0xE000E100 )
#define CPUSS_CONFIG_REG              REG( 0x40100000 )
#define CPUSS_SYSREQ_REG              REG( 0x40100004 )
#define CPUSS_SYSARG_REG              REG( 0x40100008 )

#define ROW_SIZE_          ()
#define ROW_SIZE        (ROW_SIZE_)

/*Variable to keep track of how many times SPC ISR is triggered */
__ram int iStatusInt = 0x00;

__flash int main(void)
{
  DoUserStuff();

  /*CM0+ interrupt enable bit for spc interrupt enable */
  CM0_ISER_REG |= 0x00000040;

  /*Set CPUSS_CONFIG.VECS_IN_RAM because SPC ISR should be in SRAM */
  CPUSS_CONFIG_REG |= 0x00000001;

  /*Call non-blocking write row API */
  NonBlockingWriteRow();

  /*End Program */
  while(1);
}
__sram void SpcIntHandler(void)
{
  /* Write key1, key2 parameters to SRAM */
  REG( 0x20000000 ) = 0x0000DCB6;

  /*Write the address of key1 to the CPUSS_SYSARG reg */
  CPUSS_SYSARG_REG = 0x20000000;

  /*Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
  * register and assert the sysreq bit
  */
  CPUSS_SYSREQ_REG = 0x80000009;

  /* Number of times the ISR has triggered */
  iStatusInt ++;
}
__sram void NonBlockingWriteRow(void)
{
  int iter;

  /*Load the Flash page latch with data to write*/
  * Write key1, key2, byte address, and macro sel parameters to SRAM
  */
  REG( 0x20000000 ) = 0x0000D7B6;
```

```
    //Write load size param (128 bytes) to SRAM
    REG( 0x20000004 ) = 0x0000007F;


    for(i = 0; i < ROW_SIZE/4; i += 1)
    {
        REG( 0x20000008 + i*4 ) = 0xDADADADA;
    }

    /*Write the address of the key1 param to CPUSS_SYSARG reg*/
    CPUSS_SYSARG_REG = 0x20000000;

    /*Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
    * register and assert the sysreq bit
    */
    CPUSS_SYSREQ_REG = 0x80000004;

    /*Perform Non-Blocking Write Row on Row 200 as an example.
    * Write key1, key2, row id to SRAM row id = 0xC8 -> which is row 200
    */
    REG( 0x20000000 ) = 0x00C8DAB6;

    /*Write the address of the key1 param to CPUSS_SYSARG reg */
    CPUSS_SYSARG_REG = 0x20000000;

    /*Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
    * register and assert the sysreq bit
    */
    CPUSS_SYSREQ_REG = 0x80000007;

    /*Execute user code until iStatusInt equals 3 to signify
    * 3 SPC interrupts have happened. This should be 1 in case
    * of non-blocking program System Call
    */
    while( iStatusInt != 0x03 )
    {
        DoOtherUserStuff();
    }

