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.

PSoC 4500S

PSoC 4 Architecture Technical Reference Manual (TRM)

Document No. 002-26656 Rev. *B

April 2, 2020

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

Copyrights

Contents Overview

Contents

Section A: Overview

This section encompasses the following chapters:

- Introduction chapter on page 16
- Getting Started chapter on page 24
- Document Organization and Conventions chapter on page 25

Document Revision History

Revision	Issue Date	Description of Change
**	July 15, 2019	Initial version of PSoC 4500S TRM.
*A	Mar 13, 2020	Updated Section A: Figure1-1. Updated Section 1.3 CPU System: Moved Memory chapter into CPU System. Updated Section 1.5 Fixed-Function Digital: Moved LCD Segment Drive to Section 1.5.5. Updated Section B, Section C, Section D, Section E, and Section F diagrams and main chapter descriptions. Updated Table 1-1: Added PSoC 4500S device features. Updated Table 8-3: Inserted 1, 2, 3 rows under the value column of Table 8-3. Updated Chapter 8.5: Changed “io_data” as “io_data_in” in the entire chapter. Updated Table 8-6: Updated OPC[1:0], DU_OPC[3:0] and DU_DATA0_SEL[1:0] descriptions. Update Figure 9-1 and Figure 9-3. Added new Clock Supervision sections under Chapter 9. Updated EXCO relevant description such as register name, clock source, and so on. Added Table 9-12 for peripheral clock multiplexer output mapping. Added new Chapter 14 Trigger Multiplexer Block. Updated Section 15.1.4: Changed software reset key from A05F0004 to 05FA0004. Updated Chapter 17: Figure 17-26 and Figure 17-27. Updated Chapter 19: Remove LCD Deep Sleep operates mode. Updated Chapter 22: Removed CSDv2 proprietary information and added CapSense note for user. Updated Section 25.4.1: Changed Table 25-3 IDCODE from 0x0BB11477 to 0x0BC11477. Updated Section 25.5.1.1: Deleted Primary and Secondary SWD Pin. Updated Section 26.2: Added note contents. Updated Section 26.5.1: Changed the silicon ID as 2500-25FF.
*B	Apr 02, 2020	Updated Section A through Section F: Main chapter description and diagrams. Updated Chapter 9: Figure 9-1 and Figure 9-3. Updated Chapter 16: Figure 16-1 through Figure 16-3 and chapter description corrections. Updated TRM for formatting corrections.

1. Introduction

The PSoC® 4 is a programmable embedded system controller with an Arm® Cortex®-M0+ (CM0+) CPU.

PSoC 4500S devices have these characteristics:

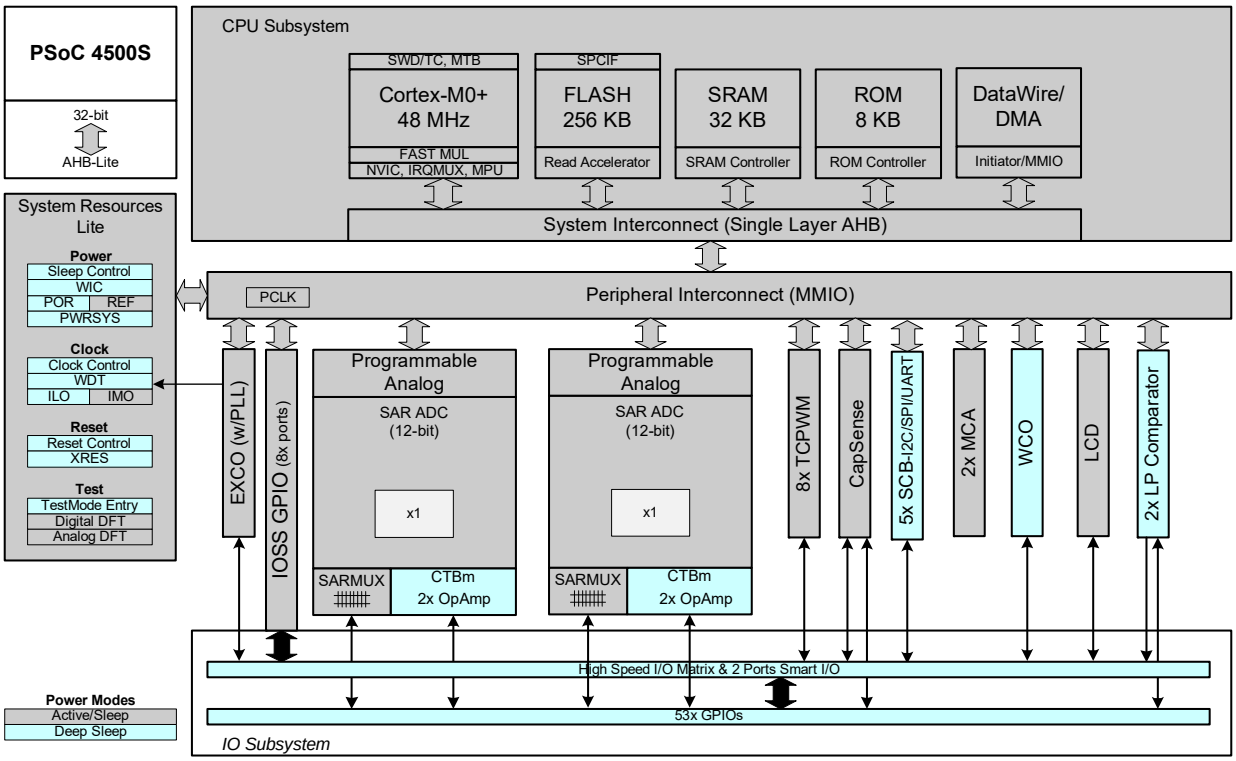
- High-performance, 32-bit single-cycle CM0+ CPU core
- High-performance analog system
- Self and Mutual Capacitive touch sensing (CapSense®)
- Configurable Timer/Counter/Pulse-Width Modulator (TCPWM) block
- Configurable analog blocks for analog signal conditioning
- Configurable communication block with I^2C, Serial Peripheral Interface (SPI), and Universal Asynchronous Receiver/Transmitter (UART) operating mode
- Motor control algorithm acceleration module MCA (Motor Control Accelerator)
- Low-Power operating modes – Sleep and Deep Sleep

This document describes each functional block of the PSoC 4500S in detail. The information in this document will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of PSoC 4500S architecture.

Figure 1-1. PSoC 4500S Block Diagram



1.2 Features

The PSoC 4500S device has these major components:

- 32-bit CM0+ CPU with single-cycle multiply, delivering up to 0.9 DMIPS/MHz
- Up to 256 KB flash and 32 KB SRAM
- Direct Memory Access (DMA)
- Up to eight center-aligned PWM with complementary, dead-band programmable outputs
- One Watchdog Timer (WDT) and three general-purpose timers with interrupt capability
- Two 12-bit 1-Msps SAR ADCs with differential and single-ended modes, and Channel Sequencer with signal averaging. Simultaneous sampling is provided.
- Two MCAs used for motor control algorithm acceleration
- Up to four opamps with reconfigurable high-drive external and high-bandwidth internal drive, Comparator modes, and ADC input buffering capability. Opamps can operate in Deep Sleep Low-Power mode.
- Two Low-Power Comparators (LPCOMP)
- Up to five independent run-time reconfigurable Serial Communication Blocks (SCBs) with re-configurable I^2C, SPI, or UART functionality
- Two Smart I/O™ blocks, with the ability to perform Boolean functions in the I/O signal path
- CapSense
- Segment LCD direct drive
- Low-Power operating modes including Sleep and Deep Sleep
- Programming and debugging system through Serial Wire Debug (SWD)
- Fully supported by PSoC Creator™ Integrated Development Environment (IDE) tool

1.3 CPU System

1.3.1 Processor

The heart of PSoC is a 32-bit CM0+ CPU core running at 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 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep Sleep mode.

1.3.3 DMA

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.

1.3.4 Memory

1.3.4.1 Flash

The PSoC 4 memory subsystem 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% of single-cycle SRAM access performance on an average.

1.3.4.2 SRAM

The PSoC 4 memory subsystem provides SRAM, which is retained in all power modes of the device.

1.4 System-Wide Resources

1.4.1 Clocking System

The clocking system consists of the Internal Main Oscillator (IMO) and Internal Low-speed Oscillator (ILO) as internal clocks and has provisions for an External Clock (EXTCLK), External Crystal Oscillator (ECO), and Watch Crystal Oscillator (WCO).

The IMO with an accuracy of ±2% 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 low-frequency clock (LFCLK) to generate clocks for peripheral operation in Deep Sleep mode. The ILO clock frequency nominally is 32 kHz and it is a relatively inaccurate (20-80 kHz) oscillator.

An EXTCLK 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 ECO generates an accurate clock, with frequency ranging from 4 MHz to 33 MHz, using an external crystal.

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

1.4.2 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} for the power digital section and V_{DDA} for noise isolation of the 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 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 LFCLK is ON and the low-frequency peripherals (see Table 12-2) are in operation.

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

1.4.3 GPIO

Every GPIO has the following characteristics:

- Eight drive strength modes
- Individual control of input and output buffer enabling/disabling in addition to the drive strength modes
- Hold mode for latching previous state
- Selectable slew rates
- Interrupt generation – edge-triggered

In addition, the device has up to two Smart I/O blocks that provide 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 ports that are 8-bit wide. A high-speed I/O matrix (HSIOM) is used to multiplex between various signals that may connect to an I/O pin. Pin locations for fixed-function peripherals are also fixed.

1.4.4 Watchdog Timer (WDT)

The device has one 16-bit WDT, 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, two 16-bit and one 32-bit up-counting timers are available for general-purpose use.

1.5 Fixed-Function Digital

1.5.1 Timer/Counter/PWM (TCPWM) Block

The TCPWM block consists of up to eight 16-bit counters 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.

1.5.2 Serial Communication Blocks (SCB)

The device has five 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) modes
- Standard UART transmitter and receiver function with SmartCard reader (ISO7816), Infrared Data Association (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

1.5.3 Motor Control Accelerator (MCA)

The device has two MCAs. The MCA block is designed specifically for motor control algorithm acceleration. It is used in the motor control PSoC to accelerate 32-bit divide and square root operations. This acceleration reduces the CPU execution time for Field-Oriented Control (FOC) and Power Factor Correction (PFC) algorithms.

Each of the two PSoC 4500S MCA modules provides the following features:

- Divide and square root operation run off High-frequency Clock (HFCLK) high-speed clock
- 32-bit unsigned integer sequential divider
- 32-bit unsigned integer sequential square root
- Input operands programmed through MCA register
- Output results accessed through MCA register
- Two methods to perform the divide operation
- Two methods to run the square root operation

1.5.4 LCD Segment Drive

The device has an LCD controller, which can drive up to eight commons; every GPIO can be configured to drive a common or a segment. It uses full digital method (PWM) to drive LCD segments, and does not require generation of internal LCD voltages.

1.6 Analog System

1.6.1 SAR ADC

The device has two configurable 12-bit 1-Msps SAR ADCs. Each ADC provides three internal voltage references (V_{DDA}, $V_{DDA}/2$, and V_{REF}) and an external reference through a GPIO pin. The SAR hardware sequencer is available, which scans multiple channels without CPU intervention.

1.6.2 Continuous Time Block mini (CTBm)

The device has two CTBm blocks. Each CTBm provides continuous time functionality at the entry and exit points of the analog subsystem. The CTBm has two highly configurable and high-performance opamps with a switch routing matrix. The opamps can also work in comparator mode.

The block allows open-loop opamp, linear buffer, and comparator functions to be performed without external components. PGAs, voltage buffers, filters, and trans-impedance amplifiers can be realized with external components. The CTBm block can work in Active, Sleep, and Deep Sleep modes.

1.6.3 Low-Power Comparators (LPCOMP)

The device has a pair of LPCOMPs, which can also operate in low power modes. This allows the analog system blocks to be disabled while retaining the ability to monitor external voltage levels during Low-Power modes. The comparator outputs are normally synchronized to avoid metastability unless operating in an asynchronous power mode where the system wake-up circuit is activated by a comparator switch event. The LPCOMP outputs can be routed to pins.

1.7 Special Function Peripherals

1.7.1 CapSense

Capacitive touch sensors are used in user interface that use human body capacitance to detect the presence of a finger on or near a sensor. Cypress CapSense solutions bring elegant, reliable, and easy-to-use capacitive touch sensing functionality to the products including Internet of Things (IoT), industrial, automotive and home appliance. 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.

1.7.1.1 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.

1.8 Program and Debug

PSoC 4 devices support 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.

1.9 Device Feature Summary

Table 1-1 shows the summary of PSoC 4500S device.

Table 1-1. PSoC 4500S Device Summary

Feature	PSoC 4500S
Maximum CPU Frequency	48 MHz
Flash	256 KB
SRAM	32 KB
DMA	8 channels
GPIOs (max)	53
Smart I/O	2 ports
CapSense	Available
LCD Driver	Available
TCPWM	8
16-bit Timer	2
32-bit Timer	1
SCB	5
Opamp (CTBm)	4
LPCOMP	2
SAR ADC	2 (12-bit, 1 Msps)
WCO	Available
ECO and PLL Subsystem (EXCO)	Available
Power Modes	Active, Sleep, Deep Sleep
MCA	Available

2. Getting Started

2.1 Support

Support for PSoC 4 products is available online at www.cypress.com/psoc4. 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 Software Upgrades

Cypress provides scheduled upgrades and version enhancements for PSoC Creator free of charge. Upgrades are available for download 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 and the accessories you need to successfully develop PSoC projects. Visit the Cypress Online Store website at www.cypress.com/cypress-store. Development kits are also available from Digi-Key, Avnet, Arrow, and Future.

2.4 Application Notes

See application note *AN79953 - Getting Started with PSoC 4* for additional information on PSoC 4 device capabilities and quickly create a simple PSoC application using PSoC Creator and PSoC 4 development kits.

3. Document Organization and Conventions

This document includes the following sections:

- Section B: CPU Subsystem on page 29
- Section C: System Resources Subsystem (SRSS) on page 64
- Section D: Digital System on page 116
- Section E: Analog System on page 196
- Section F: Program and Debug on page 248

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 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 4500S: PSoC 4 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
kbps	kilobits per second
kHz	kilohertz (32.000)
kΩ	kilohms
ksps	kilosamples per second
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 and Initializations

This table lists the acronyms and initializations used in this document.

Table 3-2. Acronyms and Initializations

Acronym	Definition
ABUS	analog output bus
ADC	analog-to-digital converter
AC	alternating current
ACK	acknowledge
ADC	analog-to-digital converter
AGC	automatic gain control
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm data transfer bus
AP	access port
API	application programming interface
APnDP	AP not DP
APSR	application program status register
ATRIM	amplitude trim
BOD	brownout detect
BOM	bill of materials
BPU	breakpoint unit
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CC	capture/compare
CI	carry in
CMP	compare
CO	carry out
COM	LCD common signal
CPHA	clock phase
CPOL	clock polarity
CPU	central processing unit
CPUSS	CPU subsytem
CRC	cyclic redundancy check
CSD	CapSense sigma delta
CSI	clock supervision interrupt
CSX	CapSense cross-point
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
DL	drive level
DMA	direct memory access
DMIPS	Dhrystone million instructions per second

Table 3-2. Acronyms and Initializations *(continued)*

Acronym	Definition
DO	digital or data output
DP	debug port
DSI	digital signal interface
DSM	Deep Sleep mode
DU	data unit
DW	data wire
DWT	debug watchpoint
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
EOC	end-of-conversion
EOS_INTR	end-of-scan interrupt
ESR	equivalent series resistance
EPSR	execution program status register
EZI2C	easy I2C
EZSPI	easy SPI
FIFO	first in first out
FOC	field-oriented control
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
IrDA	Infrared Data Association
ITO	indium tin oxide
IMO	internal main oscillator
I/O	input/output
IOR	I/O read
IOW	I/O write
IPSR	interrupt program status register
IRQ	interrupt request
ISB	instruction synchronization barrier
ISR	interrupt service routine
LCD	liquid crystal display
LFCLK	low-frequency clock
LFSR	linear feedback shift register
LIFO	last in first out
LPCOMP	low-power comparator
LR	link register
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
MCA	motor control accelerator

Table 3-2. Acronyms and Initializations *(continued)*

Acronym	Definition
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
PEP	program erase program
PFC	power factor correction
PGA	programmable gain amplifier
PLL	phase-locked loop
PM	power management
POR	power-on reset
PROT_FAULT	protection fault reset
PRS	pseudo random sequence
PSoC®	Programmable System-on-Chip
PSP	process stack pointer
PSR	program status register
PWM	pulse-width modulator
RAM	random-access memory
RETI	return from interrupt
RF	radio frequency
ROM	read only memory
RMP	risk mitigation bits
RMS	root mean square
RnW	read not write bit
RTC	real-time clocl
RW	read/write
RX	receiver output
RZI	return-to-zero-inverted
SAR	successive approximation register
SC	switched capacitor
SCB	serial communication block
SEG	LCD segment signal
SFlash	supervisory flash
SHCSR	system handler control and state register
SIO	special I/O
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer

Table 3-2. Acronyms and Initializations *(continued)*

Acronym	Definition
SPC	system performance controller
SPCIF	system performance controller interface
SPI	serial peripheral interface, a communications protocol
SPIM	serial peripheral interface master
SPIS	serial peripheral interface slave
SRAM	static random-access memory
SRES	software initiated reset
SROM	supervisory read only memory
SSC	supervisory system call
SVCall	supervisor call
SWD	serial wire debug
SWDCK	serial-wire debug clock
SWD IO	serial-wire debug input/output
SW-DP	SWD debug port
SYSCLK	system clock
TAR	turn-around time
TC	terminal count
TCPWM	timer, counter, PWM
Trn	Turnaround
TX	transmitter output
UART	universal asynchronous receiver/transmitter
UDB	universal digital block
ULP	ultra-low-power
USB	universal serial bus
USBIO	USB I/O
VTOR	vector table offset register
WCO	watch crystal oscillator
WDR	watchdog reset
WDT	watchdog timer
WDTRIM	watchdog trim
WFI	wait for interrupt
WIC	wakeup interrupt controller
XRES	external reset
XRES_N	external reset, active low

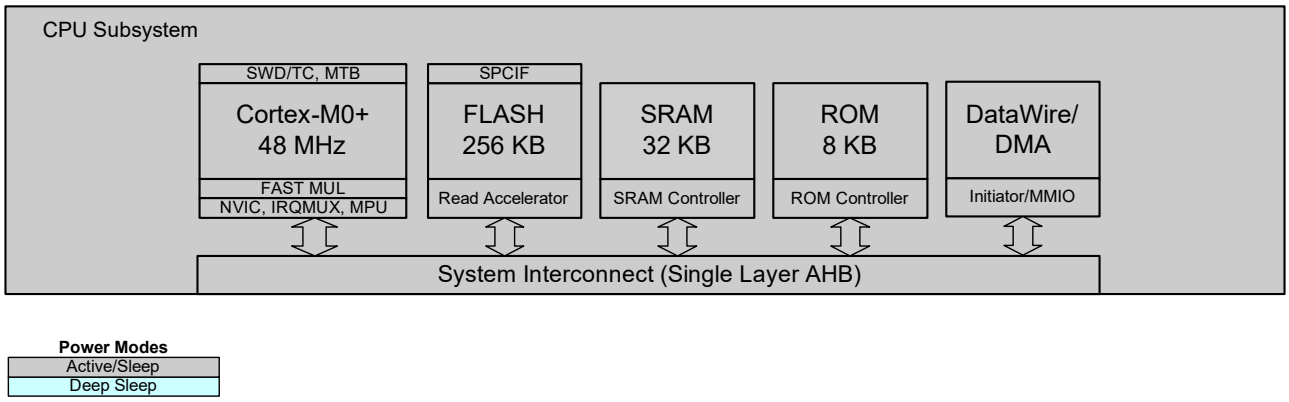
Section B: CPU Subsystem

This section encompasses the following chapters:

- Cortex-M0+ CPU chapter on page 30
- DMA Controller Modes chapter on page 36
- Interrupts chapter on page 52
- Device Security chapter on page 62

Top Level Architecture

CPU Subsystem Block Diagram



4. Cortex-M0+ CPU

The PSoC 4 Arm CM0+ 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. CM0+ also features a single-cycle 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a NVIC, a SYSTICK timer, and debug.

This section gives an overview of the CM0+ processor. For more details, see the *Arm Cortex-M0+ Generic User Guide* or *Technical Reference Manual*, both available at www.arm.com.

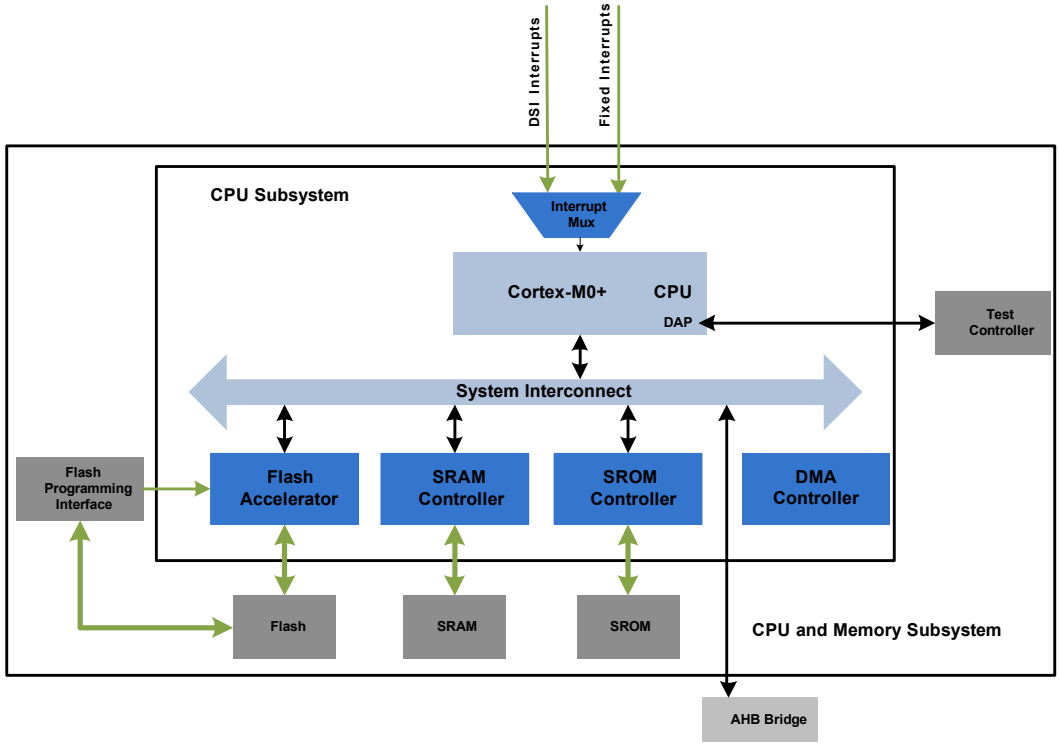
4.1 Features

The PSoC 4 CM0+ 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 supports 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

CM0+ 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 33). It has a single-cycle 32-bit multiplication instruction.

4.4 Address Map

CM0+ 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 the code can be executed from the code and SRAM regions.

Table 4-1. CM0+ 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 4 implementation-specific.

4.5 Registers

CM0+ has sixteen 32-bit registers, as Table 4-2 shows. See the Arm documentation for details.

- 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 SP to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP).
- R14 – Link register (LR). Stores the return Program Counter (PC) during function calls.
- R15 – Program counter. This register can be written to control program flow.

Table 4-2. CM0+ 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]	Stack pointer (SP) is register R13. In thread mode, bit[1] of the CONTROL register indicates which stack pointer to use: 0 = MSP. This is the reset value. 1 = PSP On reset, the processor loads the MSP with the value from address 0x00000000.
LR (R14)	RW	Undefined	Link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004]	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 Execution Program Status Register (EPSR) T-bit at reset and must be 1.
PSR	RW	Undefined	Program status register (PSR) combines: ■ Application Program Status Register (APSR) ■ EPSR ■ Interrupt Program Status Register (IPSR)
APSR	RW	Undefined	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	IPSR contains the exception number of the current interrupt service routine (ISR).
PRIMASK	RW	0	PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	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. CM0+ 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
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 … 35 =IRQ19

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 52 for a list of exceptions.

4.6 Operating Modes

The CM0+ processor supports two operating modes:

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

In thread mode, use the MSR instruction to set the SP bit in the CONTROL register. When changing the SP, use an Instruction Synchronization Barrier (ISB) instruction immediately after the MSR instruction. This action ensures that the instructions after the ISB execute using the new SP.

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

4.7 Instruction Set

The CM0+ implements a version of the Thumb instruction set, as Table 4-4 shows. For more details, see the *Arm Cortex-M0+ Generic User Guide* at www.arm.com.

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 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
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

Table 4-4. Thumb Instruction Set *(continued)*

Mnemonic	Brief Description
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 CM0+ processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

4.7.2 Memory Endianness

CM0+ uses the little-endian format, where the Least Significant Byte (LSB) 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 be used as a countdown value. The Systick timer uses either the CM0+ internal clock or the low-frequency clock (LF_CLK) as the source.

4.9 Debug

PSoC 4 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, available only in the PSoC 4 device, provides DataWire (DW) and 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 you to implement different operation scenarios for 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 provided by a descriptor structure. Each channel has an independent descriptor structure.

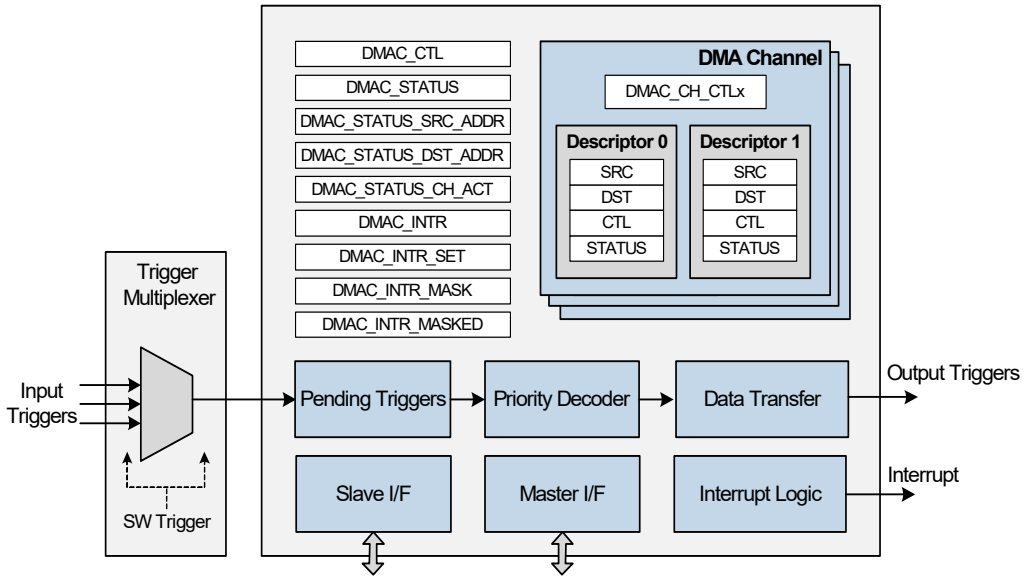
The DMA controller provides Active/Sleep functionality and is not available in the Deep Sleep power mode.

5.1 Block Diagram

DMA transfers data to and from memory, peripherals, and registers. These transfers occur independent of the CPU. 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. 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. 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, such as the source address, destination address, and data width, specific to the transfer. Transfer initiation in the DMA channel is on a trigger event. 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 interface is an AHB-Lite bus master, which allows the DMA controller to initiate AHB-Lite data transfers to the source and destination locations. 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 interface 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. The CPU is generally the master for this bus.

Receipt of a trigger activates a state machine in the DMA controller that goes through 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 the 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 “Transfer Mode” on page 44. 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 “Trigger Multiplexer” on page 37.

5.1.1.1 Trigger Multiplexer

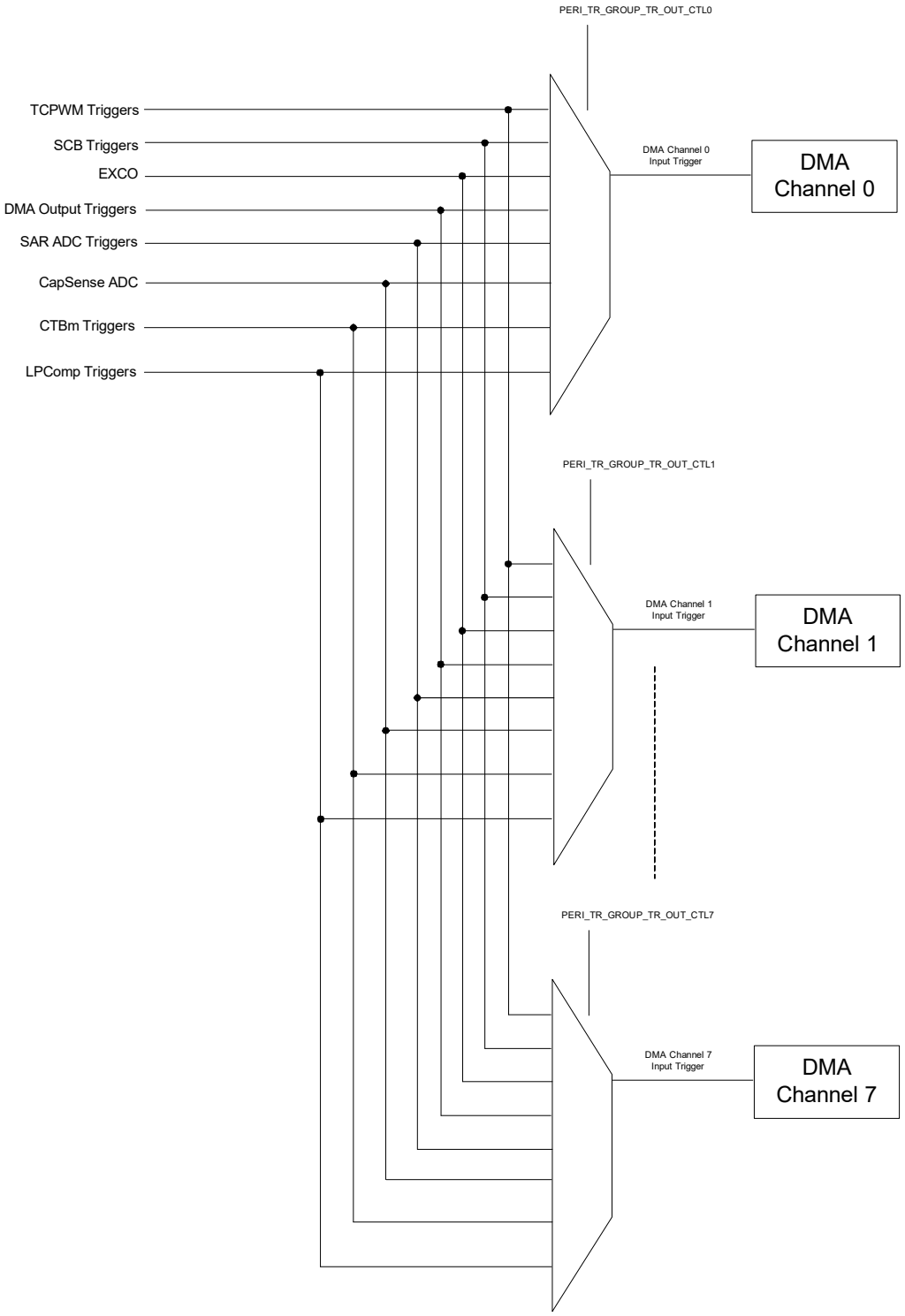
DMA channels can have trigger inputs from different peripheral sources in PSoC. This is routed to the individual DMA channel trigger inputs through the trigger multiplexer.

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 4500S device implements a single trigger group (Trigger group 0), which provides trigger inputs to DMA. The trigger input options can come from TCPWM, SCB, EXCO, DMA output triggers, SAR ADC, CapSense ADC, CTBm triggers, and LPCOMP 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_GROUP0_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
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
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
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	SCB 3 TX request
32	SCB 3 RX request
33	SCB 4 TX request
34	SCB 4 RX request
35	EXCO trigger
36	DMA Channel 0 trigger out
37	DMA Channel 1 trigger out
38	DMA Channel 2 trigger out
39	DMA Channel 3 trigger out
40	DMA Channel 4 trigger out

Table 5-1. Trigger Sources *(continued)*

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
41	DMA Channel 5 trigger out
42	DMA Channel 6 trigger out
43	DMA Channel 7 trigger out
44	SAR ADC0 sample done
45	SAR ADC0 End-of-Conversion (EOC)
46	CSD ADC sample done
47	CTBm[0] cmp0
48	CTBm[0] cmp1
49	SAR ADC1 sample done
50	SAR ADC1 EOC
51	CTBm[1] cmp0
52	CTBm[1] cmp1
53	LPCOMP[0] output
54	LPCOMP[1] output

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 37. 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. 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 interface to load data from the source location.
- State 3 - Storing Data at Destination: The data transfer engine uses the master interface to store data to the destination location.

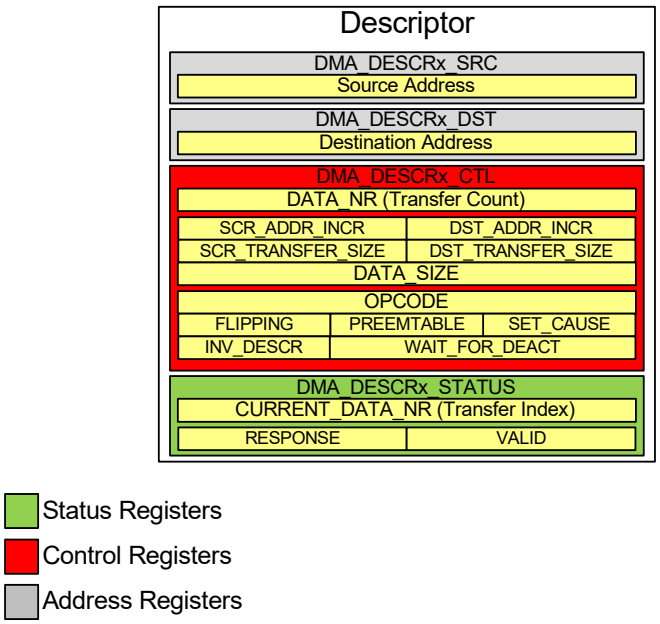
  Depending on the Transfer mode, states 2 and 3 may be performed multiple times.
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation 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. If 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 the source address, 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 whether 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 is determined based on the SCR_TRANSFER_SIZE setting described in “Transfer Size” on page 43.

Destination Address Increment (DST_ADDR_INC): This is a bit setting in the control register, which determines whether 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 “Transfer Size” on page 43.

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 if you expect 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 the Operational mode is allowed to complete undisturbed. If enabled, the current transfer as defined by the 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.

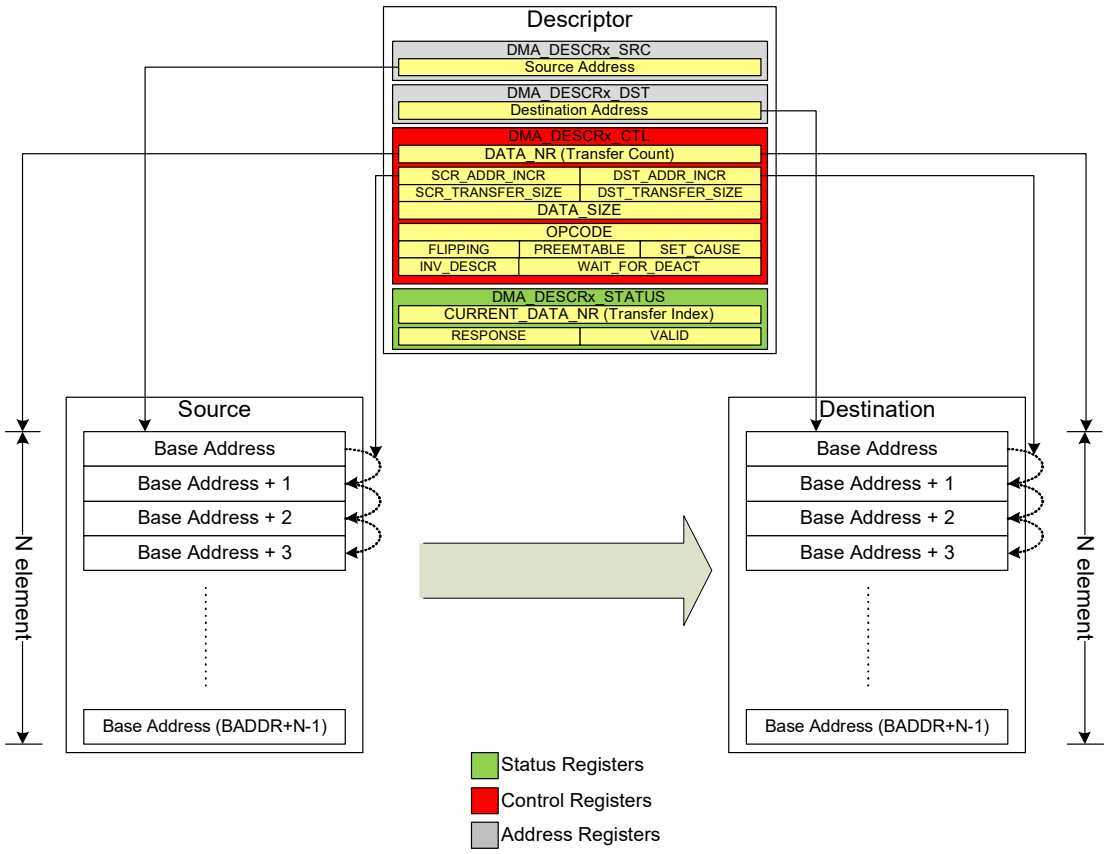
Setting Interrupt Cause (SET_CAUSE): When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

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 “Data Transfer Engine” on page 41. 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 bits. DATA_SIZE can be set to a 32-bit, 16-bit, or 8-bit setting.

The data width of most PSoC 4 peripheral registers is 4 bytes (32 bits); therefore, SCR_TRANSFER_SIZE or DST_TRANSFER_SIZE should typically be set to 32 bits 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 bits to match the width of the PWM register, because the peripheral register width for the TCPWM block (and most PSoC 4 peripherals) is always 32 bits. However, in this example, DATA_SIZE for the destination may still be set to 16 bits 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. If 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. The valid data width is only eight bits, so DATA_SIZE is maintained as eight bits.

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

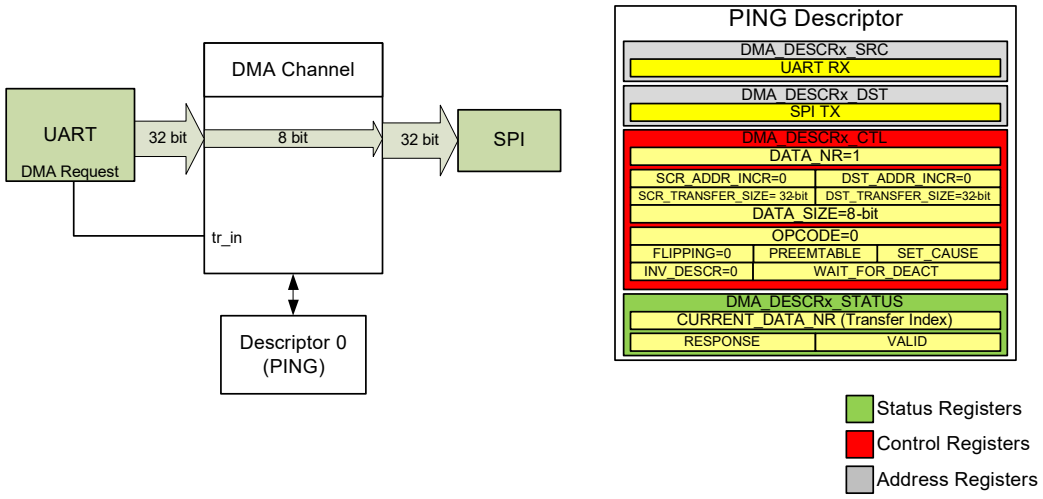
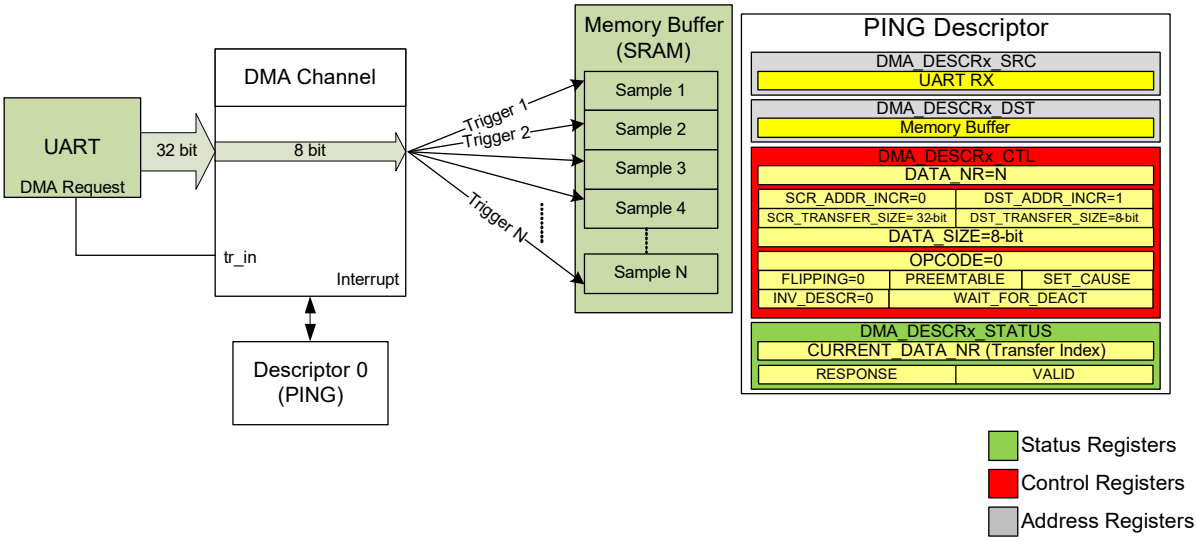


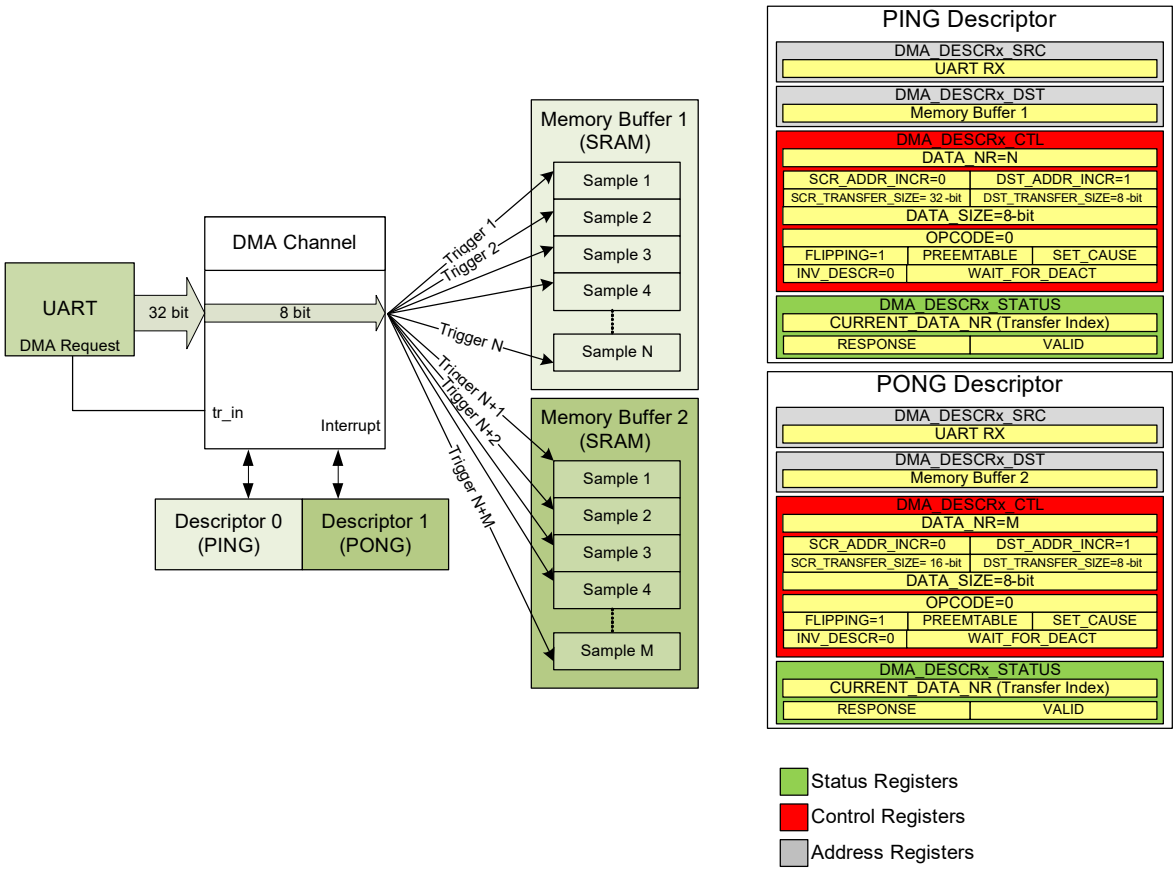
Figure 5-6 describes another use case where the data transfer is between the UART RX register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is a UART. When the trigger is received, the transfer engine will load the data from the UART RX register and store it into 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

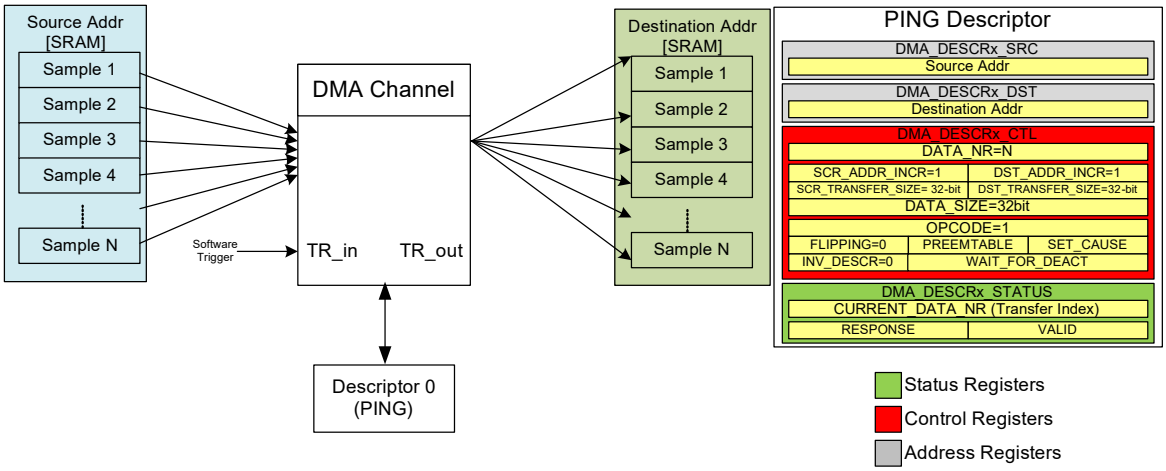


5.2.4.2 *Entire Descriptor Per Trigger (OPCODE 1)*

In this mode of operation, 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

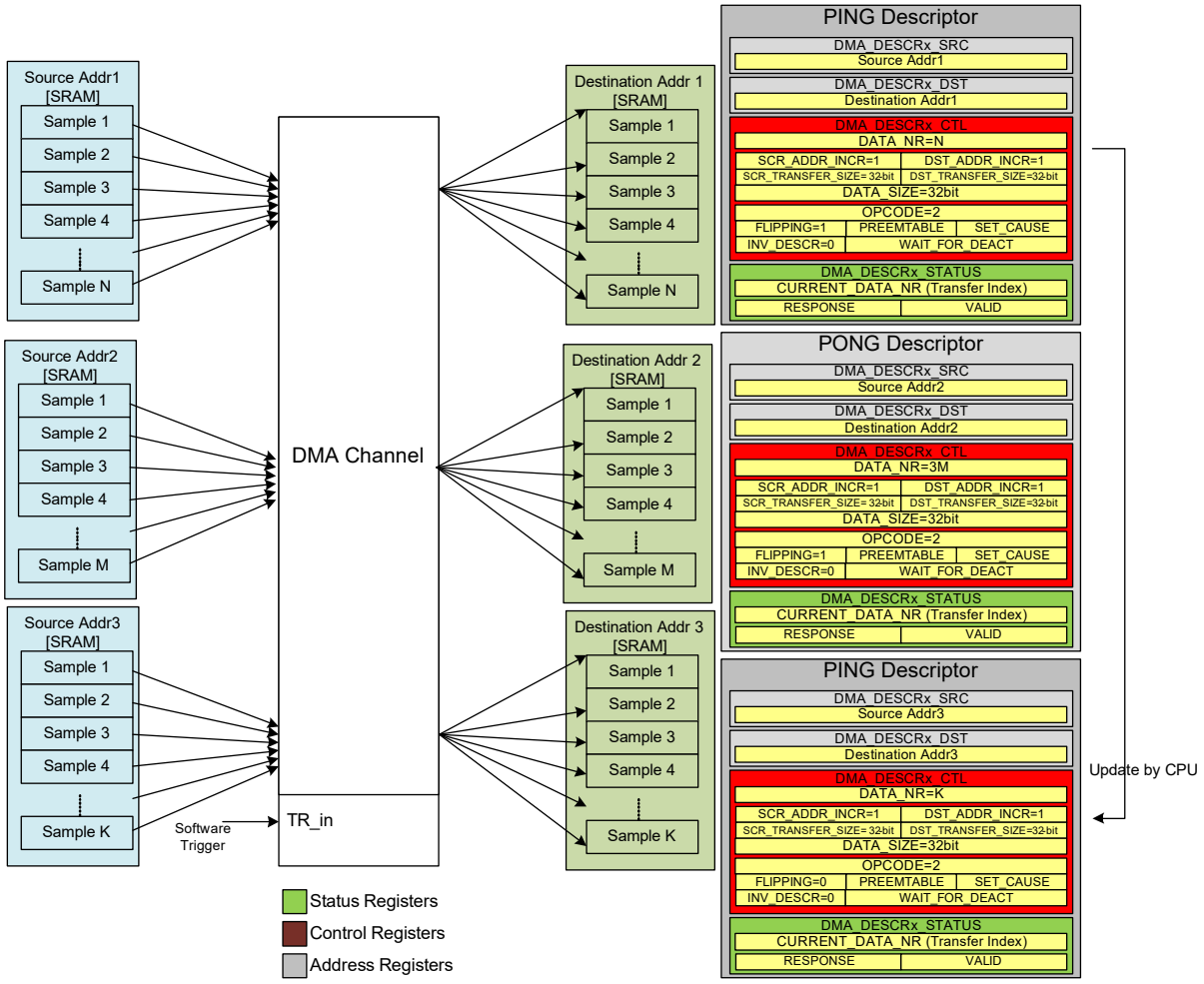


5.2.4.3 Entire Descriptor Chain Per Trigger (OPCODE 2)

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 is invalidated or changed by the CPU.

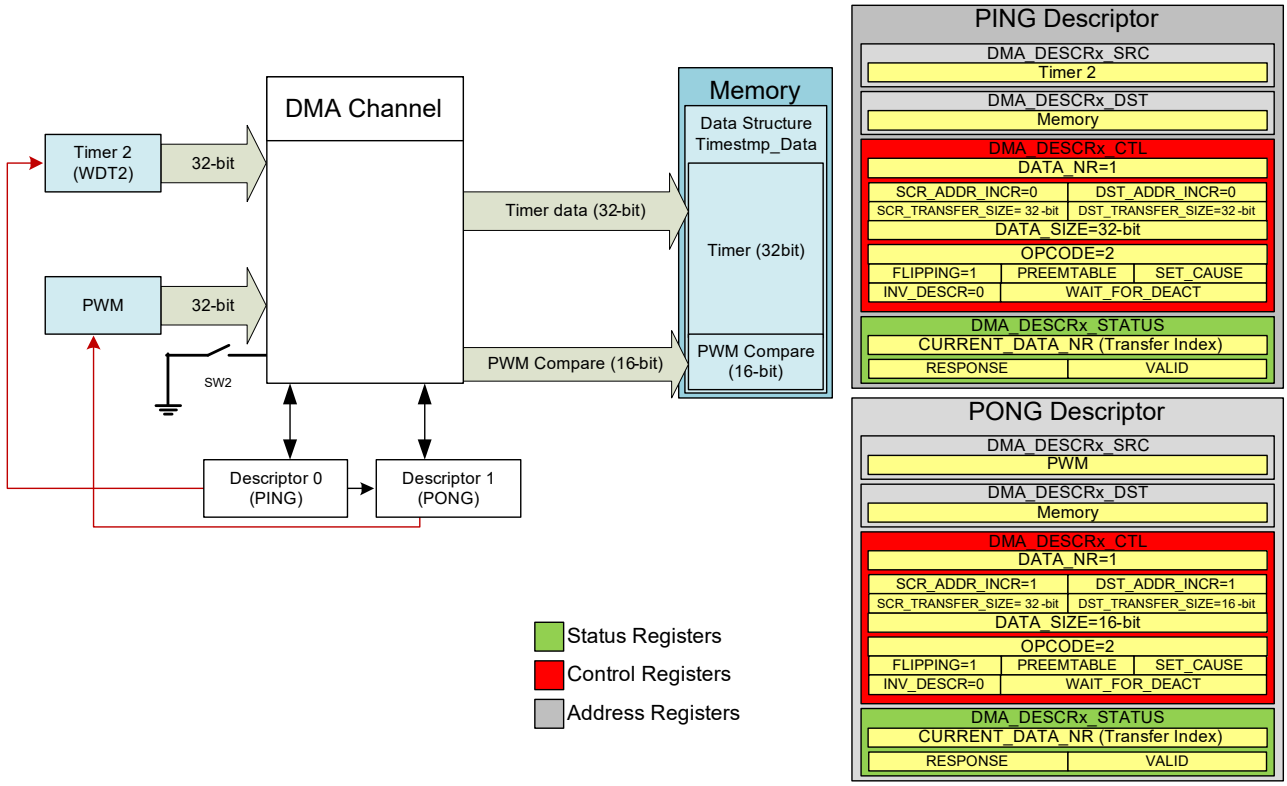
Figure 5-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation, and are on the second iteration of the PING register. FLIPPING is disabled by the CPU.

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 locations. The destination is a data structure that is in consecutive memory locations. One source is Timer 2, which holds a timing data and the other source is a PWM compare register. Both data is stored in consecutive locations in memory.