    /* Get the success or failure status of System Call*/
    syscall_status = CPUSS_SYSARG_REG;
}
```

In the code, the CM0+ exception table is configured to be in SRAM by writing 0x01 to the CPUSS_CONFIG register. The SRAM exception table should have the vector address of the SPC interrupt as the address of the *SpcIntHandler()* function, which is also defined to be in SRAM. See the Interrupts chapter on page 47 for details on configuring the CM0+ exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PSoC device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.

AND See *Boolean Algebra*.

API (Application Programming Interface) A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications.

array An array, also known as a vector or list, is one of the simplest data structures in computer programming. Arrays hold a fixed number of equally-sized data elements, generally of the same data type. Individual elements are accessed by index using a consecutive range of integers, as opposed to an associative array. Most high-level programming languages have arrays as a built-in data type. Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers; for example, by a group of two integers. One- and two-dimensional arrays are the most common. Also, an array can be a group of capacitors or resistors connected in some common form.

assembly A symbolic representation of the machine language of a specific processor. Assembly language is converted to machine code by an assembler. Usually, each line of assembly code produces one machine instruction, though the use of macros is common. Assembly languages are considered low-level languages; where as C is considered a high-level language.

asynchronous A signal whose data is acknowledged or acted upon immediately, irrespective of any clock signal.

attenuation The decrease in intensity of a signal as a result of absorption of energy and of scattering out of the path to the detector, but not including the reduction due to geometric spreading. Attenuation is usually expressed in dB.

B

bandgap reference A stable voltage reference design that matches the positive temperature coefficient of V_T with the negative temperature coefficient of V_{BE}, to produce a zero temperature coefficient (ideally) reference.

bandwidth

1. The frequency range of a message or information processing system measured in hertz.
2. The width of the spectral region over which an amplifier (or absorber) has substantial gain (or loss); it is sometimes represented more specifically as, for example, full width at half maximum.

bias

1. A systematic deviation of a value from a reference value.
2. The amount by which the average of a set of values departs from a reference value.
3. The electrical, mechanical, magnetic, or other force (field) applied to a device to establish a reference level to operate the device.

bias current The constant low-level DC current that is used to produce a stable operation in amplifiers. This current can sometimes be changed to alter the bandwidth of an amplifier.

binary The name for the base 2 numbering system. The most common numbering system is the base 10 numbering system. The base of a numbering system indicates the number of values that may exist for a particular positioning within a number for that system. For example, in base 2, binary, each position may have one of two values (0 or 1). In the base 10, decimal, numbering system, each position may have one of ten values (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

bit A single digit of a binary number. Therefore, a bit may only have a value of ‘0’ or ‘1’. A group of 8 bits is called a byte. Because the PSoC's M8CP is an 8-bit microcontroller, the PSoC devices's native data chunk size is a byte.

bit rate (BR) The number of bits occurring per unit of time in a bit stream, usually expressed in bits per second (bps).

block
1. A functional unit that performs a single function, such as an oscillator.
2. A functional unit that may be configured to perform one of several functions, such as a digital PSoC block or an analog PSoC block.

Boolean Algebra In mathematics and computer science, Boolean algebras or Boolean lattices, are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the set theoretic operations union, intersection, and complement. Boolean algebra also defines a set of theorems that describe how Boolean equations can be manipulated. For example, these theorems are used to simplify Boolean equations, which will reduce the number of logic elements needed to implement the equation.

The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR, and NOT. In describing circuits, NAND (NOT AND), NOR (NOT OR), XNOR (exclusive NOT OR), and XOR (exclusive OR) may also be used. Mathematicians often use + (for example, A+B) for OR and • for AND (for example, A*B) (in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated (for example, ~A, A_, !A).

break-before-make The elements involved go through a disconnected state entering (‘break”) before the new connected state (“make”).

broadcast net A signal that is routed throughout the microcontroller and is accessible by many blocks or systems.

buffer
1. A storage area for data that is used to compensate for a speed difference, when transferring data from one device to another. Usually refers to an area reserved for I/O operations, into which data is read, or from which data is written.
2. A portion of memory set aside to store data, often before it is sent to an external device or as it is received from an external device.
3. An amplifier used to lower the output impedance of a system.

bus
1. A named connection of nets. Bundling nets together in a bus makes it easier to route nets with similar routing patterns.
2. A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, address[7:0].
3. One or more conductors that serve as a common connection for a group of related devices.

byte A digital storage unit consisting of 8 bits.

C

C A high-level programming language.

capacitance A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.

capture	To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file.
chaining	Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another.
checksum	The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a pre-determined value.
clear	To force a bit/register to a value of logic ‘0’.
clock	The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks.
clock generator	A circuit that is used to generate a clock signal.
CMOS	The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor.
comparator	An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements.
compiler	A program that translates a high-level language, such as C, into machine language.
configuration	In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance.
configuration space	In PSoC devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’.