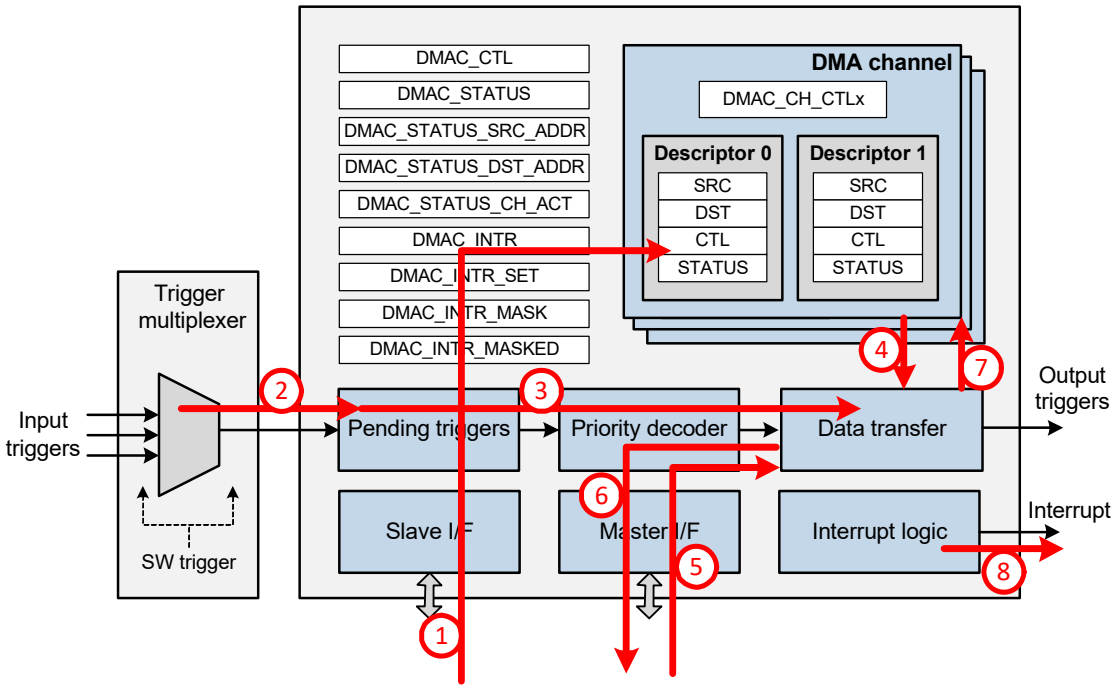
Figure 5-10. OPCODE 2: Multiple Sources to Memory



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 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.

- 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:

Number 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:

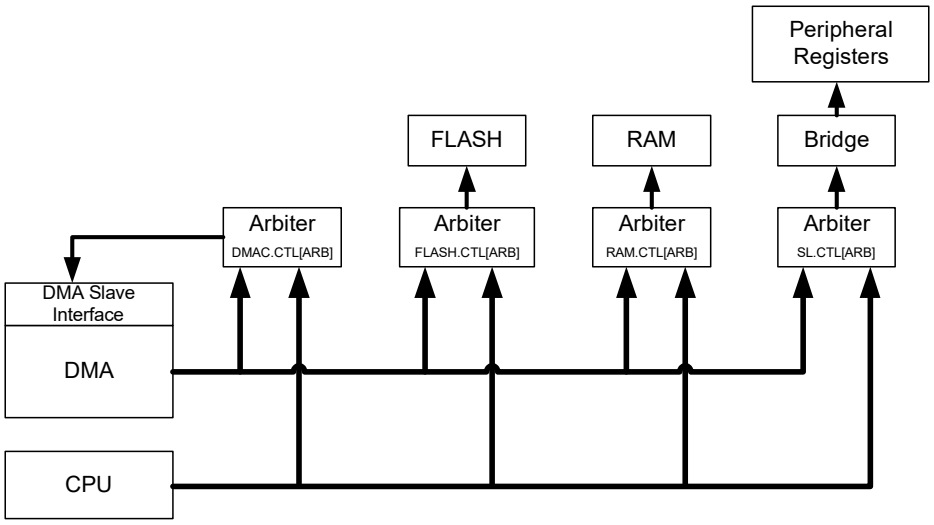
Number of cycles = (12 + LOAD wait states + STORE wait states) + (N–1)*(3 + LOAD wait states + STORE wait states)

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, connect through another slave interface.

Figure 5-12 illustrates this architecture.

Figure 5-12. PSoC 4 Bus Architecture

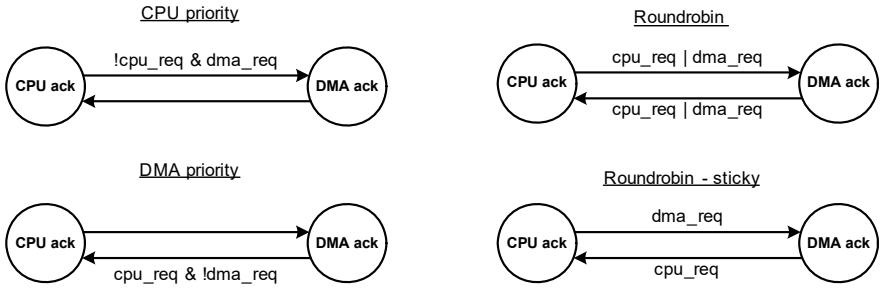


The arbitration policy for each slave can be one of the following:

- 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.

Figure 5-13 illustrates the arbitration models.

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 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

CM0+ CPU in PSoC 4 supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, SCB, 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 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

The PSoC 4 supports the following Interrupt features:

- 32 interrupts
- 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 4 Interrupts Block Diagram

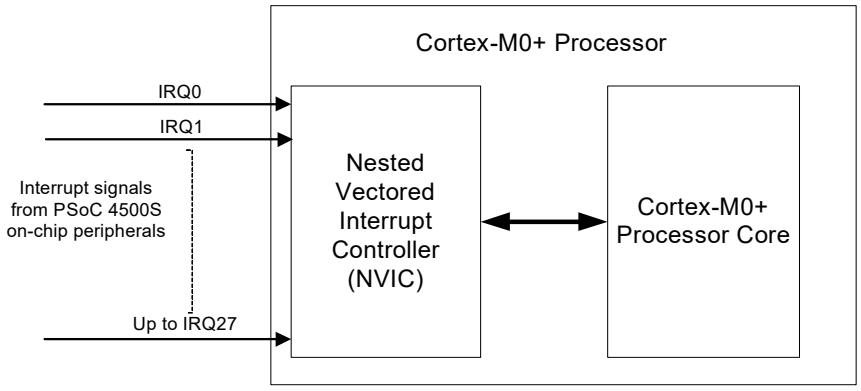


Figure 6-1 shows the interaction between interrupt signals and the CM0+ CPU. PSoC 4 has up to 32 interrupts; these interrupt signals are processed by the NVIC. NVIC takes are care of enabling/disabling individual interrupts, resolving priority, and communicating with the CPU core. The exceptions are 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.

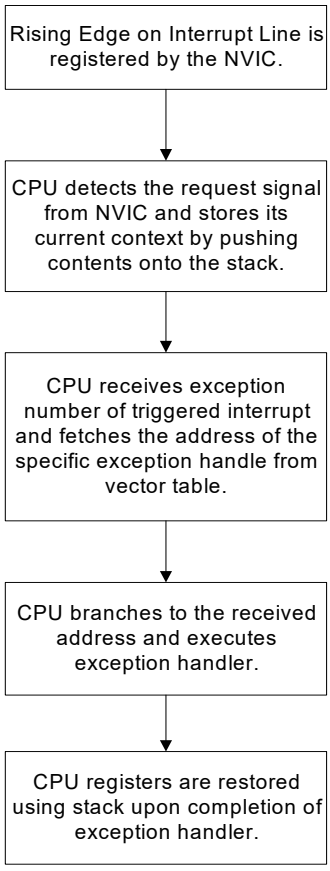
6.3 Interrupts and Exceptions - Operation

6.3.1 Interrupt/Exception Handling

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

1. Assuming that all 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, and sends 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 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 IRQ31). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 6-3. Level Interrupts

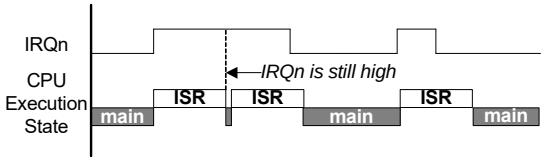


Figure 6-4. Pulse Interrupts

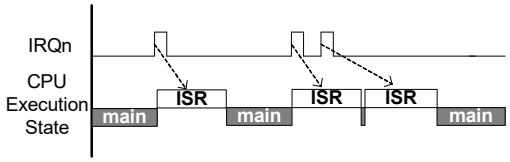


Figure 6-3 and Figure 6-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic LOW), the following sequence of events explains the handling of level and pulse interrupts:

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. 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)	…
47	External Interrupt(IRQ31)	Configurable (0 - 3)	Base_Address + 0xBC

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 MSP value on device reset; it is not considered as an exception. The vector table can be located anywhere in the memory map (flash or SRAM) by modifying the VTOR. This register is part of the System Control Space of CM0+ located at 0xE000ED08. This register takes bits 31:8 of the vector table address; bits 7:0 are reserved. Therefore, the vector table address should be 256 bytes aligned. 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 DIS_RESET_VECT_REL bit of the CPUSS_SYSREQ register is set. The default value of this bit at reset is '0', ensuring that the reset vector is always fetched from SROM. To allow flash read from addresses 0x00000000 and 0x00000004, the DIS_RESET_VECT_REL 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, the boot code in SROM is executed first and then the CPU jumps to address 0x00000004 in the flash to execute the handler in flash. The reset exception address in the SRAM vector table is never used.

Also, when the SYSCALL_REQ 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 "Exception Sources" on page 55. 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 47) are explained in "Interrupt Sources" on page 57.

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 4. 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 (XRES) signal on XRES pin, or Watchdog Reset (WDR). 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, CPU code execution jumps to the flash memory. Flash memory address 0x00000004 (Exception#1 in Table 6-1) stores the location of the startup code in the 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

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 the 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 you should not modify.

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 address. 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 the software has the ability to recover from the fault situation.

6.4.4 Supervisor Call (SVCall) Exception

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 SVCall. The SVC instruction enables the application to issue a SVCall that requires privileged access to the system. Note that the CM0+ CPU in PSoC 4 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 100 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 PendSupervisory (PendSV) Exception

PendSV is another SVCall 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

The CM0+ CPU in PSoC 4 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

The PSoC 4 supports up to 32 interrupts (IRQ0 to IRQ31 or exception numbers 16 – 47) from peripherals. The source of PSoC 4500S for each interrupt is listed in Table 6-2. PSoC 4 provides flexible sourcing options for each interrupt line. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM and SCB. 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 65 for details on GPIO interrupts.

Table 6-2. List of PSoC 4500S Interrupt Sources

Interrupt	CM0+ Exception No.	Interrupt Source
NMI	2	SYSCALL_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	WDT (Watchdog Timer)
IRQ7	23	SCB0 (Serial Communication Block 0)
IRQ8	24	SCB1 (Serial Communication Block 1)
IRQ9	25	SCB2 (Serial Communication Block 2)
IRQ10	26	SCB3 (Serial Communication Block 3
IRQ11	27	SCB4 (Serial Communication Block 4
IRQ12	28	CTBm0 (Continuous Time Block mini) - all CTBm0s
IRQ13	29	CTBm1 (Continuous Time Block mini) - all CTBm1s
IRQ14	30	WCO WDT Interrupt
IRQ15	31	DMA Interrupt
IRQ16	32	SPCIF Interrupt
IRQ17	33	CSD (CapSense)
IRQ18	34	TCPWM0 (Timer/Counter/PWM 0)
IRQ19	35	TCPWM1 (Timer/Counter/PWM 1)
IRQ20	36	TCPWM2 (Timer/Counter/PWM 2)
IRQ21	37	TCPWM3 (Timer/Counter/PWM 3)
IRQ22	38	TCPWM4 (Timer/Counter/PWM 4)
IRQ23	39	TCPWM5 (Timer/Counter/PWM 5)
IRQ24	40	TCPWM6 (Timer/Counter/PWM 6)
IRQ25	41	TCPWM7 (Timer/Counter/PWM 7)
IRQ26	42	SAR ADC 0
IRQ27	43	SAR ADC 1
IRQ28	44	EXCO Interrupt
IRQ29	45	Reserved
IRQ30	46	Reserved
IRQ31	47	Reserved

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 4 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 4, 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 4 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 the active exception. The CPU resumes execution of the lower-priority exception handler after servicing the higher-priority exception. The CM0+ CPU in PSoC 4 allows nesting of up to four exceptions. When the CPU receives two or more exception 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 55.

The priority of the 32 interrupts (IRQ0 to IRQ31) 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

NVIC provides registers to individually enable and disable the 32 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 '0' 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 IRQ31. 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 55.

The PRIMASK register in the CM0+ CPU can be used as a global exception enable register to mask all configurable priority exceptions regardless of whether they are enabled. Configurable priority exceptions include all 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 listed in Table 6-5.

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 exception states:

- The VECTACTIVE bits ([8:0]) in the CM0P_ICSR store the exception number for the current executing exception. This value is '0' 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 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 '0' if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0P_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ27) 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.

NVIC allows software pending of the 29 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 '0' 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 32 peripheral interrupts (exception numbers 16–47). 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 55.

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 PSR, ReturnAddress, Link Register (LR or R14), R12, R3, R2, R1, and R0. CM0+ 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, MSP is always used as the stack pointer. The stack pointer in CM0+ 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, MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the "Cortex-M0+ CPU" chapter on page 30 for details.

The CM0+ CPU uses two techniques, tail chaining and late arrival, to reduce the 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

The PSoC 4 allows device wakeup from Low-Power modes when certain peripheral interrupt requests are generated. The 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.

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 101.

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 (WDT 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 the PSoC 4:

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 bits 31:28 of the VTOR register with the value of the flash or SRAM address at which the vector table will reside. 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, 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 58.
   d. Enable the exception, as explained in “Enabling and Disabling Interrupts” on page 58.

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 CM0+ architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

7. Device Security

The PSoC 4 several features to protect 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 4500S, PSoC 4000S, PSoC 4100S, PSoC 4700S Device Programming Specifications.

Note All programming, debug, and test interfaces are disabled when maximum device security is enabled, so PSoC 4500S devices with full device security enabled may not be returned for failure analysis.

7.1 Features

The PSoC 4500S device security system provides the following device security 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 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.

7.2 How it Works

7.2.1 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, or 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 (DAP) 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**: You 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**: You 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 reprogram flash is not available. The part cannot be taken out of KILL mode; devices in KILL mode may not be returned for failure analysis.

7.2.2 Flash Security

The PSoC 4 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 the two protection levels to each row; see Table 7-1. Flash protection levels can only be changed by performing a complete flash erase.

For more details, see the Nonvolatile Memory Programming chapter on page 256.

Table 7-1. 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.

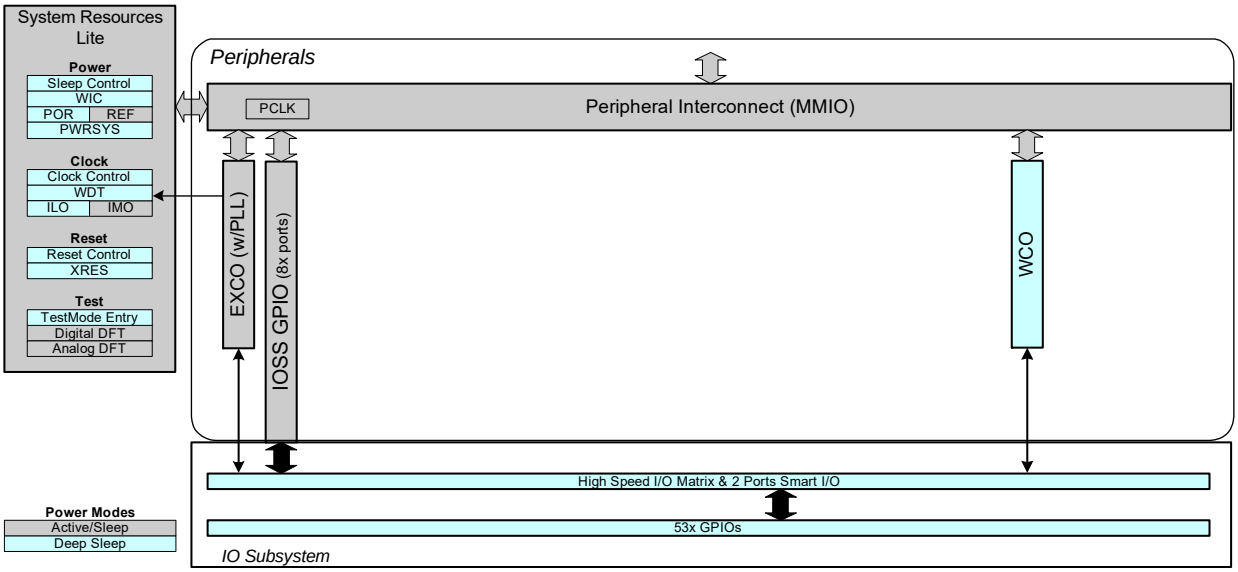
Section C: System Resources Subsystem (SRSS)

This section encompasses the following chapters:

- I/O System chapter on page 65
- Clocking System chapter on page 85
- Power Supply and Monitoring chapter on page 96
- Chip Operational Modes chapter on page 100
- Power Modes chapter on page 101
- Watchdog Timer chapter on page 105
- Trigger Multiplexer Block chapter on page 110
- Reset System chapter on page 114

Top Level Architecture

System Resource Subsystem Block Diagram



8. I/O System

This chapter explains the PSoC 4 I/O system, its features, architecture, operating modes, and interrupts. The GPIO pins in PSoC 4 are grouped into ports; a port can have a maximum of eight GPIOs. The PSoC 4500S device has a maximum of 53 GPIOs arranged in eight ports.

8.1 Features

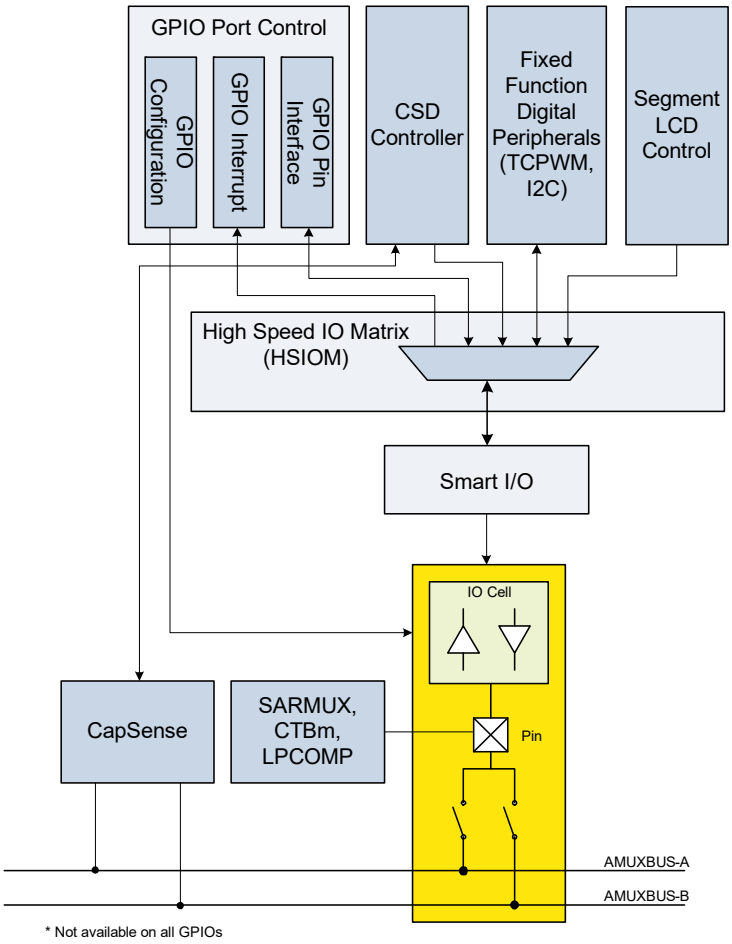
The PSoC 4 GPIOs supports these features:

- 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

8.2 GPIO Interface Overview

The PSoC 4 is equipped with analog and digital peripherals. Figure 8-1 shows an overview of the routing between the peripherals and pins.

Figure 8-1. GPIO Interface Overview

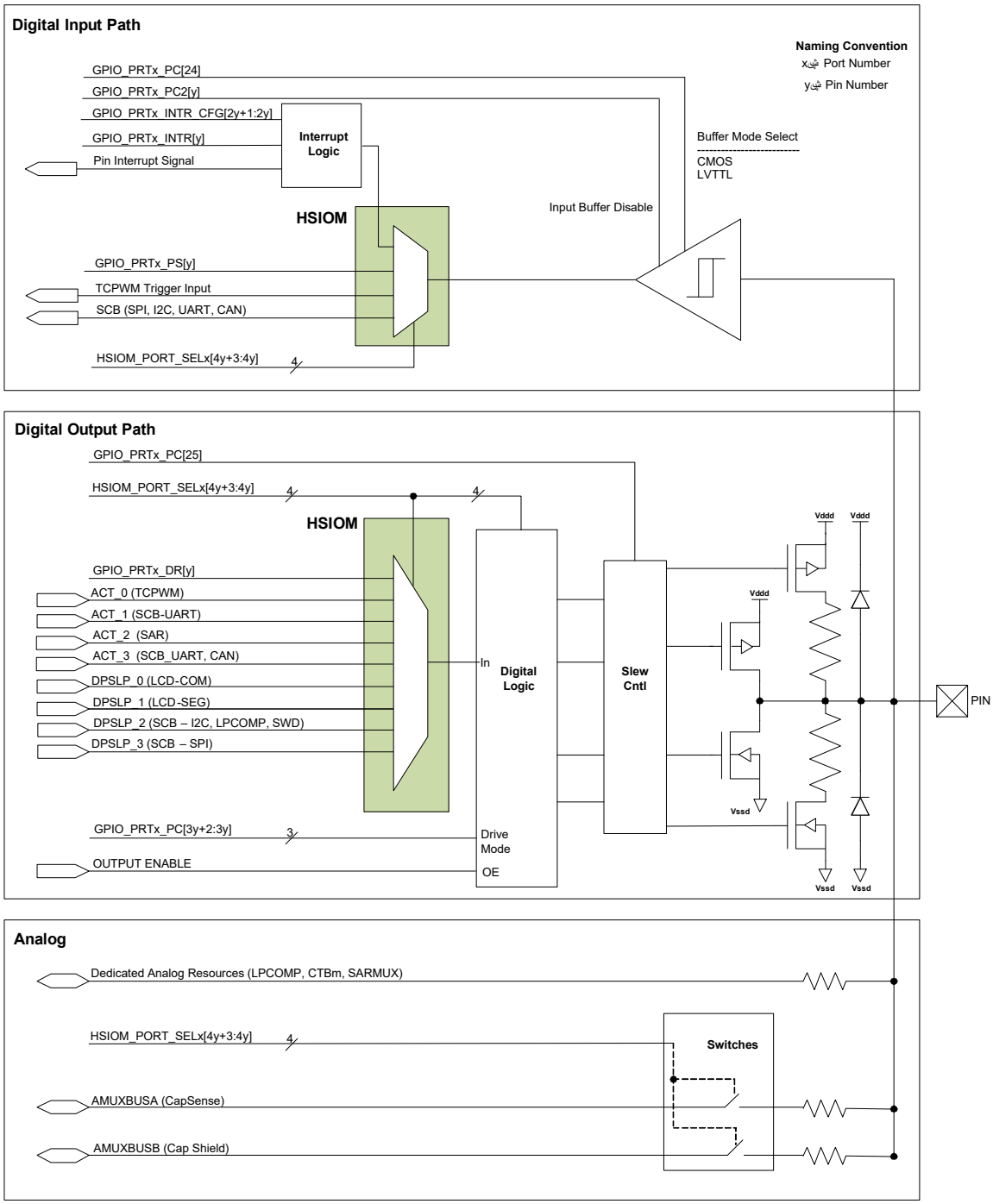


GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high input impedance and a driver for the digital output signals. The digital peripherals connect to the I/O cells via the HSIOM. HSIOM contains multiplexers that connect a peripheral you select to 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.

8.3 I/O Cell Architecture

Figure 8-2 shows the I/O cell architecture. It comprises an input buffer and an output driver. This architecture is present in every GPIO cell. The I/O cell architecture connects to the HSIOM multiplexers/Smart I/O block for the digital input and the output signal.

Figure 8-2. GPIO Block Diagram



8.3.1 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 8-1. Input Buffer Modes

PORT_VTRIP_SEL	Input Buffer Mode
0b	CMOS
1b	LVTTL

The threshold values for each mode can be obtained from the *PSoC 4500S 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 8-2, are pin-dependent. See the *PSoC 4500S datasheet* to know the functions available for each pin.

8.3.2 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 PSoC 4, I/Os are driven with the 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 *PSoC 4500S 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 "High-Speed I/O Matrix (HSIOM)" on page 70 to know about the peripheral source selection for the data and to enable or disable control source selection.

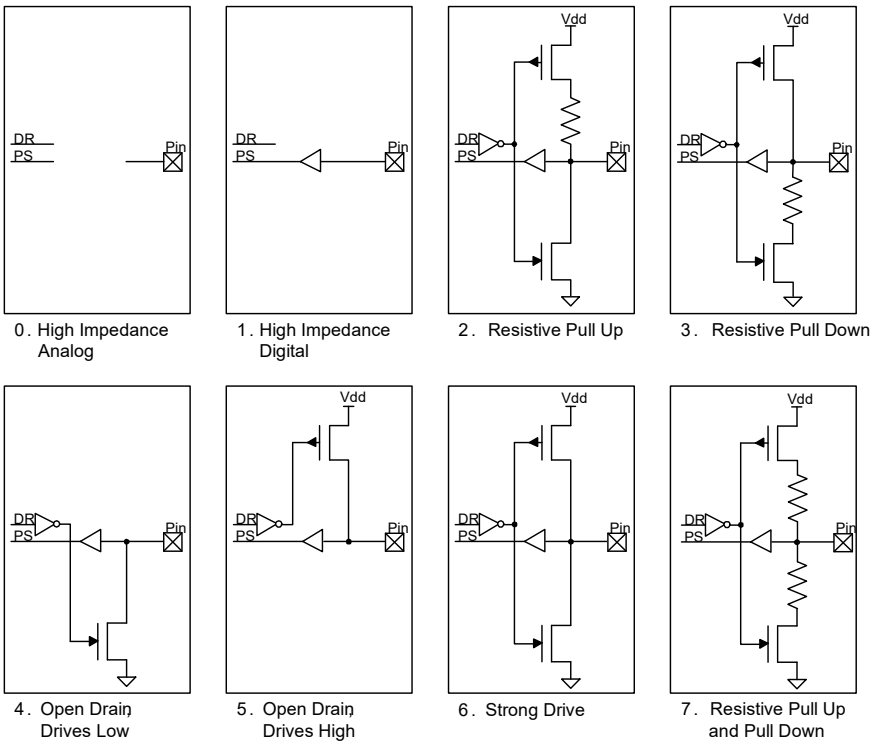
8.3.2.1 Drive Modes

Each I/O is individually configurable into one of eight drive modes using the Port Configuration register, GPIO_PRTx_PC. Table 8-2 lists the drive modes. Figure 8-2 is a simplified output driver diagram that shows the pin view based on each of the eight drive modes.

Table 8-2. 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' ($0 \le y \le 7$)		
	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 8-3. 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 the signal is externally pulled down; open drain drive low is used when the signal is externally pulled up. 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.

8.3.2.2 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.

8.4 High-Speed I/O Matrix (HSIOM)

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 you select. In PSoC 4500S, the Smart I/O block bridges the Port 2 and Port 3 pins to the HSIOM. Other ports connect directly to 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 8-3. See the example of switching a GPIO pin connection between Analog and Digital sources.

Table 8-3. PSoC 4 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 ≤ y ≤ 7))**
4y+3 : 4y	GPIO	0	The pin is regular firmware-controlled GPIO, or connected to a dedicated hardware block.
	GPIO_DSI	1	The ouput is firmware-controlled, but OE is controlled from DSI.
	DSI_DSI	2	Both output and OE are controlled from DSI.
	DSI_GPIO	3	The output is controlled from DSI, but OE is firmware-controlled.
	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 (SAR ADC).
	ACTIVE_3	11	Pin-specific Active source #3 (TCPWM Input, SCB-UART, CAN).
	DEEP_SLEEP_0	12	Reserved
	DEEP_SLEEP_1	13	Reserved
	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 *PSoC 4500S datasheet* for more details on the features supported by each pin.

8.5 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:

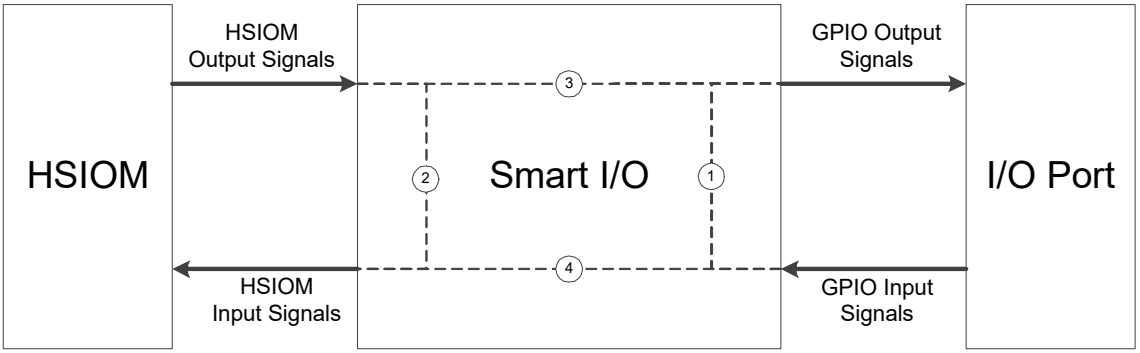
- Integrate board-level Boolean logic functionality into a port
- Ability to preprocess HSIOM input signals from the GPIO port pins
- Ability to post-process HSIOM output signals to the GPIO port pins
- Support all device power modes
- Integrate closely with the I/O pads, providing shortest signal paths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

The PSoC 4500S device supports Smart I/O on two ports – Port 2 and Port 3. The register nomenclature 'PRGIO_PRT0' denotes Port 2 Smart I/O registers and 'PRGIO_PRT1' denotes Port 3 Smart I/O registers. For a general Smart I/O register description, the 'PRGIO_PRTx' nomenclature will be used.

8.5.1 Overview

The Smart I/O block is positioned in the signal path between the HSIOM and the I/O port. HSIOM multiplexes the output signals from fixed-function peripherals and the 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 8-4.

Figure 8-4. Smart I/O Interface



The signal paths supported through the Smart I/O block, as shown in Figure 8-4, 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.
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_in) 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.

8.5.2 Block Components

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

- Clock and Reset
- Synchronizers
- Lookup Table (LUT3)
- Data Unit

8.5.2.1 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 85 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.

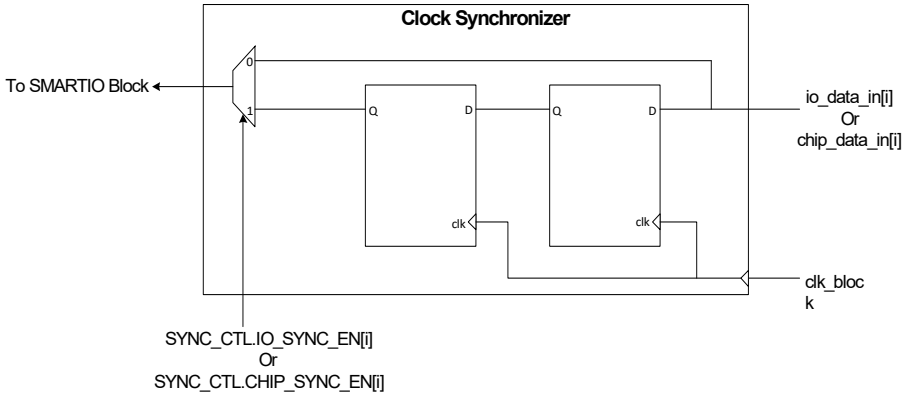
Table 8-4. Clock and Reset Register Control

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_in[0]/'1' ... "7": io_data_in[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 deactivated). To be used for asynchronous (clockless) block functionality.

8.5.2.2 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 8-5, 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 8-5. Smart I/O Clock Synchronizer



8.5.2.3 Lookup Table (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 8-6.

The 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:

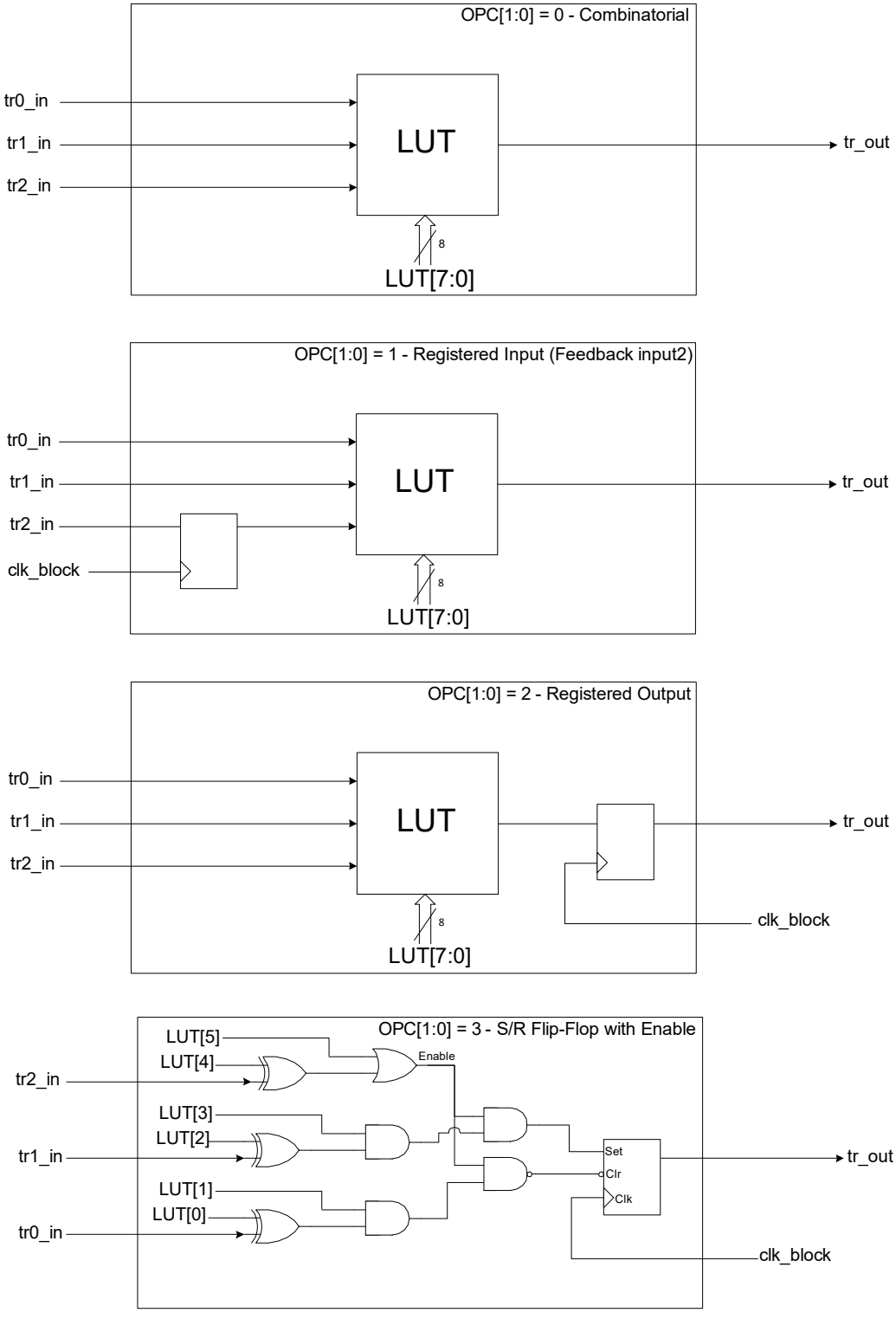
- Data unit output
- Other LUT3 output signals (tr_out)
- HSIOM output signals (chip_data[7:0])
- GPIO input signals (io_data_in[7:0])

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 8-5 for details.

Table 8-5. 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 8-6.
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_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7) "13": io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7) "14": io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7) "15": io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[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_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7) "13": io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7) "14": io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7) "15": io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[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 8-6. Smart I/O LUT3 Configuration



8.5.2.4 Data Unit (DU)

Each Smart I/O block includes a data unit (DU) component. The DU 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 DU 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 DU 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 8-6 for register control details.

Table 8-6. 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: "0": Count Up "1": Count Down "2": Count Up Wrap "3": Count Down Wrap "4": Count Up/Down "5": Count Up/Down Wrap "6": Rotate Right "7": Shift Right "8": DATA0 & DATA1 "9": Majority 3 "10": Match DATA1 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": Constant 0 "1": data[7:0] "2": gpio[7:0] "3": DU Reg
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 DU 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).

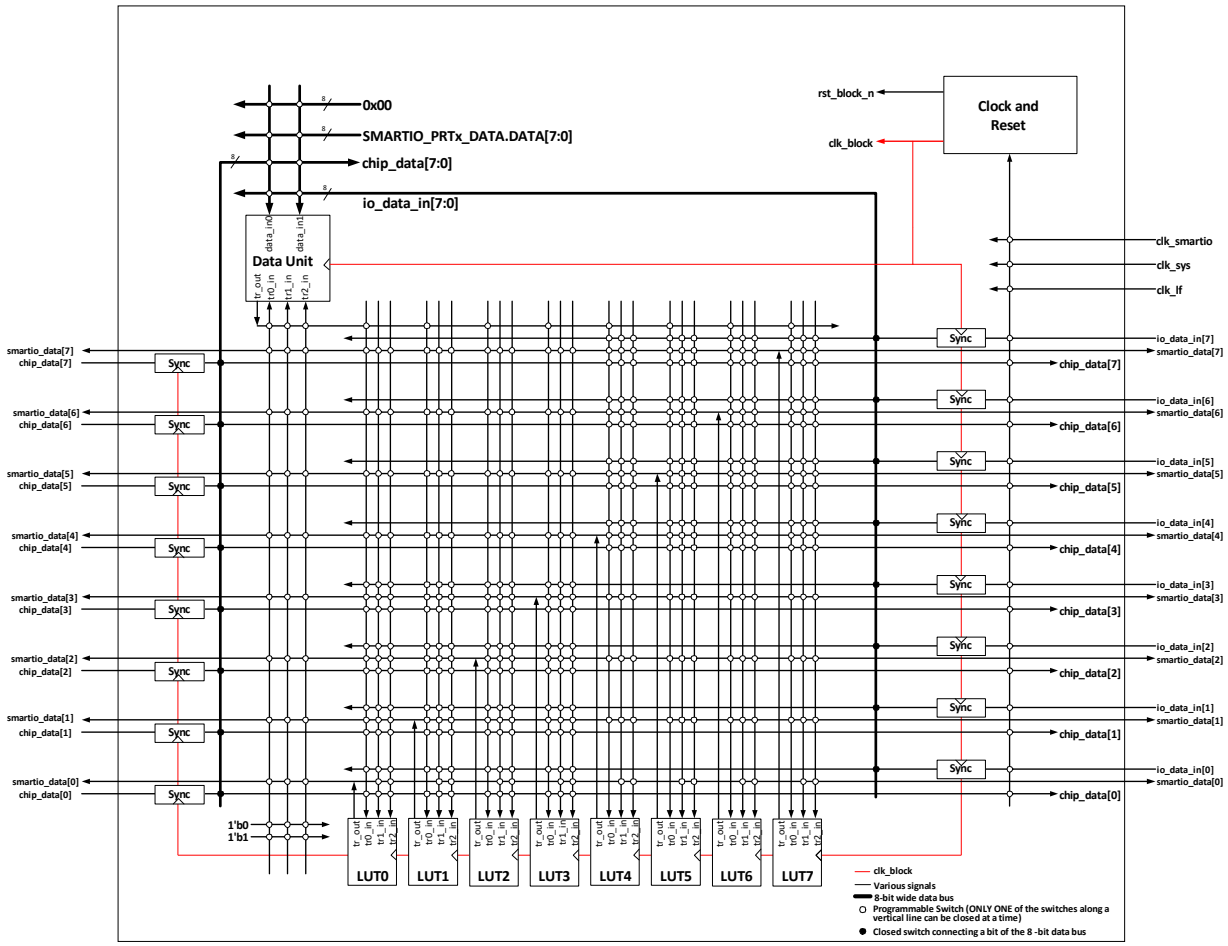
```
// The following is shared by all operations.
mask = (2 ^ (DU_SIZE+1) - 1)
data_eql_data1_in = (data & mask) == (data1_in & mask));
data_eql_0        = (data & mask) == 0);
data_incr         = (data + 1) & mask;
data_decr         = (data - 1) & mask;
data0_masked      = data_in0 & mask;

// 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;
```

8.5.3 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. See the *PSoC 4500S datasheet* for details. The Smart I/O internal routing is shown in Figure 8-7. In the figure, note that LUT7 to LUT4 operate on io_data_in/chip_data[7] to io_data_in/chip_data[4] whereas LUT3 to LUT0 operate on io_data_in/chip_data[3] to io_data_in/chip_data[0].

Figure 8-7. Smart I/O Routing



8.5.4 Operation

The Smart I/O block should be configured and operated as follows. See Table 8-7 for register control details.

1. Before enabling the block, all components should be configured and the routing should be selected, as explained in “Block Components” on page 72.
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 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 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 8-7. 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)

8.6 I/O State on Power Up

During power up all 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 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.

8.7 Behavior in Low-Power Modes

Table 8-8 shows the status of GPIOs in Low-Power modes.

Table 8-8. GPIO in Low-Power Modes

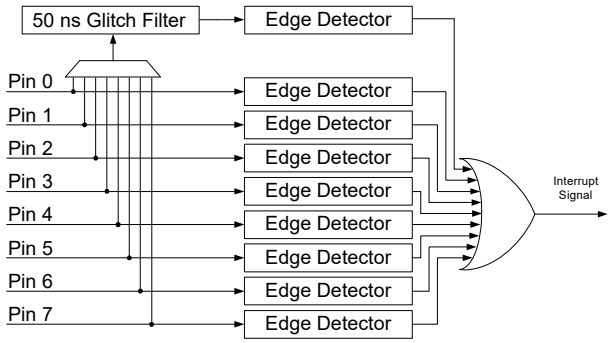
Low-Power Mode	Status
Sleep	■ GPIOs are active and can be driven by peripherals such as CapSense, CTBm, SAR ADC, TCPWM, SCBs, and LPCOMPs, which can work in sleep mode. ■ Input buffers are active; thus an interrupt on any I/O can be used to wake up the CPU. ■ AMUXBUS connections are available.
Deep Sleep	■ 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 LPCOMP can receive signals from its dedicated pins and can wake up the CPU. CTBm 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.

8.8 Interrupt

In the PSoC 4500S device, all port pins have the capability to generate interrupts. As shown in Figure 8-2, the pin signal is routed to the interrupt controller through the GPIO Edge Detect block.

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

Figure 8-8. GPIO Edge Detect Block Architecture



An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, or 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 8-9.

Table 8-9. 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 8-10.

Table 8-10. 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 8-11 shows the Port Interrupt Status register bit fields.

Table 8-11. 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 54, 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.

8.9 Peripheral Connections

8.9.1 Firmware Controlled GPIO

See Table 8-3 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 not 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.

8.9.2 Analog I/O

Analog resources, such as LPCOMP, SARMUX, and CTBm, which require low-impedance routing paths have dedicated pins. Dedicated analog pins provide direct connections to specific analog blocks. They help improve performance and should be given priority over other pins when using these analog resources. See the *PSoC 4500S 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 8-2) 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.

8.9.3 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. Refer to the LCD Direct Drive chapter on page 187 for details.

8.9.4 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 8-12 shows the GPIO and HSIOM settings required for CapSense. Refer to the CapSense chapter on page 233 for more information.

Table 8-12. 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

8.9.5 Serial Communication Blocks (SCB)

SCB, which can be configured as UART, I^2C, and SPI, has dedicated connections to the pin. See the *PSoC 4500S 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.

8.9.6 Timer/Counter/PWM (TCPWM) Block

TCPWM has dedicated connections to the pin. See the *PSoC 4500S 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.

8.10 Registers

Table 8-13. 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. Refer to the “Smart I/Oä” on page 71 for details.

9. Clocking System

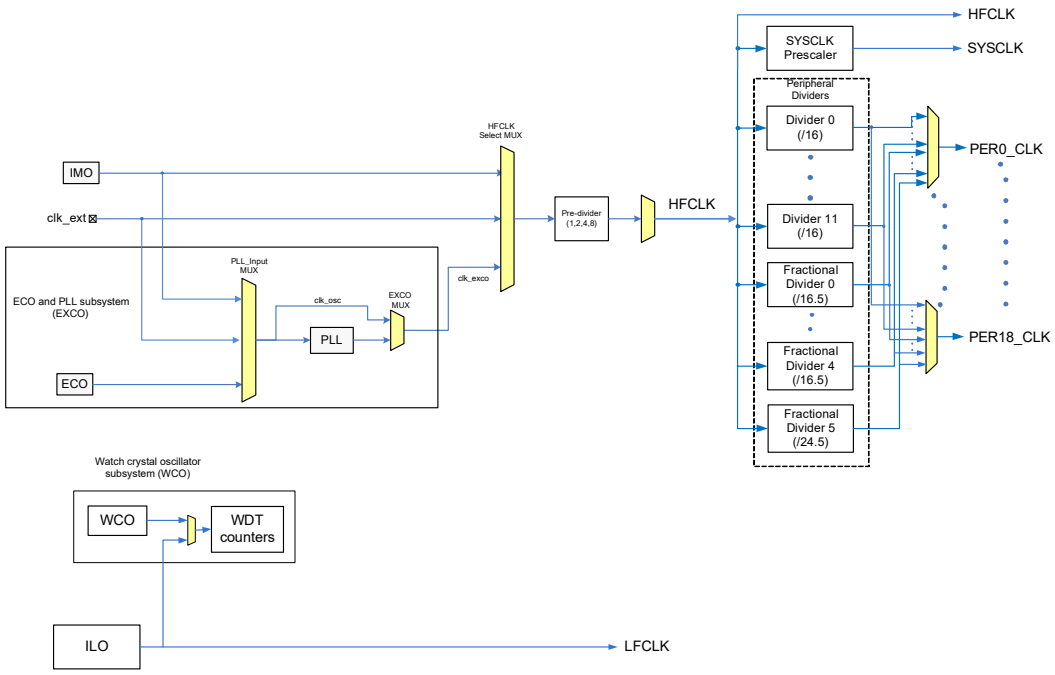
The PSoC 4 clock system includes these clock resources:

- Two internal clock sources:
  - 24–48 MHz IMO with ±2% accuracy across all frequencies with trim
  - 40-kHz ILO with -50%, +100% accuracy
- Three EXTCLK sources:
  - EXTCLK generated using a signal from an I/O pin
  - 4- to 33-MHz ECO
  - 32-kHz external WCO
- HFCLK of up to 48 MHz, selected from IMO, ECO, EXTCLK, or Phase-Locked Loop (PLL)
- LFCLK sourced by ILO or WCO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- Twelve 16-bit peripheral clock dividers
- Six fractional dividers (five 16.5 fractional dividers and one 24.5 fractional divider) for accurate clock generation
- Nineteen digital and analog peripheral clocks

9.1 Block Diagram

Figure 9-1 gives a generic view of the clocking system in PSoC 4500S devices.

Figure 9-1. Clocking System Block Diagram



The five clock sources in the device are IMO, ECO, EXTCLK, WCO, and ILO, as shown in Figure 9-1. There is a PLL that can be configured to take clock sources from the IMO, the EXTCLK, or the ECO. The PLL output frequency can be as high as 104 MHz. Although the PLL output frequency can be as high as 104 MHz, not all blocks accept frequencies higher than 48 MHz. Refer to "Phase-Locked Loop (PLL)" on page 89 for details. The HFCLK MUX selects the HFCLK source from the EXTCLK, ECO, PLL (through the EXCO MUX) or the IMO. The HFCLK frequency can be a maximum of 48 MHz.

9.2 Clock Sources

9.2.1 Internal Main Oscillator (IMO)

The 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%.

The IMO frequency is changed using the CLK_IMO_SELECT register, as shown in Table 9-1. The default frequency is 24 MHz.

Table 9-1. 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

IMO trim values are stored in Supervisory Flash (SFlash) Frequency Trim registers (SFLASH_IMO_TRIM_LTx) and TempCo Trim registers (SFLASH_IMO_TCTRIM_LTx).

The registers in Frequency Trim registers or TempCo Trim registers are in groups of four. For example, SFLASH_IMO_TRM_LT1 ~ SFLASH_IMO_TRM_LT4 is a group, SFLASH_IMO_TRM_LT5 ~ SFLASH_IMO_TRM_LT8 is a group … SFLASH_IMO_TRM_LT21 ~ SFLASH_IMO_TRM_LT24 is a group. In each group, only one in four addresses are updated. The remaining three are just pasted with the exact same data (for example, if SFLASH_IMO_TRM_LT1 is written with 0x80 data, then the same is copied over to SFLASH_IMO_TRM_LT2, SFLASH_IMO_TRM_LT3 and SFLASH_IMO_TRM_LT4). Similarly, SFLASH_IMO_TCTRIM_LT registers have the same usage as SFLASH_IMO_TRM_LT registers.

Each IMO clock frequency which selected in CLK_IMO_SELECT[2:0] corresponds with one SFlash trim value register.

For example:

24 MHZ SFLASH_IMO_TRIM_LT0 & SFLASH_IMO_TCTRIM_LT0

28 MHZ SFLASH_IMO_TRIM_LT4 & SFLASH_IMO_TCTRIM_LT4

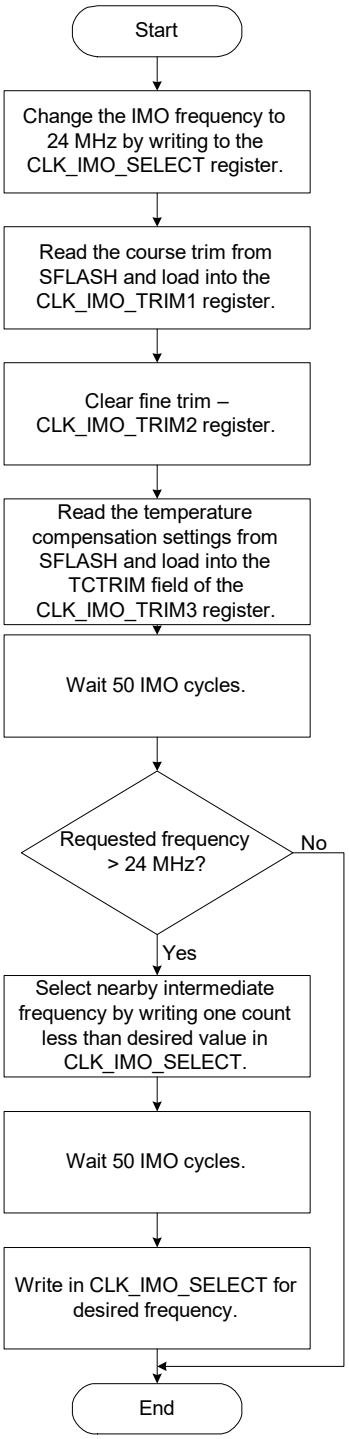
….

48MHZ SFLASH_IMO_TRIM_LT24 & SFLASH_IMO_TCTRIM_LT24

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 during run time. Follow the algorithm in Figure 9-2 to change the IMO frequency.

Figure 9-2. Change IMO Frequency



9.2.1.1 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.

9.2.1.2 Programming Clock (36 MHz)

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.

9.2.2 Internal Low-speed Oscillator (ILO)

The ILO 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, HFCLK to improve accuracy. The ILO is available in all power modes. The ILO is used as the system low-frequency clock LFCLK in the device. The ILO is a relatively inaccurate (-50, +100% over voltage 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% for stable temperature and voltage. The ILO is enabled and disabled with register CLK_ILO_CONFIG bit ENABLE.

9.2.3 External Clock (EXTCLK)

The external clock is a MHz-range clock that can be generated from a signal on a designated PSoC 4500S pin (P0.6). This clock may be used instead of the IMO as the source of the system HFCLK. The allowable range of EXTCLK frequencies is 1–48 MHz. The device always starts up using the IMO and the EXTCLK must be enabled in User mode; therefore, the device cannot be started from a reset, which is clocked by the EXTCLK.

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. Refer to the I/O System chapter on page 65 for more details.

9.2.4 External Crystal Oscillator (ECO)

The PSoC 4 device contains an oscillator to drive an external 4-MHz to 33.33-MHz crystal. This clock source is built using an oscillator circuit in PSoC. The circuit employs an external crystal that needs to be populated on the external crystal pins of the PSoC.

The ECO can be enabled by using the EXCO_ECO_CONFIG.CLK_EN (bit 0) and EXCO_ECO_CONFIG.ENABLE (bit 31) register bit fields.

9.2.4.1 ECO Trimming

The ECO supports a wide variety of crystals and ceramic resonators with the nominal frequency range specification of f = 4 MHz – 33.333 MHz. The crystal manufacturer typically provides numerical values for parameters, namely the maximum drive level (DL), the equivalent series resistance (ESR), and the parallel load capacitance (C_L). These parameters can be used to calculate the transconductance (g_m) and the maximum peak-to-peak value (V_{PP}).

Maximum peak to peak value:

$$V_{PP} = 2 \times \frac{\sqrt{\frac{2 \times DL}{ESR}}}{4\pi \times f \times C_L}$$

Transconductance:

$$g_m = 4 \times 5 \times (2\pi \times f \times C_L)^2 \times ESR$$

V_{PP} < 0.4 V is not supported by the ECO. Similarly, g_m cannot be greater than or equal to 18 mA/V.

The ATRIM and WDTRIM settings control the trim for amplitude of the oscillator output. Amplitude trim (ATRIM) sets the crystal drive level when AGC is enabled (EXCO_ECO_CONFIG.AGC_EN=1). AGC must be enabled for V_{PP} < 2 V and disabled for all other cases. Based on the V_{PP} value, the ATRIM and WDTRIM values are set as shown in Table 9-2.

Table 9-2. ATRIM and WDTRIM Setting

VPP	ATRIM	WDTRIM
0.4 V ≤ V_{PP} < 0.5 V	0x00	0x00
V_{PP} < 0.6 V	0x01	0x00
V_{PP} < 0.7 V	0x02	0x01
V_{PP} < 0.8 V	0x03	0x01
V_{PP} < 0.9 V	0x04	0x02
V_{PP} < 1.025 V	0x05	0x02
V_{PP} < 1.15 V	0x06	0x03
V_{PP} < 1.275 V	0x07	0x03

The GTRIM sets up the trim for amplifier gain and is set based on the g_m calculated as shown in Table 9-3.

Table 9-3. GTRIM Setting

G_m	GTRIM
0 mA/V < g_m ≤ 4.5 mA/V	0x00
4.5 mA/V < g_m ≤ 9 mA/V	0x01
9 mA/V < g_m ≤ 13.5 mA/V	0x02
13.5 mA/V < g_m ≤ 18 mA/V	0x03

The FTRIM and RTRIM sets up the trim for the filter characteristics and is set based on the g_m calculated as shown in Table 9-4.

Table 9-4. FTRIM and RTRIM Setting

Nominal Frequency f (MHz)	RTRIM	FTRIM
f > 30 MHz	0x00	0x00
30 MHz ≥ f > 24 MHz	0x01	0x01
24 MHz ≥ f > 17 MHz	0x02	0x02
17 MHz ≥ f	0x03	0x03

9.2.5 Phase-Locked Loop (PLL)

The PSoC device implements a PLL. The PLL provides highly configurable frequency synthesis. The PLL is configured primarily using two settings P and Q. The P is the feedback divider setting for the PLL and the Q is the reference clock divider. Hence, the output frequency of the PLL is P/Q times the reference clock.