crowbar	A type of over-voltage protection that rapidly places a low-resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value.
CPUSS	CPU subsystem
crystal oscillator	An oscillator in which the frequency is controlled by a piezoelectric crystal. Typically a piezoelectric crystal is less sensitive to ambient temperature than other circuit components.
cyclic redundancy check (CRC)	A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression.

D

data bus	A bi-directional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions.
data stream	A sequence of digitally encoded signals used to represent information in transmission.
data transmission	Sending data from one place to another by means of signals over a channel.
debugger	A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory.
dead band	A period of time when neither of two or more signals are in their active state or in transition.
decimal	A base-10 numbering system, which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 (called digits) together with the decimal point and the sign symbols + (plus) and - (minus) to represent numbers.
default value	Pertaining to the pre-defined initial, original, or specific setting, condition, value, or action a system will assume, use, or take in the absence of instructions from the user.
device	The device referred to in this manual is the PSoC device, unless otherwise specified.
die	An non-packaged integrated circuit (IC), normally cut from a wafer.
digital	A signal or function, the amplitude of which is characterized by one of two discrete values: ‘0’ or ‘1’.
digital blocks	The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
digital logic	A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system.
digital-to-analog (DAC)	A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation.
direct access	The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data.
duty cycle	The relationship of a clock period *high time* to its *low time*, expressed as a percent.

E

External Reset (XRES_N)	An active high signal that is driven into the PSoC device. It causes all operation of the CPU and blocks to stop and return to a pre-defined state.

F

falling edge A transition from a logic 1 to a logic 0. Also known as a negative edge.

feedback The return of a portion of the output, or processed portion of the output, of a (usually active) device to the input.

filter A device or process by which certain frequency components of a signal are attenuated.

firmware The software that is embedded in a hardware device and executed by the CPU. The software may be executed by the end user, but it may not be modified.

flag Any of various types of indicators used for identification of a condition or event (for example, a character that signals the termination of a transmission).

Flash An electrically programmable and erasable, *volatile* technology that provides users with the programmability and data storage of EPROMs, plus in-system erasability. Nonvolatile means that the data is retained when power is off.

Flash bank A group of flash ROM blocks where flash block numbers always begin with '0' in an individual flash bank. A flash bank also has its own block level protection information.

Flash block The smallest amount of flash ROM space that may be programmed at one time and the smallest amount of flash space that may be protected. A flash block holds 64 bytes.

flip-flop A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until it is made to change to the other state by application of the corresponding signal.

frequency The number of cycles or events per unit of time, for a periodic function.

G

gain The ratio of output current, voltage, or power to input current, voltage, or power, respectively. Gain is usually expressed in dB.

gate

1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients.
2. One of many types of combinational logic elements having at least two inputs (for example, AND, OR, NAND, and NOR (also see *Boolean Algebra*)).

ground

1. The electrical neutral line having the same potential as the surrounding earth.
2. The negative side of DC power supply.
3. The reference point for an electrical system.
4. The conducting paths between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

hardware A comprehensive term for all of the physical parts of a computer or embedded system, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.

hardware reset A reset that is caused by a circuit, such as a POR, watchdog reset, or external reset. A hardware reset restores the state of the device as it was when it was first powered up. Therefore, all registers are set to the POR value as indicated in register tables throughout this document.

hexadecimal A base 16 numeral system (often abbreviated and called hex), usually written using the symbols 0-9 and A-F. It is a useful system in computers because there is an easy mapping from four bits to a single hex digit. Thus, one can represent every byte as two consecutive hexadecimal digits. Compare the binary, hex, and decimal representations:

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

So the decimal numeral 79 whose binary representation is 0100 1111b can be written as 4Fh in hexadecimal (0x4F).

high time The amount of time the signal has a value of ‘1’ in one period, for a periodic digital signal.

I

I^2C A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). I^2C is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. I^2C uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.

idle state A condition that exists whenever user messages are not being transmitted, but the service is immediately available for use.

impedance

1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.
2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.

input A point that accepts data, in a device, process, or channel.

input/output (I/O) A device that introduces data into or extracts data from a system.

instruction An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.

instruction mnemonics A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.

integrated circuit (IC) A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.

interface The means by which two systems or devices are connected and interact with each other.

interrupt A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

interrupt service routine (ISR) A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter

1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams.
2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

L

latency The time or delay that it takes for a signal to pass through a given circuit or network.

least significant bit (LSb) The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.

least significant byte (LSB) The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

Linear Feedback Shift Register (LFSR) A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.

load The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).