P and Q for the PLL are configured by EXCO_PLL_CONFIG. In this register, EXCO_PLL_CONFIG[7:0] is the P value and EXCO_PLL_CONFIG[13:8] is the Q value. By varying this integer value for the feedback counter (P has eight bits of resolution) and the reference counter (Q has six bits of resolution), the PLL can synthesize a large number of output frequencies from an input reference clock frequency (clk_ref).

9.2.6 Watch Crystal Oscillator (WCO)

The PSoC device contains an oscillator to drive a 32.768-kHz watch crystal. Similar to ILO, the 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.

The WCO can be forced into a 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. Therefore, Auto mode is not suggested for high-accuracy applications such as real-time clock (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. 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%. 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.

9.3 Clock Supervision

In PSoC 4500S device, ECO and PLL subsystem can be called block. The ECO and PLL Subsystem (EXCO) block can supervise its output clock (clk_exco). There are mainly two blocks (CSV and PGM_DELAY) to implement this feature.

9.3.1 Clock Supervision Block (CSV)

The PSoC 4500S device has one clock supervision block, which uses a reference clock (IMO clock) to check that a monitored clock (clk_exco) is within an allowed frequency window. The CSV can be enabled by using the EXCO_REF_CTL.CSV_EN (bit 31) register bit fields.

If the monitored clock fails, the CSV block can switch the source of clk_exco to IMO clock by setting the EXCO_REF_CTL.CSV_CLK_SW_EN bit. If setting the EXCO_REF_CTL.CSV_INT_EN bit or EXCO_REF_CTL.CSV_TRIG_EN bit, switching the clk_exco source clock event can also assert the interrupt and trigger outputs. While CSV block will continue to monitor the output clock, it will not assert the interrupt or trigger outputs again until the original event has been cleared by FW.

9.3.2 Clock Supervision Interrupt (CSI)

Enable or disable the CSV interrupt by setting or clearing the CSV_CLK_SW, WD_ERR, and PLL_LOCK bits in the EXCO_INTR_MASK register. FW can get specific interrupt information from the EXCO_INTR_MASKED register.

9.3.3 Programmable Delay Block (PGM_DELAY)

For the CSV block clock switch over to be successful, it is essential that the failing clock continues to produce clock cycles. If on the other hand, the clock suddenly stops completely the switch over mechanism will fail. To address this type of failure, PSoC 4500S device has one programmable delay (PGM_DELAY) block.

PGM_DELAY block is a down counter clocked by IMO clock. The starting value is 16-bit programmable. PGM_DELAY block can be enabled and configured using the EXCO_RSTDLY_CTL.EN bit and EXCO_RSTDLY_CTL.LOAD bit.

On reaching zero, the block asserts a reset request signal to the SRSS block. It is expected that in normal circumstances, FW will intervene before the counter reaches zero so that the reset can be avoided. PGM_DELAY initial value can be set by the EXCO_RSTDLY register.

9.4 Clock Distribution

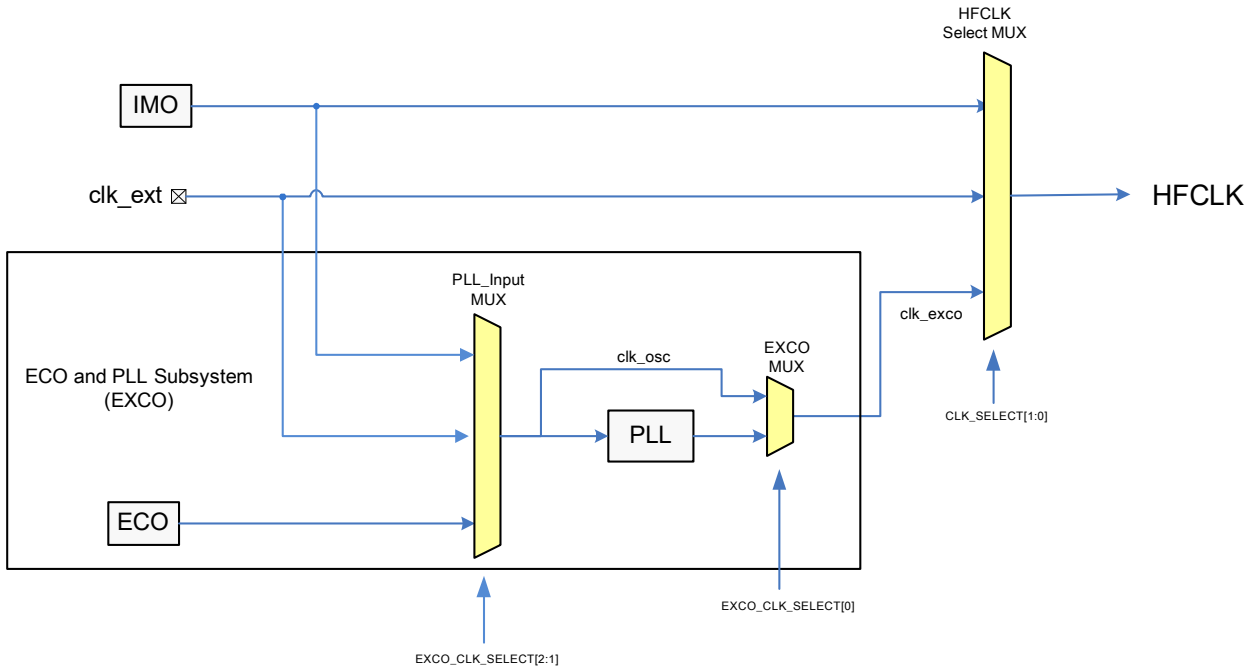
The PSoC 4500S clocks are developed and distributed throughout the device, as shown in Figure 9-1. The distribution configuration options are as follows:

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

9.4.1 HFCLK and PLL Input Selection

Figure 9-3 shows the selection options for HFCLK and PLL.

Figure 9-3. HFCLK and PLL Selection Options



The PLL input has a multiplexer that selects from IMO, ECO, and EXTCLK output signals. The input selection for the PLL is in the EXCO_CLK_SELECT[2:1] register.

For HFCLK, it is a two-stage selection. The two sources clk_osc and PLL are first multiplexed using an EXCO multiplexer whose selection is configured in EXCO_CLK_SELECT[0]. The second multiplexer, called the HFCLK multiplexer, selects between the EXCO output, IMO, and EXTCLK. This selection is configured in the CLK_SELECT[1:0] register.

Toggle the EXCO_EXCO_PGM_CLK.CLK_ECO bit five times by alternatively writing a ‘0’ and ‘1’ to it after setting the EXCO_PGM_CLK.ENABLE bit. After the toggle, clear the EXCO_EXCO_PGM_CLK.ENABLE bit by writing a ‘0’ to start the PLL.

Low Power Operation: Note that this procedure should be used when the device exits the Deep Sleep power mode. The system clock (HFCLK/SYSCLK) source should be changed from PLL to IMO before the devices enter Deep Sleep mode. After waking up from the Deep Sleep mode, the system clock source can be changed back from IMO to the PLL.

9.4.2 High-Frequency Clock (HFCLK) Input Selection

The High-Frequency Clock (HFCLK) in PSoC 4500S has four input options: IMO, ECO, PLL, and EXTCLK. The clk_osc and PLL are first selected in a stage to get the ECO and PLL subsystem (EXCO) clock. This input for EXCO MUX is selected using the EXCO_CLK_SELECT register CLK_SELECT bit, as described in Table 9-6. The HFCLK input is selected using the CLK_SELECT register's HFCLK_SEL bits, as described in Table 9-5.

Table 9-5. HFCLK Input Selection Bits HFCLK_SEL

Name	Description
HFCLK_SEL[1:0]	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: EXCO: Uses the ECO and PLL subsystem (EXCO) output as the source of the HFCLK

Table 9-6. CLK_SELECT Input Selection Bit

Name	Description
CLK_SELECT	ECO and PLL subsystem output clock selection 0: clk_osc. Uses the clk_osc as the source of the HFCLK 1: PLL. Uses the PLL output as the source of the HFCLK

Pre-divider is provided for HFCLK to limit the peak current of the device. The divider options are 1, 2, 4 and 8, are configured using HFCLK_DIV bits of the CLK_SELECT register. The default divider is 4.

9.4.3 Low-Frequency Clock (LFCLK) Input Selection

Both WCO and ILO can be the source for LFCLK in the PSoC 4500S device.

9.4.4 System Clock (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 2^3 = 8. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in Table 9-7. The prescaler is initially configured to divide by 1.

Table 9-7. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

Name	Description
SYSCLK_DIV[1:0]	SYSCLK prescaler divide value 0: SYSCLK = HFCLK 1: SYSCLK = HFCLK/2 2: SYSCLK = HFCLK/4 3: SYSCLK = HFCLK/8

9.4.5 Peripheral Clock Divider Configuration

The PSoC 4 has eighteen clock dividers, which include twelve 16-bit clock dividers, five 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 divide value for each of the 12 integer clock dividers are configured with the PERI_DIV_16_CTLx registers and the five 16.5-bit fractional clock dividers are configured with the PERI_DIV_16_5_CTLx registers. Table 9-8 and Table 9-9 describe the configurations for these registers. The 24.5-bit fractional divider is configured using the PERI_DIV_24_5_CTL register. Table 9-10 describes the configuration for these registers.

Table 9-8. Non-Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_CTLx

Bits	Name	Description
0	ENABLE_x	Divider enabled. HW sets this field to '1' as a result of an ENABLE command. HW sets this field to '0' as a result on a DISABLE command.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65536].

Table 9-9. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_5_CTLx

Bits	Name	Description
0	ENABLE_x	Divider enabled. HW sets this field to '1' as a result of an ENABLE command. HW sets this field to '0' as a result on a DISABLE command.
7:3	FRAC5_DIV_x	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.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65,536].

Table 9-10. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_24_5_CTL

Bits	Name	Description
0	EN	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	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	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, ‘2’ for the 16.5-bit fractional divider, and ‘3’ for the 24.5-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. PA_SEL_DIV specifies the divider which is phase aligned. Any enabled divider can be used as a reference. 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 9-11 shows the mapping of the MUX output to the corresponding peripheral blocks (shown in Figure 9-1). Any of the peripheral clock dividers can be mapped to a specific peripheral by using their respective PERI_PCLK_CTLx register.

Table 9-11. Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
0	SCB0
1	SCB1
2	SCB2
3	SCB3
4	SCB4
5	CSD
6	TCPWM0
7	TCPWM1
8	TCPWM2
9	TCPWM3
10	TCPWM4
11	TCPWM5
12	TCPWM6
13	TCPWM7
14	SmartIO Port2
15	SmartIO Port3
16	SAR ADC0
17	LCD
18	SAR ADC1

Table 9-12. Programmable Clock Control Register - PERI_PCLK_CTLx

Bits	Name	Description
5:0	SEL_DIV	Specifies one of the dividers of the divider type specified by SEL_TYPE. If SEL_DIV is ‘4’ and SEL_TYPE is ‘1’, then the fifth (zero being first) 16-bit clock divider will be routed to the MUX output for peripheral clock_x. Similarly, if SEL_DIV is ‘0’ and SEL_TYPE is ‘2’, then the first 16.5 clock divider will be routed to the MUX output.
7:6	SEL_TYPE	0: Do not use 1: 16.0 (integer) clock dividers 2: 16.5 (fractional) clock dividers 3: 24.5 (fractional) clock dividers

9.5 Low-Power Mode Operation

The HFCLKs including the IMO, EXTCLK, ECO, HFCLK, SYSCLK, PLL, and peripheral clocks, operate only in Active and Sleep modes. The ILO, WCO, and LFCLK operate in all power modes.

9.6 Register List

Table 9-13. Clocking System Register List

Register Name	Description
CLK_IMO_TRIM1	IMO Trim Register - This register contains IMO trim for coarse 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 coarse 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 and selects different sources for the system clocks.
EXCO_CLK_SELECT	Clock select for ECO and PLL subsystem - This register controls selecting the PLL input and output of the ECO and PLL subsystem.
EXCO_ECO_CONFIG	This register controls the ECO configuration.
EXCO_PLL_CONFIG	This register controls the PLL configuration.
WCO_CONFIG	WCO Enable. This register enables or disables the external WCO.
EXCO_REF_CTL	This register controls clock supervision for a clock tree.
EXCO_REF_LIMIT	Clock Supervision Reference Limits.
EXCO_MON_CTL	Clock Supervision Monitor Control.
EXCO_INTR	Interrupt Request Register.
EXCO_INTR_SET	Interrupt Set Register.
EXCO_INTR_MASK	Interrupt Mask Register.
EXCO_INTR_MASKED	Interrupt Masked Register.
EXCO_RSTDLY_CTL	Programmable Delay Counter Control.
EXCO_RSTDLY	Programmable Delay Counter Initial Amount.
EXCO_RSTDLY_COUNT_VAL	Programmable Delay Counter Value.
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_CTL	Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting the fractional divide value and enabling or disabling the divider.

10. Power Supply and Monitoring

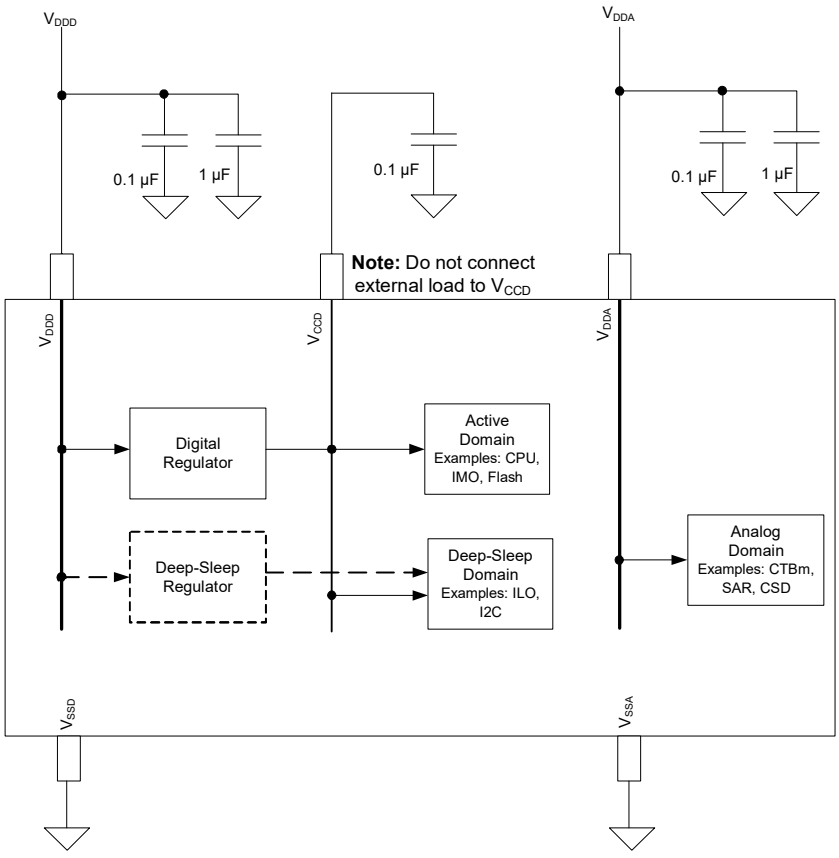
The PSoC 4 is capable of operating from an externally supplied voltage of 1.71 V to 5.5 V. This is supported through one of the following operating ranges:

- 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.

10.1 Block Diagram

Figure 10-1. Power System Block Diagram



1. When the system supply is in the range of 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.

Figure 10-1 shows the power system diagram and 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 from 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.

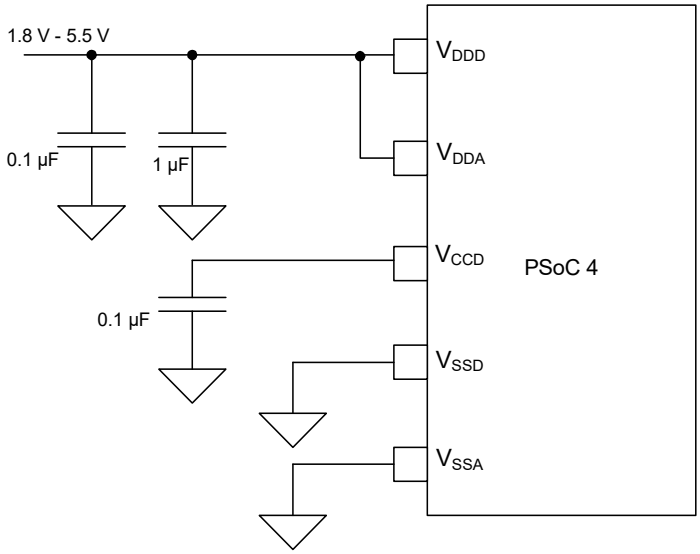
10.2 Power Supply Scenarios

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

10.2.1 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 10-2.

Figure 10-2. Single Regulated V_{DDD} Supply



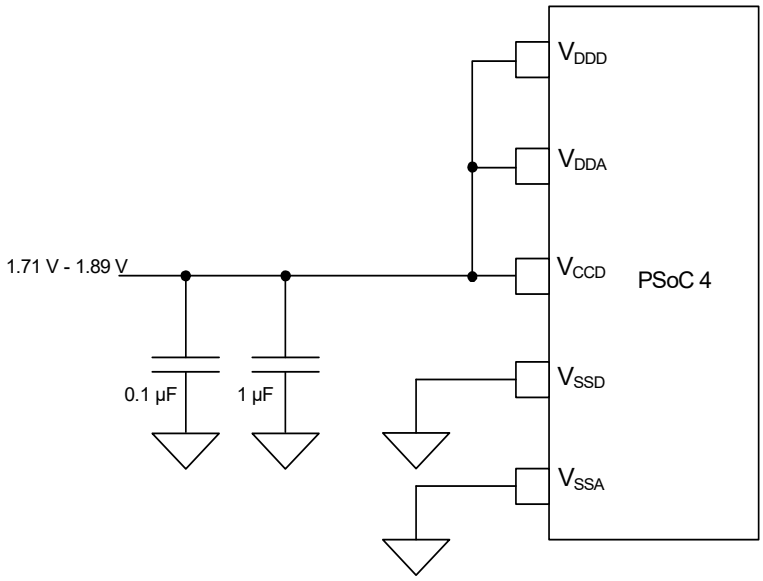
In this mode, the device is powered by an external power supply that can be anywhere in the range of 1.8 V to 5.5 V. This range is also designed for battery-powered operation; for instance, the chip can be powered from a battery system that starts at 3.5 V and works down to 1.8 V. In this mode, the internal 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, for critical applications, the PCB layout, lead inductance, and the bypass capacitor parasitic should be simulated to design and obtain optimal bypassing.

10.2.2 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 10-3.

Figure 10-3. Single Unregulated V_{DDD} Supply



In this mode, VCCD and VDDD pins are shorted together and bypassed. The internal regulator should be disabled in firmware. Refer to the "Active Digital Regulator" on page 98 for details.

10.3 How it Works

The regulators in Figure 10-1 power the various domains of the device. All core regulators draw their input power from the VDDD pin supply. The analog circuits run directly from the V_{DDA} input.

10.3.1 Regulator Summary

10.3.1.1 Active Digital Regulator

Table 10-1. 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 (VCCD) and requires an external decoupling capacitor (1 µF X5R).

For supplies below 1.8 V, V_{CCD} must be supplied directly. In this case, VCCD and VDDD must be shorted together, as shown in Figure 10-3.

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.

10.3.1.2 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 LPCOMP. 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}).

10.4 Voltage Monitoring

The voltage monitoring system includes POR and Brownout Detect (BOD).

10.4.1 Power-On Reset (POR)

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor the 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.

10.4.1.1 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 *PSoC 4500S 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 WDT should be used in all designs. WDT provides protection against abnormal brownout conditions that may compromise the CPU functionality. Refer to the Watchdog Timer chapter on page 105 for more details.

10.5 Register List

Table 10-2. 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.

11. Chip Operational Modes

The PSoC 4 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.

The PSoC 4 operational modes are:

- Boot
- User
- Privileged
- Debug

11.1 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.

11.2 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 you. 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.

11.3 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 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 258. Exit from this mode returns the device to User mode.

11.4 Debug

Debug mode is an operational mode that allows observation of the PSoC 4 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 249.

For more details on protection modes, see the Device Security chapter on page 62.

12. Power Modes

The PSoC 4 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

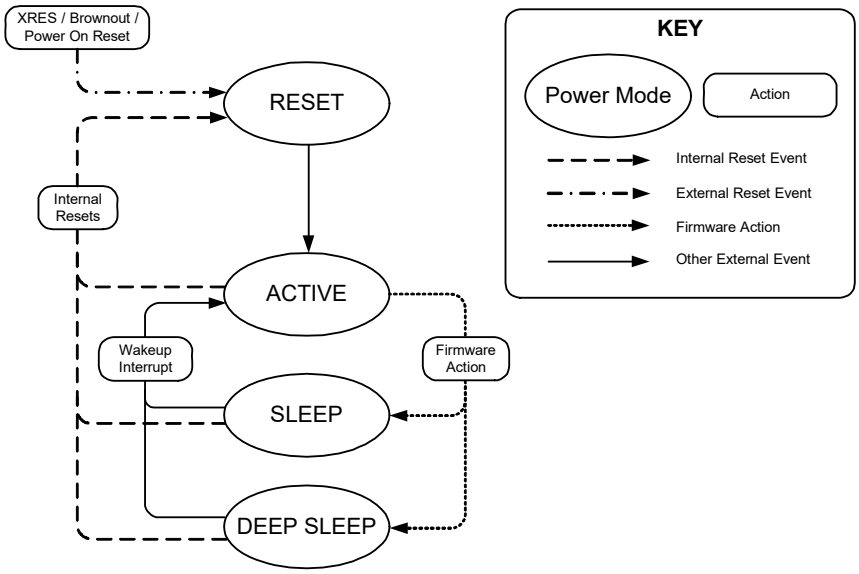
Active, Sleep, and Deep Sleep are standard Arm-defined power modes, supported by the Arm CPUs.

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 PSoC 4

Figure 12-1 illustrates various power modes and the possible transitions between them.

Figure 12-1. Power Mode Transitions State Diagram



Note: Arm nomenclature for Deep Sleep power mode is ‘SLEEPDEEP’.

Table 12-1 illustrates the power modes offered by PSoC 4.

Table 12-1. PSoC 4 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, POR	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, LPCOMP, SCB, WDT	ILO (40 kHz), WCO (32 kHz)	Interrupt	Deep Sleep regulator

In addition to the wakeup sources mentioned in Table 12-1, XRES and BOD bring the device to Active mode from any power mode.

12.1 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 peripherals are powered. The firmware may be configured to disable specific peripherals that are not in use, to reduce power consumption.

12.2 Sleep Mode

This is a CPU-centric power mode. In this mode, the CM0+ 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.

12.3 Deep Sleep Mode

In Deep Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The HFCLKs 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. CTBm can also operate in this mode with reduced power and bandwidth. For details on power consumption and CTBm bandwidth, see the *PSoC 4500S datasheet*.

12.4 Power Mode Summary

Table 12-2 illustrates the peripherals available in each Low-Power mode; Table 12-3 lists the available wakeup sources in each power mode.

Table 12-2. 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)
IMO	Available	Available	Not Available
ILO, 40 kHz	Available	Available	Available (optional)
Asynchronous peripherals (peripherals that do not run on internal clock)	Available	Available	Available
POR, BOD	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 12-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any enabled interrupt source	Interrupt
	Any reset source	Reset
Deep Sleep	GPIO interrupt	Interrupt
	I2C address match	Interrupt
	WDT	Interrupt / Reset
	LPCOMP	Interrupt
	CTBm	Interrupt

Note: In addition to the wakeup sources mentioned in Table 12-3, XRES and BOD bring the device to Active mode from any power mode. XRES and brownout trigger a full system restart. All states including frozen GPIOs are lost. In this case, the cause of wakeup is not readable after the device restarts.

12.5 Low-Power Mode Entry and Exit

WFI instruction from the CM0+ triggers the transitions into Sleep and Deep Sleep mode. CM0+ 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 CM0P System Control Register (CM0P_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 4 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, peripherals are optional (see Table 12-4), 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 52 and the Reset System chapter on page 114 for details.

12.6 Register List

Table 12-4. Power Mode Register List

Register Name	Description
CM0P_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.

13. 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.

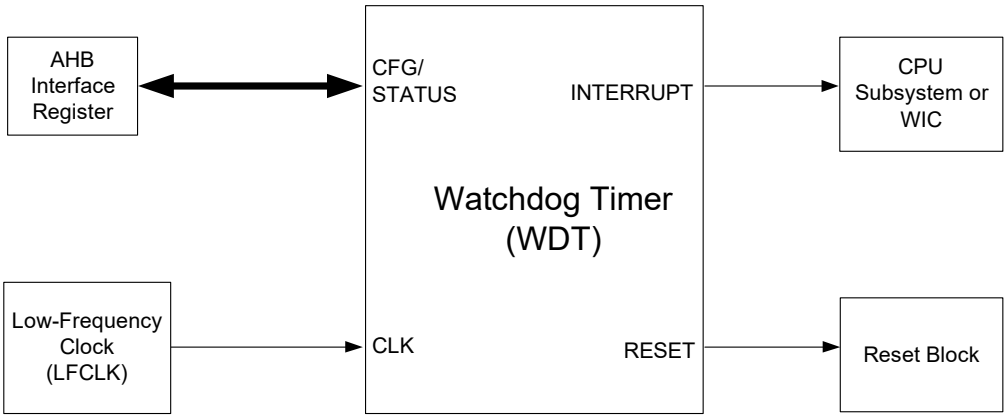
13.1 Features

The WDT supports 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

13.2 Block Diagram

Figure 13-1. WDT Block Diagram



13.3 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 Most Significant bits (MSb) 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 *PSoC 4500S: PSoC 4 Registers TRM*.

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 WDR 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 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.
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 52.

13.3.1 Enabling and Disabling Watchdog Timer

The watchdog counter is a free-running counter that cannot be disabled. However, it is possible to disable the WDR by writing a key 0xACED8865 to the WDT_DISABLE_KEY register. Writing any other value to this register will enable the WDR. 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 WDR must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDR 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.

13.3.2 Watchdog Interrupts and Low-Power Modes

The watchdog counter can send interrupt requests to the CPU in Active power mode and to the 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 101.

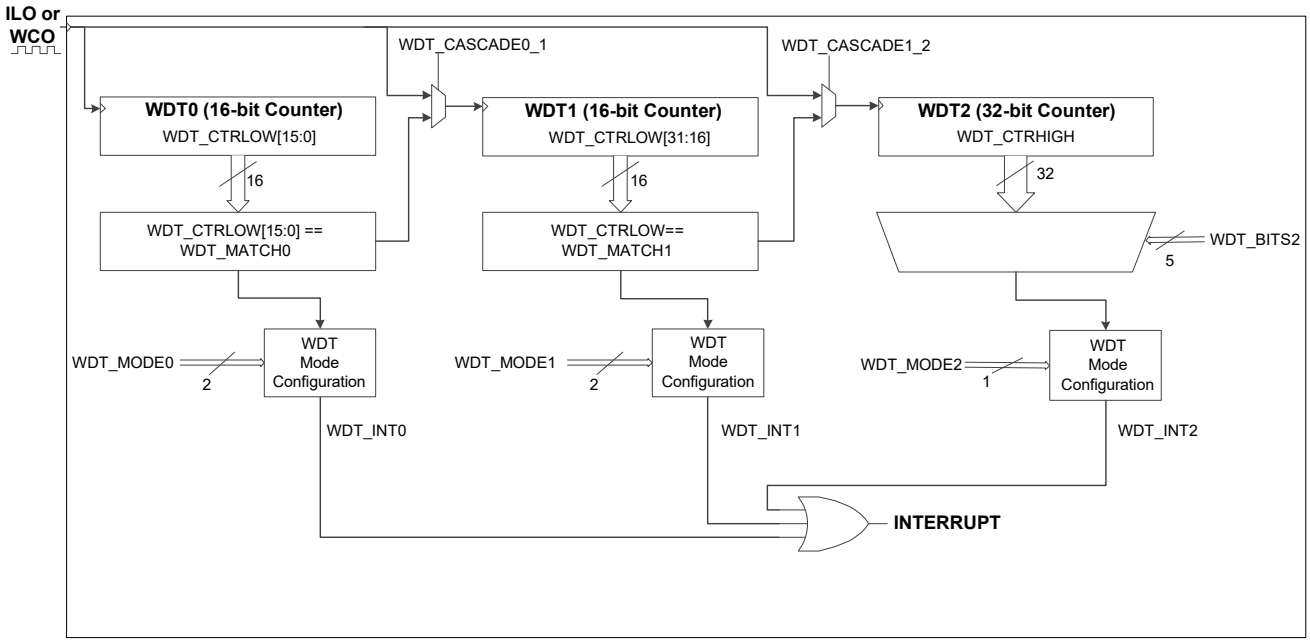
13.3.3 Watchdog Timer Reset Mode

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 POR, BOD, or XRES occurs. All other resets leave this bit untouched. For more details, see the Reset System chapter on page 114.

13.4 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 13-2. WDT Additional Timers Block Diagram



13.4.1 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.

13.4.2 WDT2

WDT2 is similar to WDT0 and WDT1 with following differences:

- WDT2 is a 32-bit up-counting timer
- WDT2 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.

13.4.3 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.

13.5 Register List

Table 13-1 provides the register control details.

Table 13-1. 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	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	Enables and resets the timer.
WCO_WDT_CLKEN	Enables the clock (ILO/WCO) to be used with the timer.

14. Trigger Multiplexer Block

Select peripherals in the PSoC 4 are interconnected using trigger signals. Trigger signals are means by which peripherals denote an occurrence of an event or a state. These triggers are used as means to affect or initiate some action in other peripherals. The trigger multiplexer block helps to route triggers from a source peripheral block to a destination.

14.1 Features

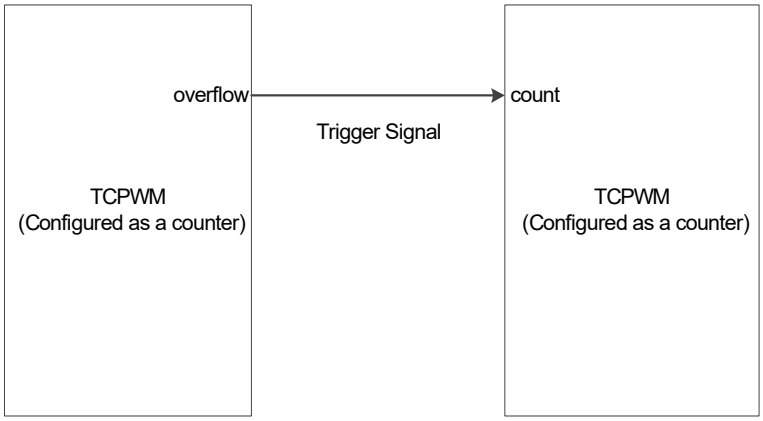
The Trigger Multiplex Block supports these features:

- Ability to connect trigger signals from one peripheral to another
- Supports a software trigger, which can trigger signals in the block
- Supports multiplexing of triggers between peripherals

14.2 Architecture

The trigger signals in the PSoC 4 are digital signals generated by peripheral blocks to denote a state such as TCPWM overflow, or an event such as the completion of an action. These trigger signals typically serve as initiator of other actions in other peripheral blocks. An example is chaining two TCPWMs together in order to make a 32-bit counter instead of a 16-bit counter. This can be done by using a counter overflow trigger to then trigger a second TCPWM's count input as shown in Figure 14-1.

Figure 14-1. Trigger Signal Example



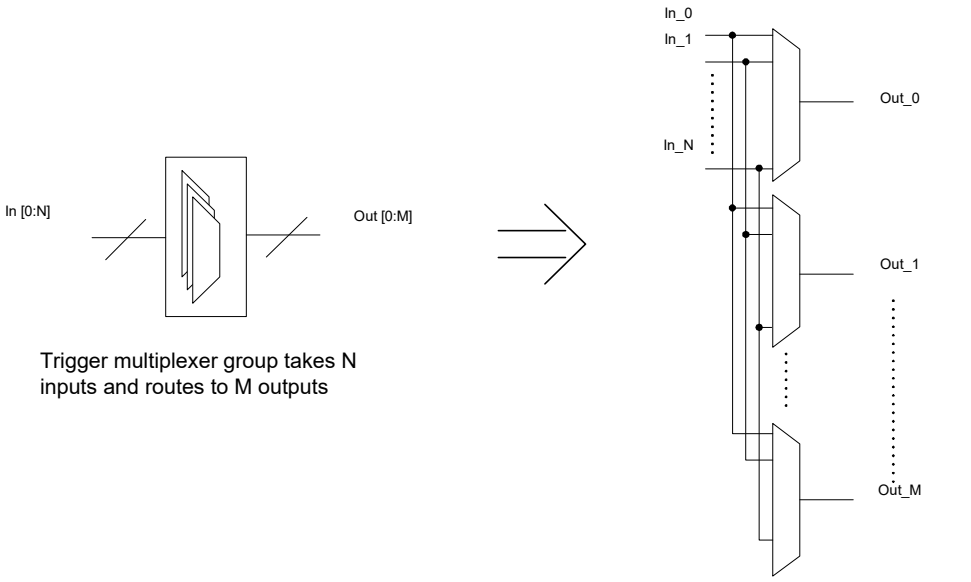
To support trigger routing, the PSoC 4 has hardware, which is a series of multiplexers used to route the trigger signals from potential sources to destinations. This hardware is called the trigger multiplexer block. The trigger multiplexer can connect to a trigger signal emanating out of a peripheral block in the PSoC 4 and route it to a different peripheral to initiate or affect an operation at the destination peripheral block. There are two types of triggers, level sensitive triggers and rising edge triggers. Rising edge triggers should remain '1' for at least 2 "clk_sys" cycles.

14.2.1 Trigger Multiplexer Group

The trigger multiplexer block is implemented using several trigger multiplexers. A trigger multiplexer selects a signal from a set of trigger output signals from different peripheral blocks to route it to a specific trigger input of another peripheral block as shown in Figure 14-3. The multiplexers are grouped into a trigger group. All trigger multiplexers in a trigger group have similar input options and are designed to feed similar destination signals. Hence, the trigger group can be considered as a block that multiplexes multiple inputs to multiple outputs. This concept is illustrated in Figure 14-2.

Note: The triggers output into different peripherals, which may have more routing than shown on the trigger routing diagram. For more information on this routing, go to the trigger destination peripheral block such as, Figure 5-2 and Figure 18-3.

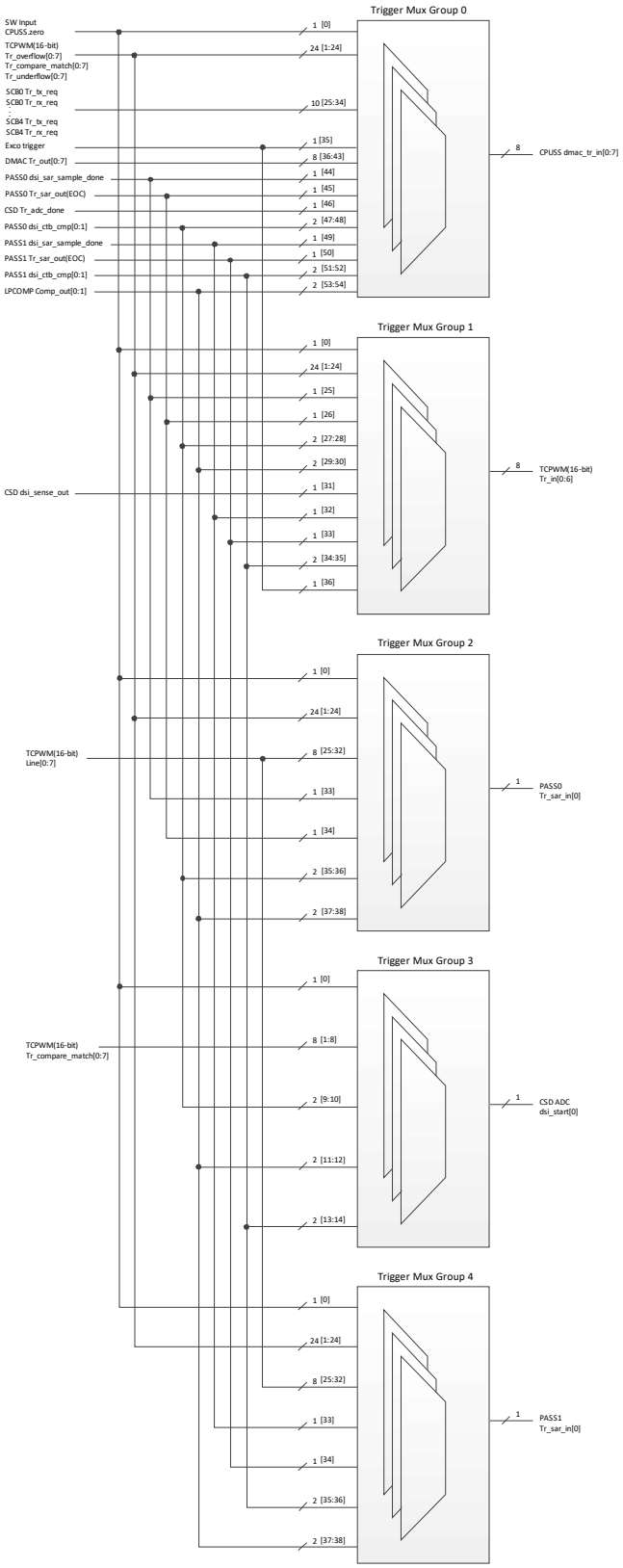
Figure 14-2. Trigger Multiplexer Groups



Trigger multiplexer group takes N inputs and routes to M outputs

An equivalent implementation of a trigger multiplexer group with N inputs and M outputs

Figure 14-3. Trigger Multiplexer Block Architecture



14.2.2 Software Triggers

All input and output signals to a trigger multiplexer can be triggered from software. This is accomplished by writing into the PERI_TR_CTL register. This register allows you to trigger the corresponding signal for a number of peripheral clock cycles. The PERI_TR_CTL[TR_GROUP] bit field selects the trigger group of the signal being activated. The PERI_TR_CTL[TR_OUT] bit field determines whether the trigger signal is in output or input of the multiplexer. PERI_TR_CTL[TR_SEL] selects the specific line in the trigger group. The PERI_TR_CTL[TR_COUNT] bit field sets up the number of triggers which will be activated. The PERI_TR_CTL[TR_ACT] bit field is set to '1' to activate the specified trigger line. Hardware resets this bit after the trigger is deactivated after the number of cycles set by PERI_TR_CTL[TR_COUNT].

14.3 Register List

Table 14-1. Register List

Register Name	Description
PERT_TR_CTL	Trigger command register. The control enables software activation of a specific input trigger or output trigger of the trigger multiplexer structure.
PERI_TR_GROUP[X]_TR_OUT_CTL[Y]	This register specifies the input trigger for a specific output trigger in a trigger group. Every trigger multiplexer group has a group of registers, the number of registers being equal to the output bus size from that multiplexer group. In the register format, X is the trigger group and Y is the output trigger line number from the multiplexer.

15. Reset System

The PSoC 4 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 4 also contains hardware to enable the detection of certain resets.

The PSoC 4 has these reset sources:

- POR to hold the device in reset while the power supply ramps up
- BOD to reset the device if the power supply falls below specifications during operation
- WDR to reset the device if firmware execution fails to service the WDT
- Software Initiated Reset (SRES) to reset the device on demand using firmware
- XRES to reset the device using an external electrical signal
- Protection fault reset (PROT_FAULT) to reset the device if unauthorized operating conditions occur

15.1 Reset Sources

The following sections provide a description of the reset sources available in PSoC 4.

15.1.1 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.

15.1.2 Brownout Detect Reset

Brownout detect 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 *PSoC 4500S datasheet*. BOD is available in all power modes.

15.1.3 Watchdog Reset

Watchdog reset detects errant code by causing a reset if the WDT 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 105.

15.1.4 Software-Initiated Reset (SRES)

Software initiated reset is a mechanism that allows a software-driven reset. The CM0+ 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 '05FA' written to the top two bytes for writes. Therefore, write '05FA0004' 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.

15.1.5 External Reset (XRES)

External reset 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 *PSoC 4500S 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 *PSoC 4500S 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.

15.1.6 Protection Fault Reset (PROT_FAULT)

Protection fault reset 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 100.

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.

15.2 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 PROT_FAULT. 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, XRES, or BOD.

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.

15.3 Register List

Table 15-1. Reset System Register List

Register Name	Description
WDT_DISABLE_KEY	This register disables the WDT when 0XACED8865 is written, for any other value WDT works normally
CM0P_AIRCR	CM0+ Application Interrupt and Reset Control Register - This register allows initiation of software resets, among other CM0+ functions.
RES_CAUSE	Reset Cause Register - This register captures the cause of recent resets.

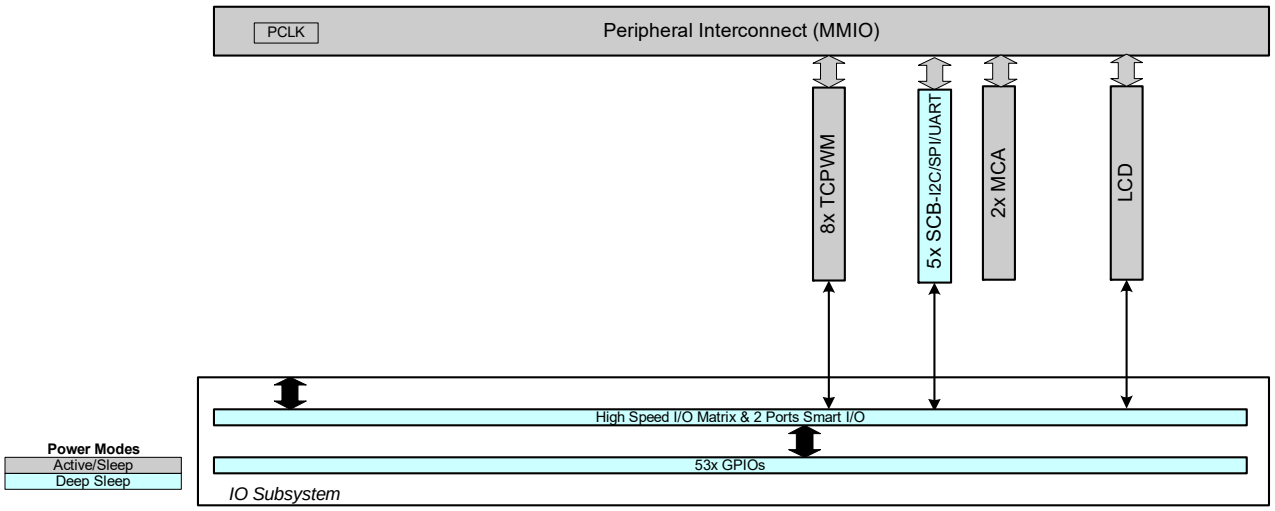
Section D: Digital System

This section encompasses the following chapters:

- Motor Control Accelerator chapter on page 117
- Serial Communications Block (SCB) chapter on page 122
- Timer, Counter, and PWM (TCPWM) chapter on page 165
- LCD Direct Drive chapter on page 187

Top Level Architecture

Digital System Block Diagram



16. Motor Control Accelerator

The PSoC 4500S has two MCAs. Each MCA consists of a high-performance 32-bit division hardware accelerator and a 32-bit square root hardware accelerator. PSoC 4500S will be used in the motor control applications. This acceleration enhances the CPU execution time for the FOC and PFC algorithms.

Note: The MCA block could also be used in other applications for significantly improving the operation efficiency of the program.

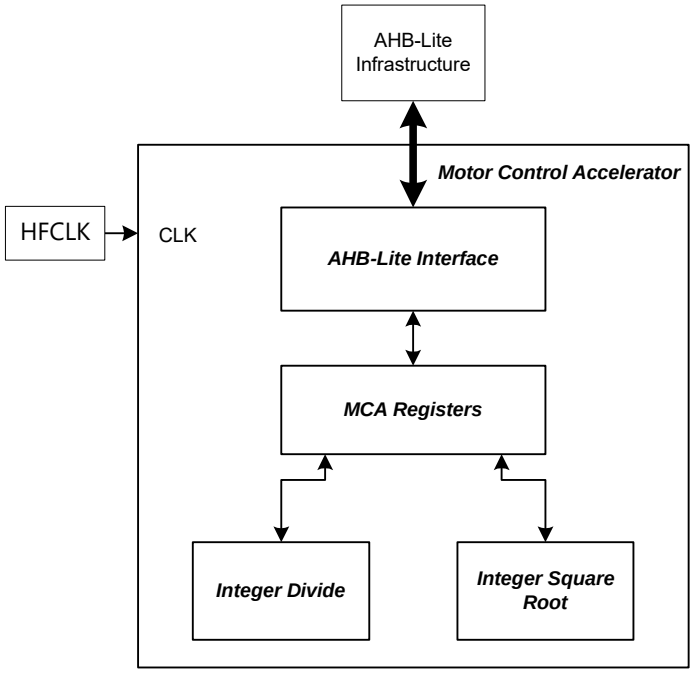
16.1 Features

Each of the two PSoC 4500S MCA modules provides the following features:

- Divide and square root operation run off HFCLK high speed clock
- 32-bit unsigned integer sequential divider
- 32-bit unsigned integer sequential square root
- Input operands programmed through MCA registers
- Output results accessed through MCA registers
- Two methods to perform the divide operation
- Two methods to perform the square root operation

16.2 Block Diagram

Figure 16-1. MCA Block Diagram



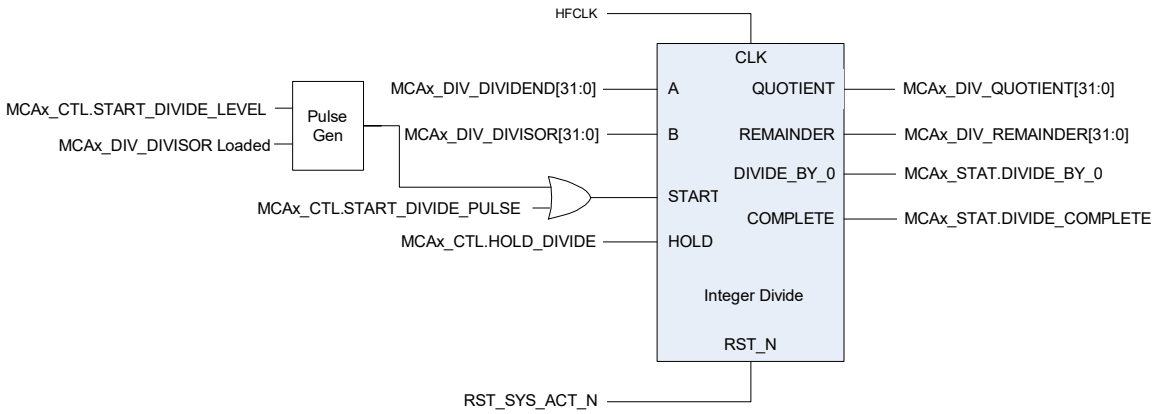
16.3 How it Works

In a typical motor control algorithm, there are multiple divide and square root operations. The accelerator replaces the inline software execution of these functions.

16.3.1 Integer Divider

The integer divider is shown in Figure 16-2.

Figure 16-2. Integer Divider Block Diagram



The interface to the device is sourced/synched by the MCA registers. The MCA registers setting is configured for the 32-bit Dividend/Divisor Width. Each divide operation takes eight clock cycles. The eight clock cycles do not include CPU and AHB access latencies. Table 16-1 provides the module interfaces.

Table 16-1. Module Interface

Port Name	Direction	Function
CLK	Input	Clock
RST_N	Input	Reset (Active Low)
START	Input	Start Operation – a new operation is started by setting START = 1.
HOLD	Input	Hold Current operation
A	Input	Dividend
B	Input	Divisor
COMPLETE	Output	Operation Complete
DIVIDE_BY_0	Output	Indicates that B is 0 value
QUOTIENT	Output	Quotient
REMAINDER	Output	Remainder

There are two methods to implement the module interfaces:

- **One-shot Operation Method**

START signal in Figure 16-2 can be used as a single clock pulse with the operation finished when the MCAx_STAT.DIVIDE_COMPLETE bit asserts. When the MCAx_CTL.HOLD_DIVIDE bit is set to ‘1’, the result of the operation will remain constant. Write 1 to the MCAx_CTL.START_DIVIDE_PULSE bit to provide the single clock pulse to the divider.

Pseudocode Sequence:

1. Enable block (MCAx_CONFIG.ENABLED).
2. Write MCAx_DIV_DIVIDEND.
3. Write MCAx_DIV_DIVISOR.
4. Write 1 to MCAx_CTL.START_DIVIDE_PULSE.
5. Check the state of MCAx_STAT.DIV_COMPLETE, or wait at least eight clock cycles for result generation.
6. Read MCAx_DIV_QUOTIENT.
7. Repeat steps 2 through 6 for more divide operations.

■ Continuous Operation Method

START signal in Figure 16-2 can be left constantly asserted, at which point the calculation starts after A and B are written. B should always be written last. In this method, the MCAx_CTL.HOLD_DIVIDE and MCAx_STAT.DIVIDE_COMPLETE bits are not used.

Use MCAx_CTL.START_DIVIDE_LEVEL to enable constant divider operation. Note that MCAx_CTL.START_DIVIDE_-PULSE and MCAx_CTL.START_DIVIDE_PULSE are mutually exclusive.

Pseudocode Sequence:

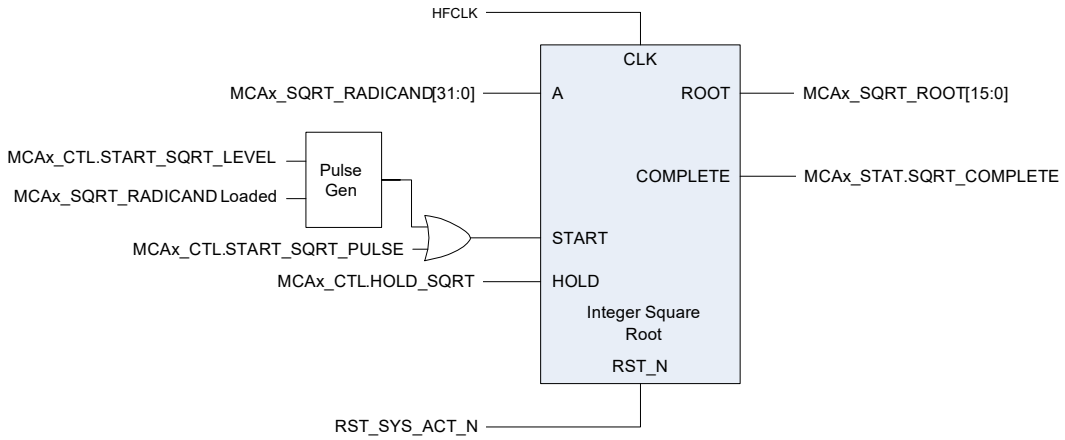
1. Enable block (MCAx_CONFIG.ENABLED).
2. Write 1 to MCAx_CTL.START_DIVIDE_LEVEL.
3. Write MCAx_DIV_DIVIDEND.
4. Write MCAx_DIV_DIVISOR (must follow MCAx_DIV_DIVIDEND writing).
5. Wait at least eight clock cycles for result generation.
6. Read MCAx_DIV_QUOTIENT.
7. Repeat steps 3 through 6 for more divide operations.

Note: The eight clock cycles do not include CPU and AHB access latencies.

16.3.2 Integer Square Root

The Integer Square Root module is shown in Figure 16-3.

Figure 16-3. Integer Square Root Block Diagram



The interface to the device is sourced/synched by the MCA registers. The MCA registers setting is configured for the 32-bit Radicand and a 16-bit Root. Each square root operation takes four clock cycles. The four clock cycles do not include CPU and AHB access latencies.

Table 16-2 documents the module interfaces.

Table 16-2. Module Interface

Port Name	Direction	Function
CLK	Input	Clock
RST_N	Input	Reset (Active Low)
START	Input	Start Operation – a new operation is started by setting START = 1.
HOLD	Input	Hold Current operation
A	Input	Radicand
COMPLETE	Output	Operation Complete
ROOT	Output	Root

There are two methods to implement the module interfaces:

- **One-shot Operation Method**

START bit in Figure 16-3 can be used as a single clock pulse with the operation finished when the MCAx_STAT.SQRT_COMPLETE bit asserts. When the MCAx_CTL.HOLD_SQRT bit is set to '1', the result of the operation will remain constant. Write 1 to the MCAx_CTL.START_SQRT_PULSE bit to provide the single clock pulse to the Square Root module.

Pseudocode Sequence:

1. Enable block (MCAx_CONFIG.ENABLED).
2. Write MCAx_SQRT_RADICAND.
3. Write 1 to CTL.START_SQRT_PULSE.
4. Check state of MCAx_STAT.SQRT_COMPLETE, or wait at least four clock cycles for result generation.
5. Read MCAx_SQRT_ROOT.
6. Repeat steps 2 through 5 for more square root operations.

- **Continuous Operation Method**

START bit in Figure 16-3 can be left constantly asserted, at which point the calculation starts after A is written. In this method, the MCAx_CTL.HOLD_SQRT and MCAx_STAT.SQRT_COMPLETE bits are not used.

Use MCAx_CTL.START_SQRT_LEVEL to enable constant Square Root operation. Note that MCAx_CTL.START_SQRT_PULSE and MCAx_CTL.START_SQRT_LEVEL are mutually exclusive.

Pseudocode Sequence:

1. Enable block (MCAx_CONFIG.ENABLED).
2. Write 1 to MCAx_CTL.START_SQRT_LEVEL.
3. Write MCAx_SQRT_RADICAND.
4. Wait at least four clock cycles for result generation.
5. Read MCAx_SQRT_ROOT.
6. Repeat steps 3 through 5 for more square root operations.

Note: The four clock cycles do not include CPU and AHB access latencies.

16.3.3 Power Mode

The block only works in Active power mode.

16.3.4 Initialization

After disable, reset, or wakeup from Low-Power mode, all MCA registers enter the default state.

16.4 Register

Table 16-3. List of MCA Registers

Register	Description
MCAx_CONFIG	Configuration Register
MCAx_CTL	Control Register
MCAx_STAT	Status Register
MCAx_DIV_DIVIDEND	32-bit Dividend
MCAx_DIV_DIVISOR	32-bit Divisor
MCAx_DIV_QUOTIENT	32-bit Quotient
MCAx_DIV_REMAINDER	32-bit Remainder
MCAx_SQRT_RADICAND	Square Root Source Value
MCAx_SQRT_ROOT	Square Root Result

17. Serial Communications Block (SCB)

The Serial Communications Block (SCB) of PSoC 4 supports three serial interface protocols: SPI, UART, and I^2C. Only one of the protocols is supported by an SCB at any given time. The PSoC 4500S device has up to five SCBs.