logic function A mathematical function that performs a digital operation on digital data and returns a digital value.

lookup table (LUT) A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.

low time The amount of time the signal has a value of '0' in one period, for a periodic digital signal.

low-voltage detect (LVD) A circuit that senses V_{DDD} and provides an interrupt to the system when V_{DDD} falls below a selected threshold.

M

M8CP An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PSoC device by interfacing to the flash, SRAM, and register space.

macro A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.

mask

1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference.
2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.

master device A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.

microcontroller An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.

mnemonic A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode A distinct method of operation for software or hardware. For example, the Digital PSoC block may be in either counter mode or timer mode.

modulation A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux)

1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

negative edge A transition from a logic 1 to a logic 0. Also known as a falling edge.

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise

1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR See *Boolean Algebra*.

NOT See *Boolean Algebra*.

O

OR See *Boolean Algebra*.

oscillator A circuit that may be crystal controlled and is used to generate a clock frequency.

output The electrical signal or signals which are produced by an analog or digital block.

P

parallel The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.

parameter Characteristics for a given block that have either been characterized or may be defined by the designer.

parameter block A location in memory where parameters for the SSC instruction are placed prior to execution.

parity A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).

path
1. The logical sequence of instructions executed by a computer.
2. The flow of an electrical signal through a circuit.

pending interrupts An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.

phase The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).

pin A terminal on a hardware component. Also called lead.

pinouts The pin number assignment: the relation between the logical inputs and outputs of the PSoC device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names.

port A group of pins, usually eight.

positive edge A transition from a logic 0 to a logic 1. Also known as a rising edge.

posted interrupts An interrupt that is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.

Power On Reset (POR) A circuit that forces the PSoC device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*.

program counter The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.

protocol A set of rules. Particularly the rules that govern networked communications.

PSoC® Cypress's Programmable System-on-Chip (PSoC®) devices.

PSoC blocks *See analog blocks* and *digital blocks*.

PSoC Creator™ The software for Cypress's next generation Programmable System-on-Chip technology.

pulse A rapid change in some characteristic of a signal (for example, phase or frequency), from a baseline value to a higher or lower value, followed by a rapid return to the baseline value.

pulse width modulator (PWM) An output in the form of duty cycle which varies as a function of the applied measure.

R

RAM An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.

register A storage device with a specific capacity, such as a bit or byte.

reset A means of bringing a system back to a know state. See *hardware reset* and *software reset*.

resistance The resistance to the flow of electric current measured in ohms for a conductor.

revision ID A unique identifier of the PSoC device.

ripple divider An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to 2^n - 1.

rising edge *See positive edge*.

ROM An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.

routine A block of code, called by another block of code, that may have some general or frequent use.

routing Physically connecting objects in a design according to design rules set in the reference library.

runt pulses In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling The process of converting an analog signal into a series of digital values or reversed.

schematic A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.

seed value An initial value loaded into a linear feedback shift register or random number generator.

serial
1. Pertaining to a process in which all events occur one after the other.
2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.

set To force a bit/register to a value of logic 1.

settling time The time it takes for an output signal or value to stabilize after the input has changed from one value to another.

shift The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.

shift register A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.

sign bit The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.

signal A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.

silicon ID A unique identifier of the PSoC silicon.

skew The difference in arrival time of bits transmitted at the same time, in parallel transmission.

slave device A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.

software A set of computer programs, procedures, and associated documentation about the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed.

software reset A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not PSoC blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at flash address 0x0000.

SRAM An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.

SROM An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash.

stack A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.

stack pointer A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.

state machine The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.

sticky A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The PSoC SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous

1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the PSoC device and reading this manual.

user modules Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital PSoC Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

V_{DDD} A name for a power net meaning "voltage drain." The most positive power supply signal. Usually 5 or 3.3 volts.

volatile Not guaranteed to stay the same value or level when not in scope.

V_{SS} A name for a power net meaning "voltage source." The most negative power supply signal.

W

watchdog timer A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time.

waveform The representation of a signal as a plot of amplitude versus time.

X

XOR See *Boolean Algebra*.

Index