17.1 Features

The SCB supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, LIN, and IrDA protocols
  - Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
- Standard I^2C master and slave functionality
- Features available only on Deep Sleep-capable SCB:
  - 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)

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

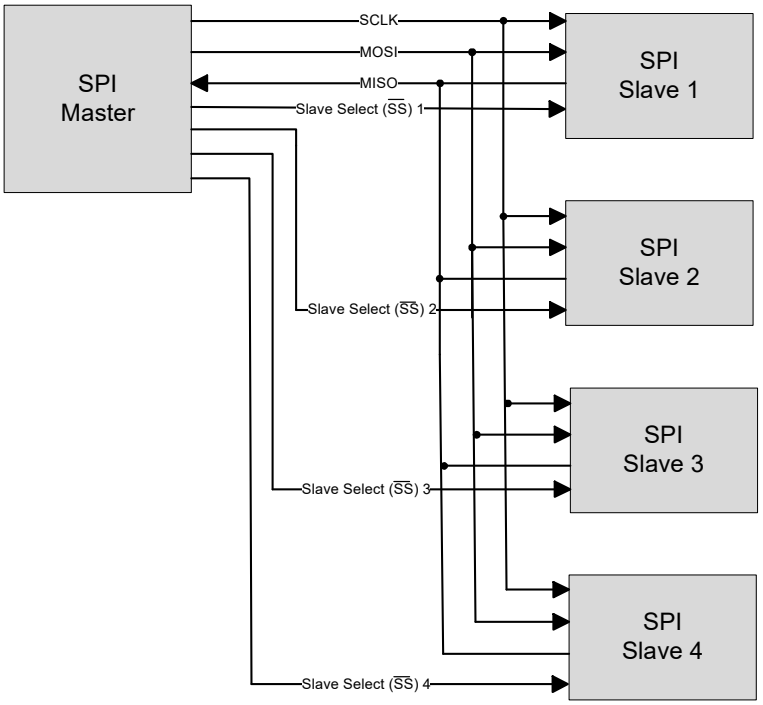
17.2.1 Features

- 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 (EZSPI) 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

17.2.2 General Description

Figure 17-1 illustrates an example of SPI master with four slaves.

Figure 17-1. SPI Example



A standard SPI interface consists of the following four signals:

- 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 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.

17.2.3 SPI Modes of Operation

17.2.3.1 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 17-2 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

Figure 17-2. SPI Motorola, Four Modes

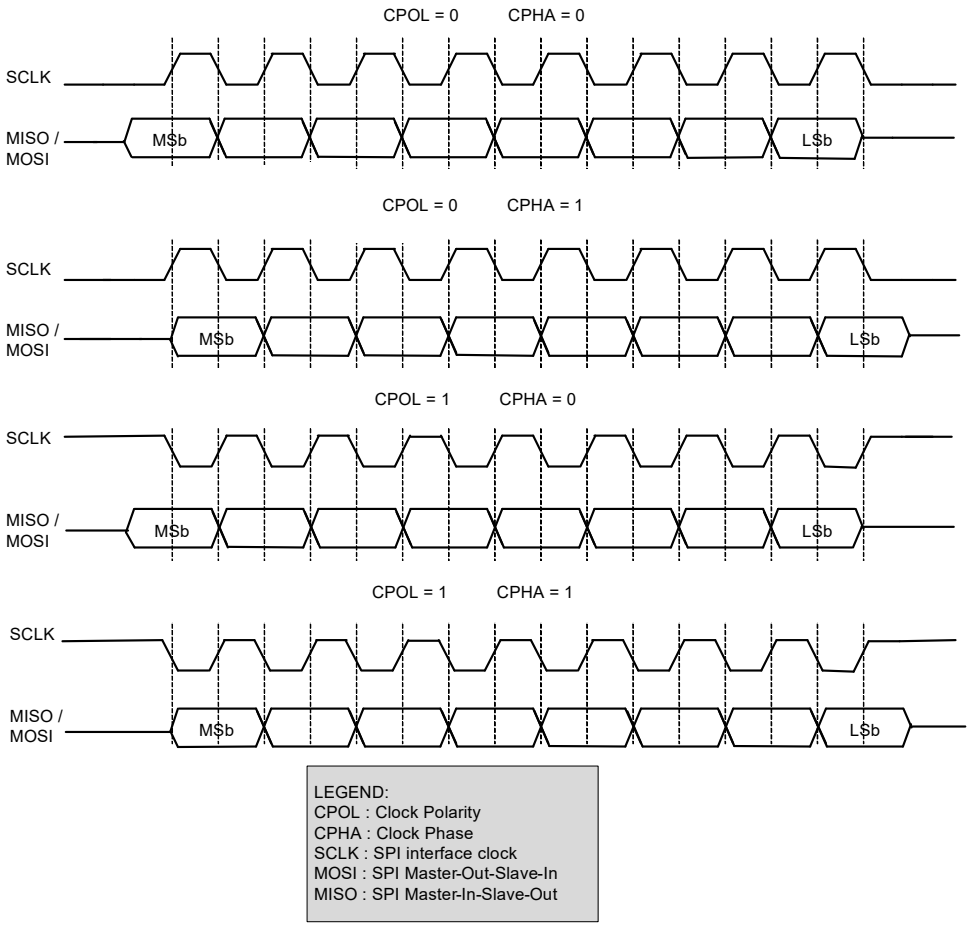
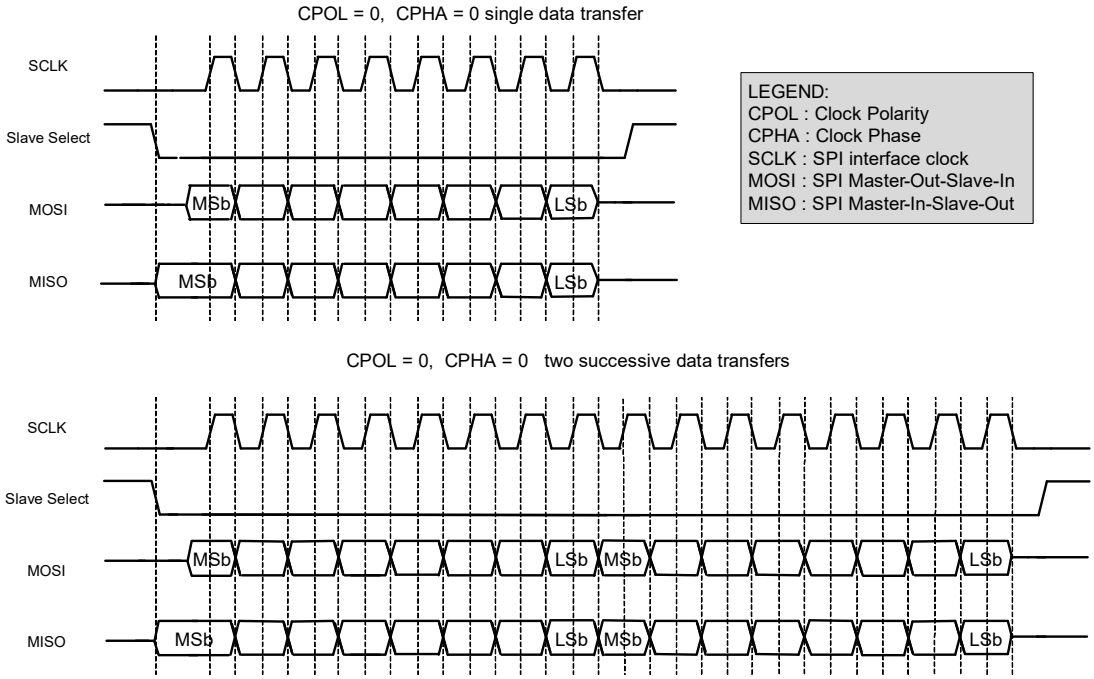


Figure 17-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is ‘0’, CPHA is ‘0’).

Figure 17-3. 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 132.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

17.2.3.2 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 an active LOW 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 an active HIGH 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 17-4 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 17-4. SPI TI Data Transfer Example

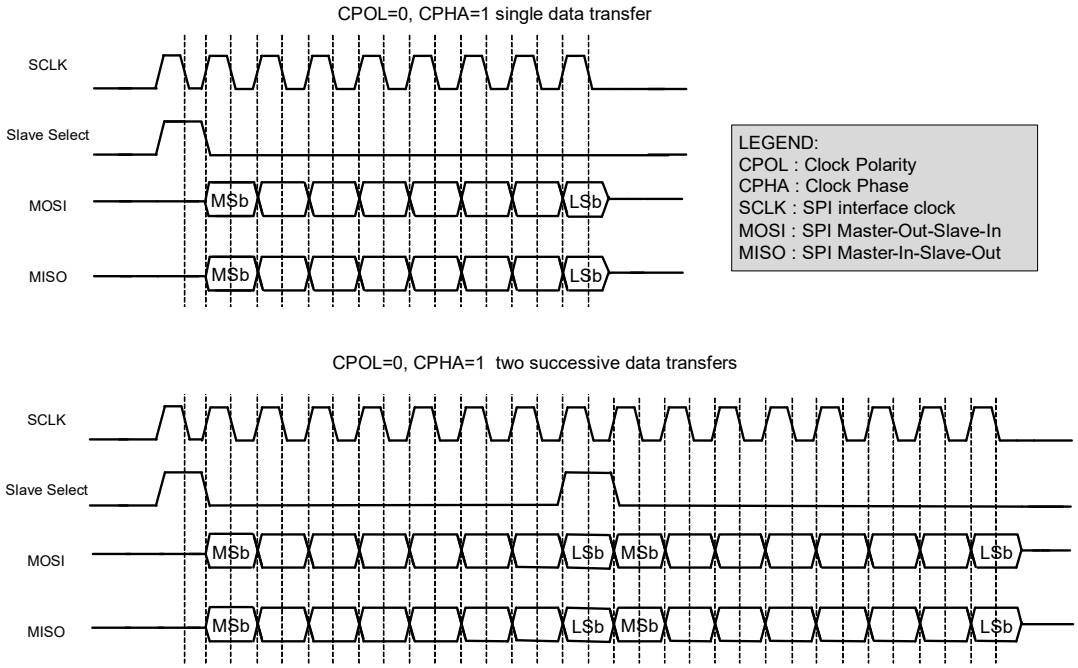
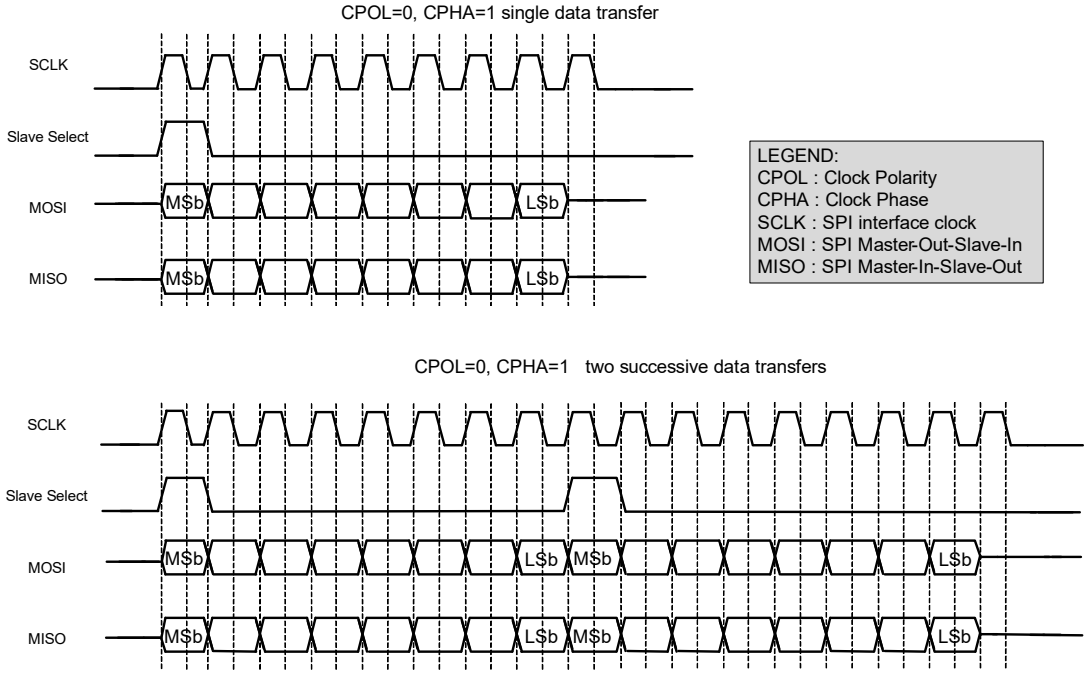


Figure 17-5 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

Figure 17-5. SPI TI Data Transfer Example



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 132.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

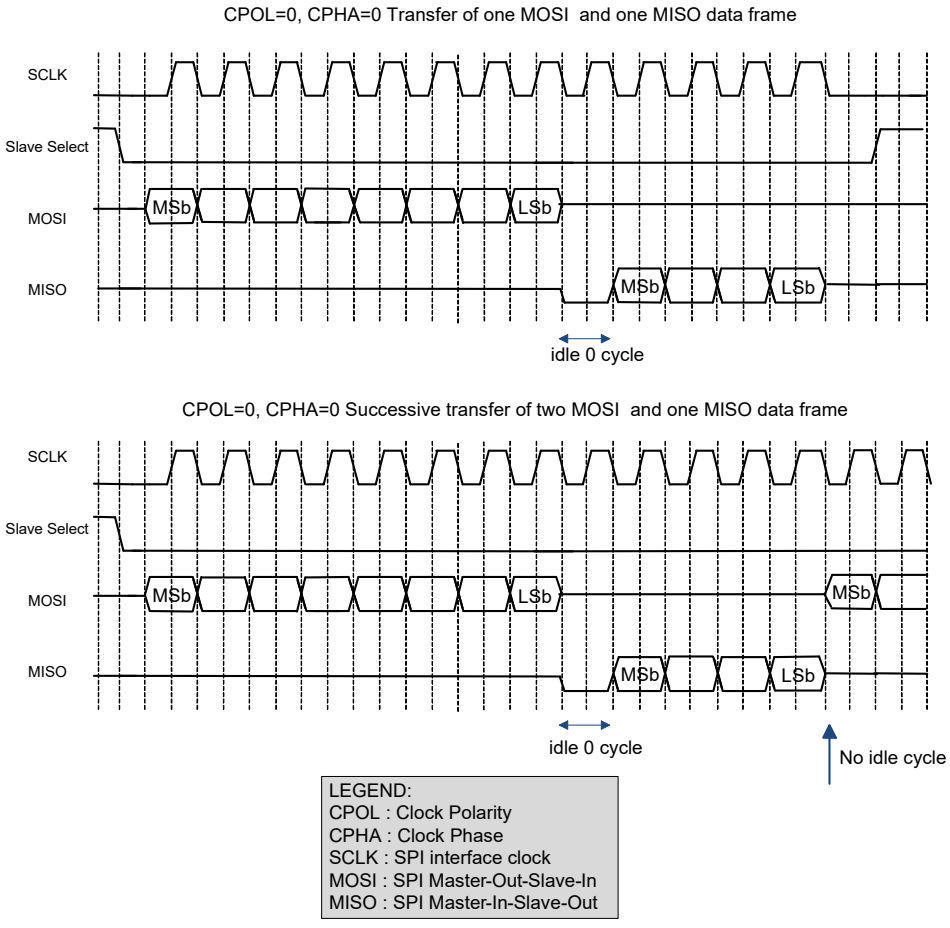
17.2.3.3 National Semiconductors SPI

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 17-6 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 17-6. 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 132.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

17.2.4 Using SPI Master to Clock Slave

In a normal SPI Master mode transmission, 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.

17.2.5 Easy SPI (EZSPI) Protocol

The 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 bits per entry is supported) located on the slave device. To address these 32 locations, the lower 5 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 enabled. 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.

17.2.5.1 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.

17.2.5.2 Memory Array Write

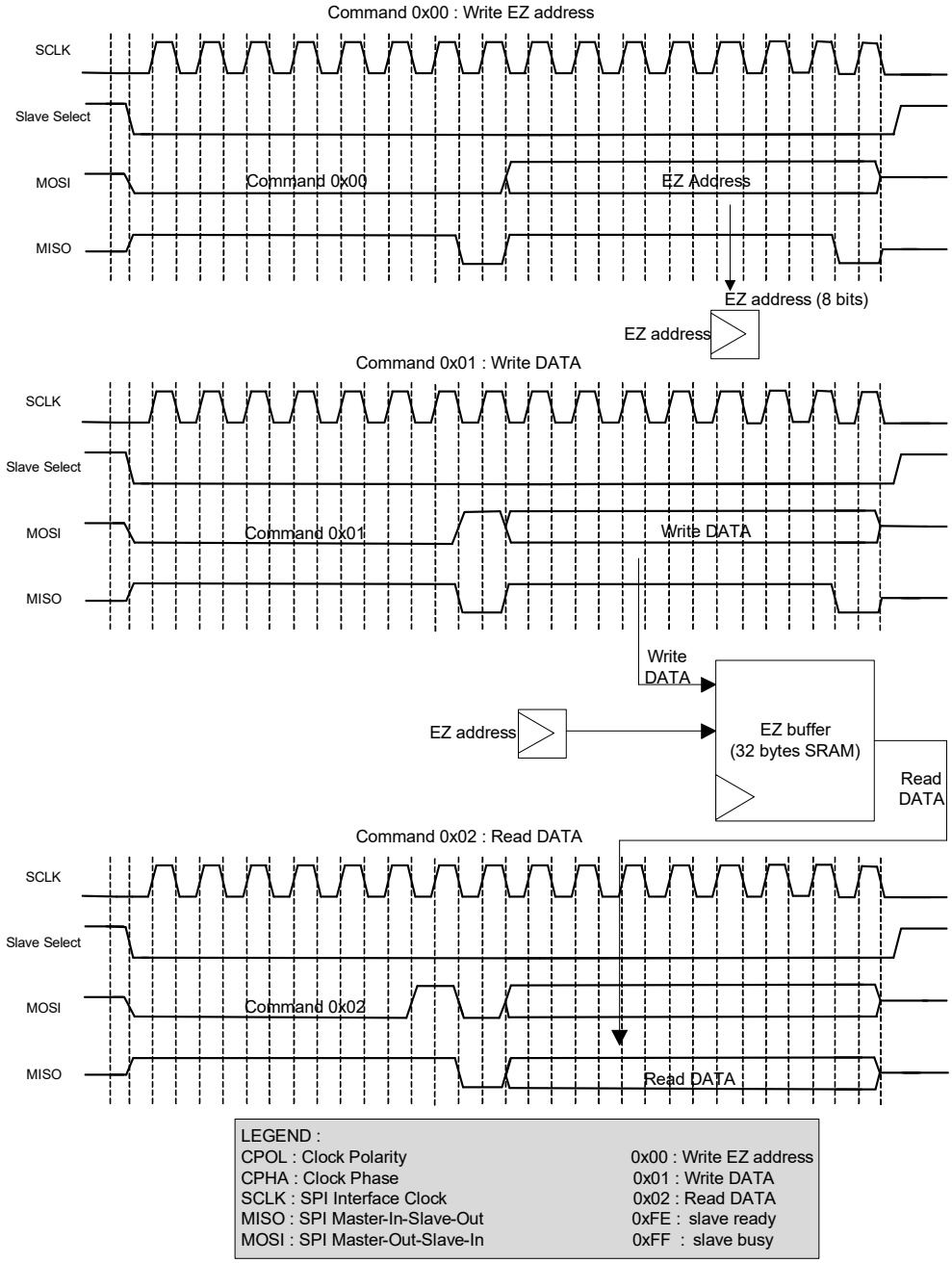
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional write data bytes on the 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’.

17.2.5.3 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 17-7 illustrates the write of EZ address, write to a memory array, and read from a memory array operations in the EZSPI protocol.

Figure 17-7. EZSPI Example



17.2.5.4 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 132.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

17.2.6 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 17-1. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

Table 17-1. 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. When SPI work as slave mode, only the first Chip Select pin SPI_SELECT[0] can be used in slave mode.
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 whether 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

17.2.7 SPI Interrupts

SPI supports both internal and external interrupt requests. The internal interrupt events are listed here. PSoC Creator generates the necessary 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 CM0+ NVIC and cannot be routed to external pins.

17.2.8 Enabling and Initializing SPI

SPI must be programmed in the following order:

1. Program protocol-specific information using the SCB_SPI_CTRL register, according to Table 17-3. 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 17-4.
   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 17-5.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFO.
4. Program the SCB_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in Table 17-2.
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 17-2. 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 17-3. 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 17-4. 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 first. For 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 17-5. 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.

17.2.9 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 HFCLK of the system. For more information on system clocking, see the Clocking System chapter on page 85. It also supports oversampling. Oversampling is implemented with respect to the HFCLK. The OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

In SPI master mode, the valid range for oversampling is from 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 the 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 listed in Table 17-6.

Table 17-6. 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 operations are 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 modes. Improper setting may result in faulty behavior in certain system power modes. Table 17-7 and Table 17-8 describe the settings for SPI (in non-EZ and EZ modes).

17.2.9.1 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 17-7 gives an overview of the possibilities.

Table 17-7. SPI Operation in Non-EZ Mode

SPI (non-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	Selection using internal clock. Operation using internal clock.	Selection using EXTCLK: Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). You can configure the MASK bit in Sleep mode.	Not supported	Not supported
Deep Sleep	Not supported	Selection using EXTCLK: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		

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 (wakeup) 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’). However, 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 Sleep mode are same as that of 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.

17.2.9.2 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 17-8 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 17-8. SPI Operation in EZ Mode

SPI, 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	Selection using internal clock. Operation using internal clock.	Selection using EXTCLK. Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). You can configure the MASK bit in Sleep mode.	Invalid	Selection using EXTCLK. Operation using EXTCLK.
Deep Sleep	Not supported	Selection using EXTCLK: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		Selection using EXTCLK. Operation using EXTCLK.

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 (wakeup) 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’). However, 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 Sleep mode are same as that of 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.

17.3 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

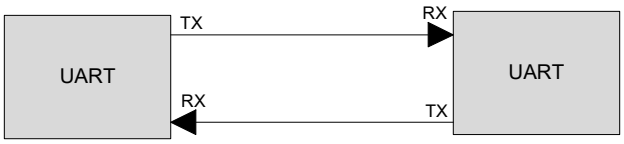
17.3.1 Features

- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 3 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports 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

17.3.2 General Description

Figure 17-8 illustrates a standard UART TX and RX.

Figure 17-8. 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
  - LIN
- SmartCard, similar to UART, but with a possibility to send a negative acknowledgment
- 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 17-9 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.

17.3.3 UART Modes of Operation

17.3.3.1 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.

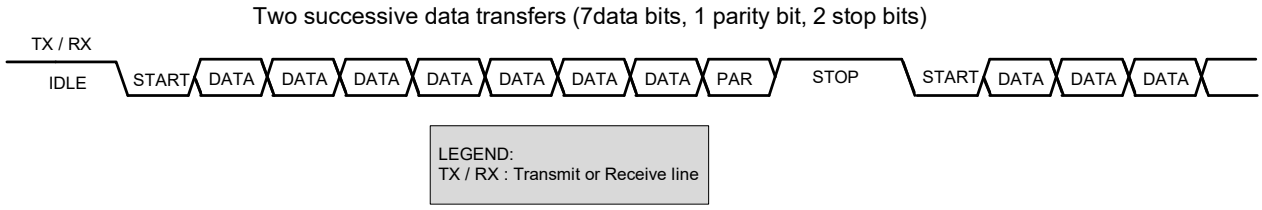
The interface does not have a clock, so the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample the incoming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

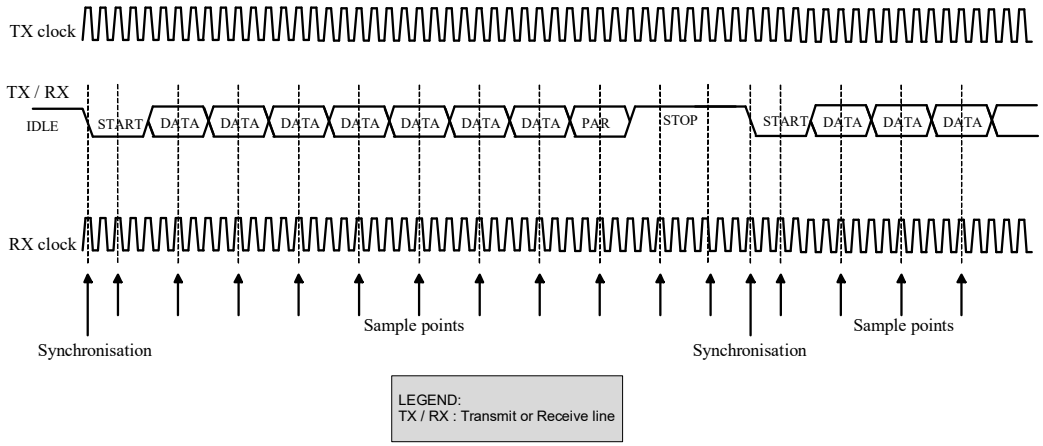
Figure 17-9 illustrates the UART protocol.

Figure 17-9. UART, Standard Protocol Example



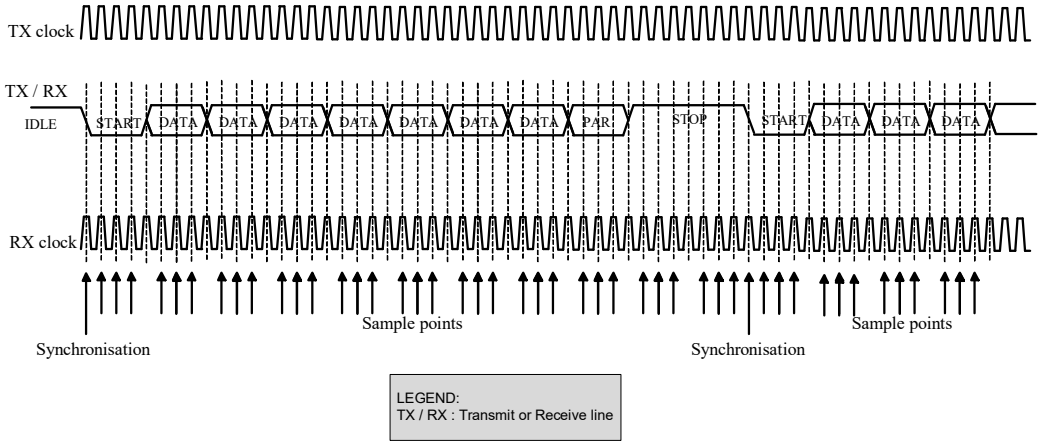
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used (see Figure 17-10).

Figure 17-10. 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 (see Figure 17-11).

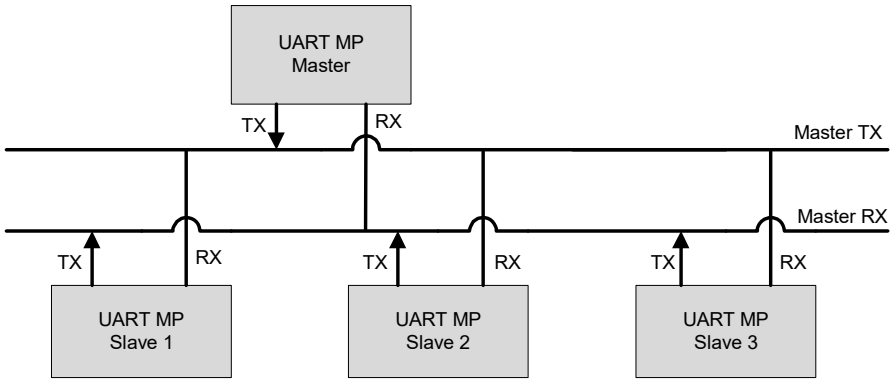
Figure 17-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor (UART_MP) Mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as Figure 17-12 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 17-12. UART MP Mode Bus Connections



The main properties of the 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 17-13.
- Parity bit is disabled.

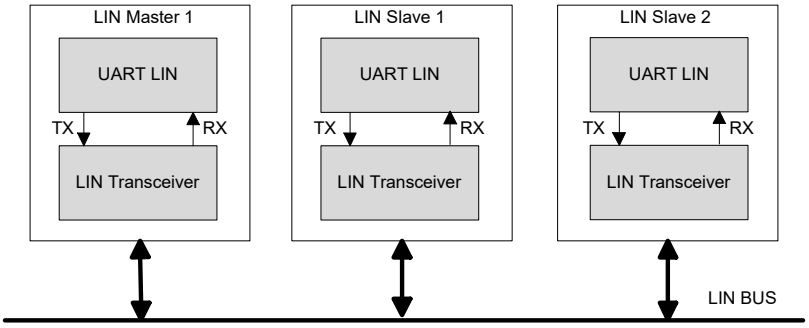
Figure 17-13. UART MP Data Frame

The SCB can be used as either master or slave device in the 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 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 17-14 illustrates the UART_LIN and LIN Transceiver.

Figure 17-14. UART_LIN and LIN Transceiver

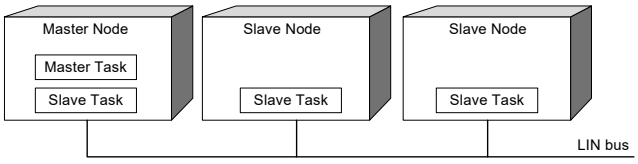


LIN protocol defines two tasks:

- 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 17-15.

Figure 17-15. LIN Bus Nodes and Tasks

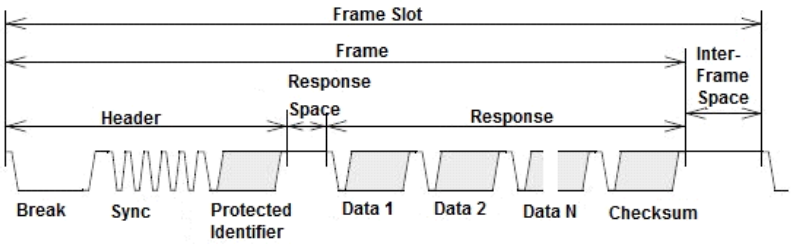


LIN Frame Structure

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 17-16.

- 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 17-16. LIN Frame Structure

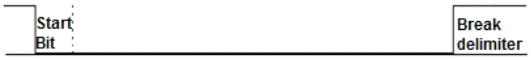


In LIN protocol communication, the LSb of the data is sent first and the MSb last. The start bit is encoded as zero and the stop bit is encoded as one. The following sections describe all byte fields in the LIN frame.

Break Field

Every new frame starts with a break field, which is always generated by the master. The break field 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 17-17.

Figure 17-17. 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 17-18 shows the LIN sync field structure.

Figure 17-18. LIN Sync Field

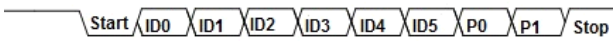
Protected Identifier (PID) Field

A PID 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 17-19.

- 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 17-19 shows the PID field structure.

Figure 17-19. 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 contain eight data bytes.

The frame identifier for diagnostic frame can be either of the following:

- Master request frame (0x3C)
- 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 Wakeup

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.

Wakeup 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 146.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

17.3.3.2 SmartCard (ISO7816)

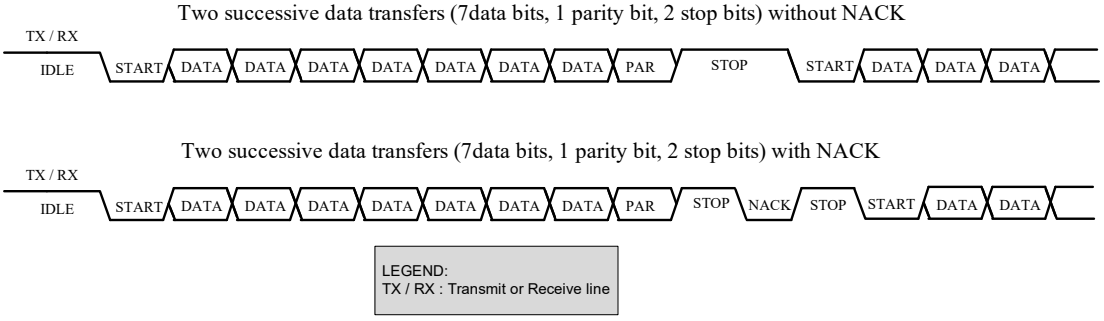
ISO7816 is an asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, see the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic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

The SmartCard transfer is similar to a UART transfer, with the addition of a negative acknowledgment (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 17-20 illustrates the SmartCard protocol.

Figure 17-20. 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 4500S: PSoC 4 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 146.

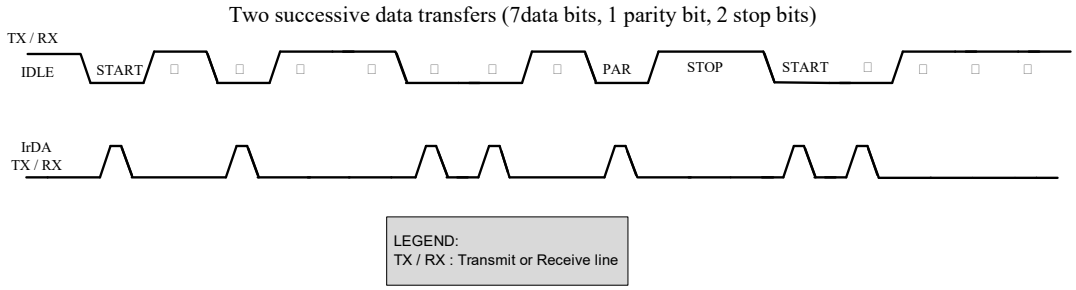
17.3.3.3 IrDA

The SCB supports the IrDA protocol for data rates of up to 115.2 kbps using the UART interface. It supports only the basic physical layer of the 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 17-21 shows how a UART transfer is IrDA modulated.

Figure 17-21. 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 4500S: PSoC 4 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 146.

17.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 17-9. For more information on these registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

Table 17-9. 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 the 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 the same function as that of the SCB_UART_TX_CTRL register, 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.

17.3.5 UART Interrupts

The UART supports both internal and external interrupt requests. This section lists the internal interrupt events. PSoC Creator generates the necessary 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

17.3.6 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 17-10. 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 17-11.
   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 17-12.
   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 (see Table 17-13).
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 17-10. 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 17-11. 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.
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 17-12. 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 17-13. 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

17.4 Inter Integrated Circuit (I^2C)

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

17.4.1 Features

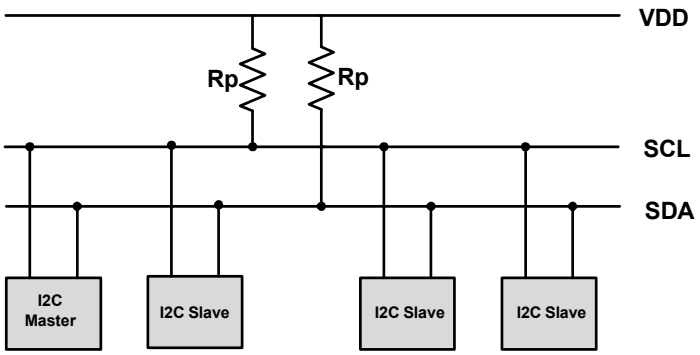
This block supports the following features:

- 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 serial clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I^2C serial data (SDA) signal
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface

17.4.2 General Description

Figure 17-22 illustrates an example of an I^2C communication network.

Figure 17-22. I^2C Interface Block Diagram



The standard I²C bus is a two-wire interface with the following lines:

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

I²C 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 I²C with firmware support.

17.4.3 Terms and Definitions

Table 17-14 explains the commonly used terms in an I²C communication network.

Table 17-14. Definition of I²C 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.

17.4.3.1 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.

17.4.3.2 Bus Arbitration

The I²C 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.

17.4.4 I²C Modes of Operation

I²C 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 17-15 illustrates the I²C modes of operation.

Table 17-15. I²C 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 I2C bus follows a specific format. Table 17-16 lists some common bus events that are part of an I2C data transfer. The Write Transfer and Read Transfer sections explain the I2C bus bit format during data transfer.

Table 17-16. I2C Bus Events Terminology

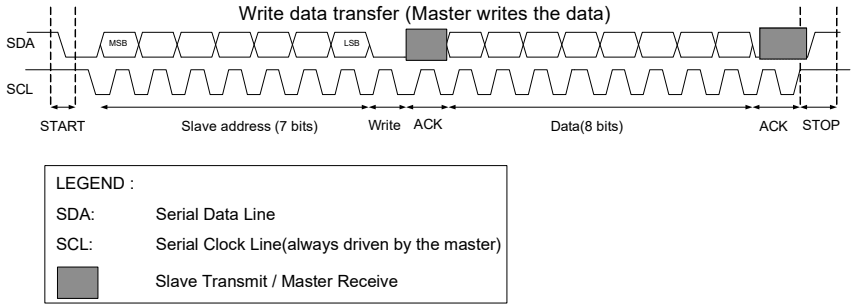
Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH.
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH.
ACK	The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse, after the transmitter transmits each byte. This 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 17-16, Figure 17-23, and Figure 17-24). 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 157, “Slave Mode Transfer Examples” on page 159, and “Multi-Master Mode Transfer Example” on page 163.

17.4.4.1 Write Transfer

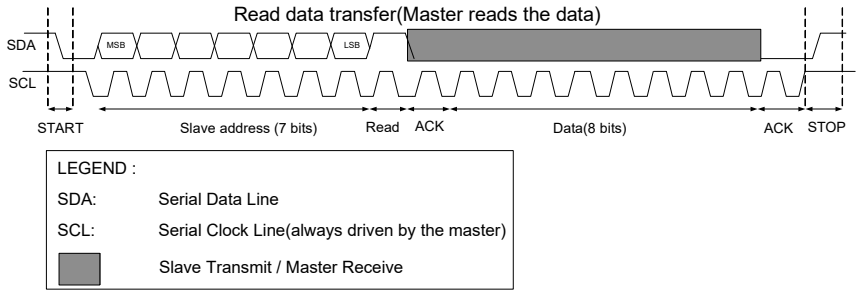
Figure 17-23. Master Write Data Transfer



- A typical write transfer begins with the master generating a START condition on the I2C bus. The master then writes a 7-bit I2C slave address and a write indicator (‘0’) after the START condition. The addressed slave transmits an acknowledgment 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 acknowledgment (NACK) by not pulling the SDA line low. The absence of an acknowledgment, results in an SDA line value of ‘1’ due to the pull-up resistor implementation.
- If no acknowledgment 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 acknowledgment. The addressed slave transmits an acknowledgment to confirm the receipt of every byte of data written. Upon receipt of this acknowledgment, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

17.4.4.2 Read Transfer

Figure 17-24. Master Read Data Transfer



- A typical read transfer begins with the master generating a START condition on the I^{2}C bus. The master then writes a 7-bit I^{2}C slave address and a read indicator (‘1’) after the START condition. The addressed slave transmits an acknowledgment 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 acknowledgment (NACK) is transmitted by not pulling the SDA line low. The absence of an acknowledgment, results in an SDA line value of ‘1’ due to the pull-up resistor implementation.
- If no acknowledgment 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 acknowledgment signal. The master transmits an acknowledgment to confirm the receipt of each data byte sent by the slave. Upon receipt of this acknowledgment, 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.

17.4.5 Easy I2C (EZI2C) Protocol

The EZI2C protocol is a unique communication scheme built on top of the I^{2}C protocol by Cypress. It uses a software wrapper around the standard I^{2}C protocol to communicate to an I^{2}C 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^{2}C 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.

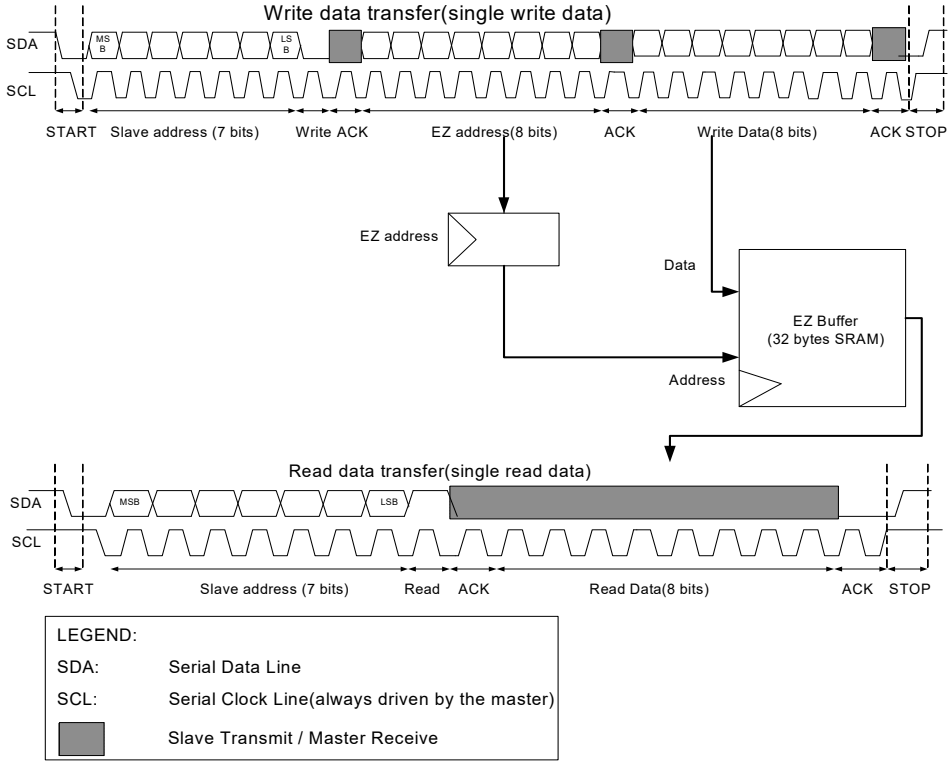
17.4.5.1 Memory Array Write

An EZ write to a memory array index is by means of an I^{2}C 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.

17.4.5.2 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 17-25. EZI2C Write and Read Data Transfer



17.4.6 I2C Registers

The I2C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in Table 17-17.

Table 17-17. I2C Registers

Register	Function
SCB_CTRL	Enables the I2C block and selects the type of serial interface (SPI, UART, 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.

Table 17-17. I2C Registers *(continued)*

Register	Function
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 4500S: PSoC 4 Registers TRM*.

17.4.7 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 CM0+ 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 4500S: PSoC 4 Registers TRM*.

17.4.8 Enabling and Initializing the I2C

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

17.4.8.1 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 17-18. 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 17-19.
   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 17-20.
   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 17-21. For a complete description of the I2C registers, see the *PSoC 4500S: PSoC 4 Registers TRM*.

Table 17-18. SCB_I2C_CTRL Register

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

Table 17-19. 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 17-20. 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 17-21. 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

17.4.8.2 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.

17.4.9 Internal and EXTCLK 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 that you provide. 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 85.

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 17-22 and Table 17-23 describe the settings for I2C in EZ and non-EZ mode.

17.4.9.1 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 17-22 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. 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 17-22. I2C Operation in Non-EZ Mode

I2C (Non-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 EXTCLK. Operation using internal clock.	Not supported	
Deep Sleep	Not supported	Address match using EXTCLK. Operation using internal clock.		

- 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’).

17.4.9.2 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 17-23 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 17-23. 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 EXTCLK Operation using internal clock	Invalid	Address match using EXTCLK Operation using EXTCLK
Deep Sleep	Not supported	Address match using EXTCLK Operation using internal clock		Address match using EXTCLK Operation using EXTCLK

- **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.

17.4.10 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 Sleep mode.

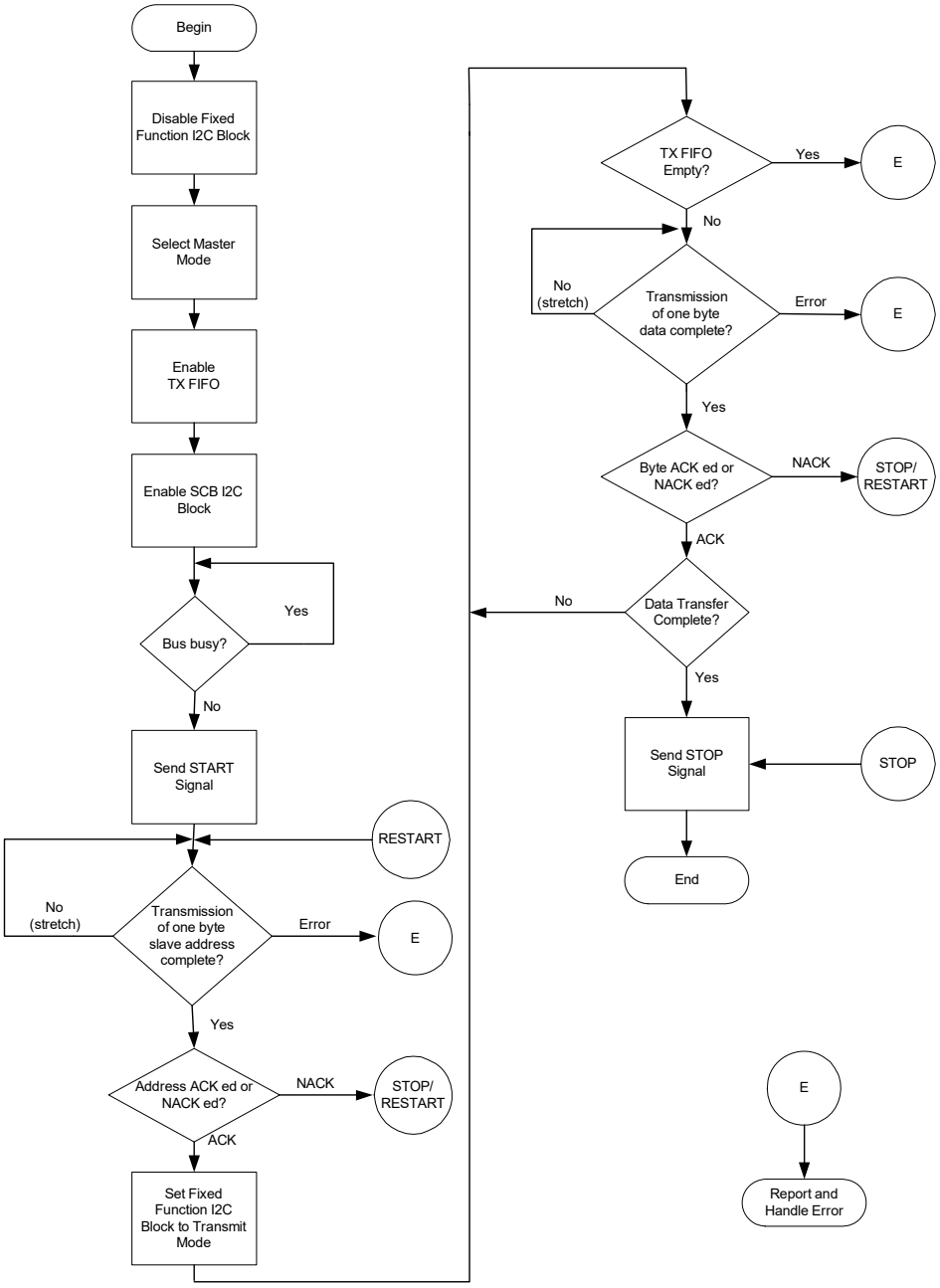
Note: If the device is configured in I2C slave mode, the clock to the SCB should be disabled when entering Deep Sleep power mode; enable the clock when waking up from Deep Sleep mode.

17.4.11 Master Mode Transfer Examples

Master mode transmits or receives data.

17.4.11.1 Master Transmit

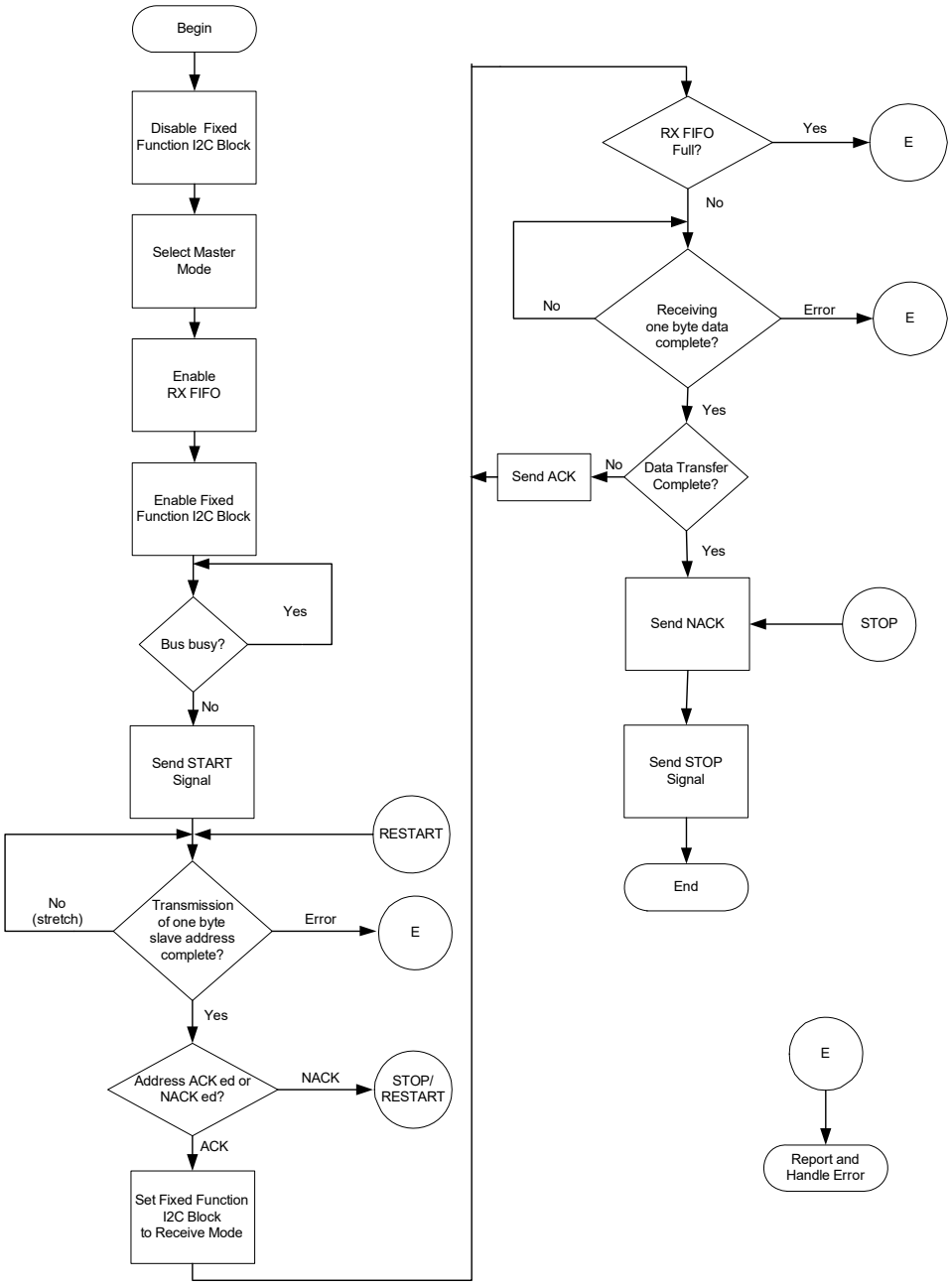
Figure 17-26. Single Master Mode Write Operation Flowchart



Note: The SCB hardware generates a Start condition when there is no pending transfer and returns (does not wait until hardware generates a start condition). If the I2C bus is busy, the hardware will not generate Start condition until bus becomes free. The SCB hardware sets the busy status after detecting Start, and clears it on detecting Stop. Noise caused by the ESD or other events may cause an erroneous Start condition on the bus. Then, the master will not generate a Start condition because the hardware assumes the bus is busy. If this occurs, the SCB block must be reset.

17.4.11.2 Master Receive

Figure 17-27. Single Master Mode Read Operation Flowchart



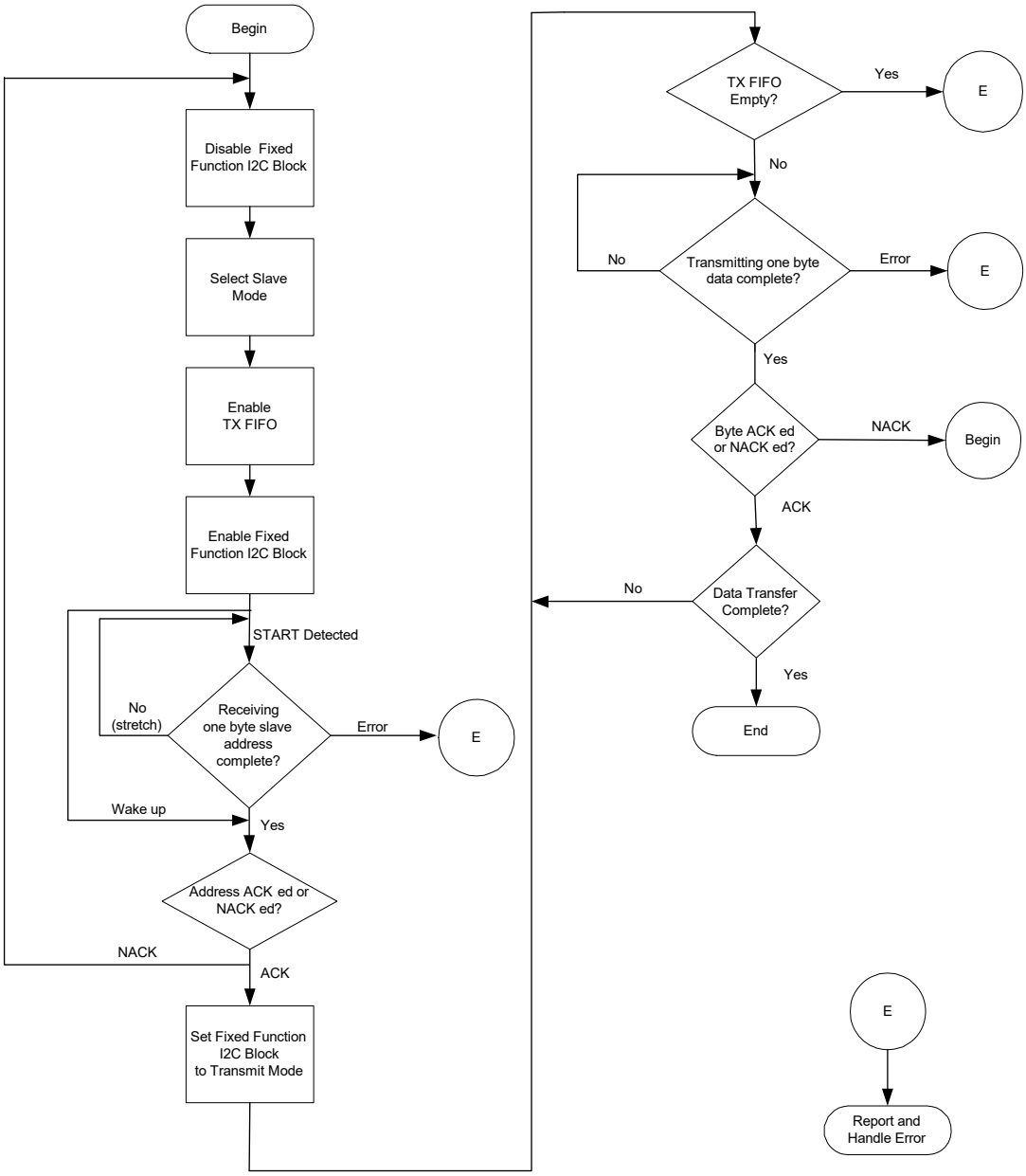
Note: The SCB hardware generates a Start condition when there is no pending transfer and returns (does not wait until hardware generates a Start condition). If the I2C bus is busy, the hardware will not generate the Start condition until bus becomes free. The SCB hardware sets the busy status after detecting Start, and clears it on detecting Stop. Noise caused by the ESD or other events may cause an erroneous Start condition on the bus. Then, the master will not generate a Start condition because the hardware assumes the bus is busy. If this occurs, the SCB block must be reset.

17.4.12 Slave Mode Transfer Examples

Slave mode transmits or receives data.

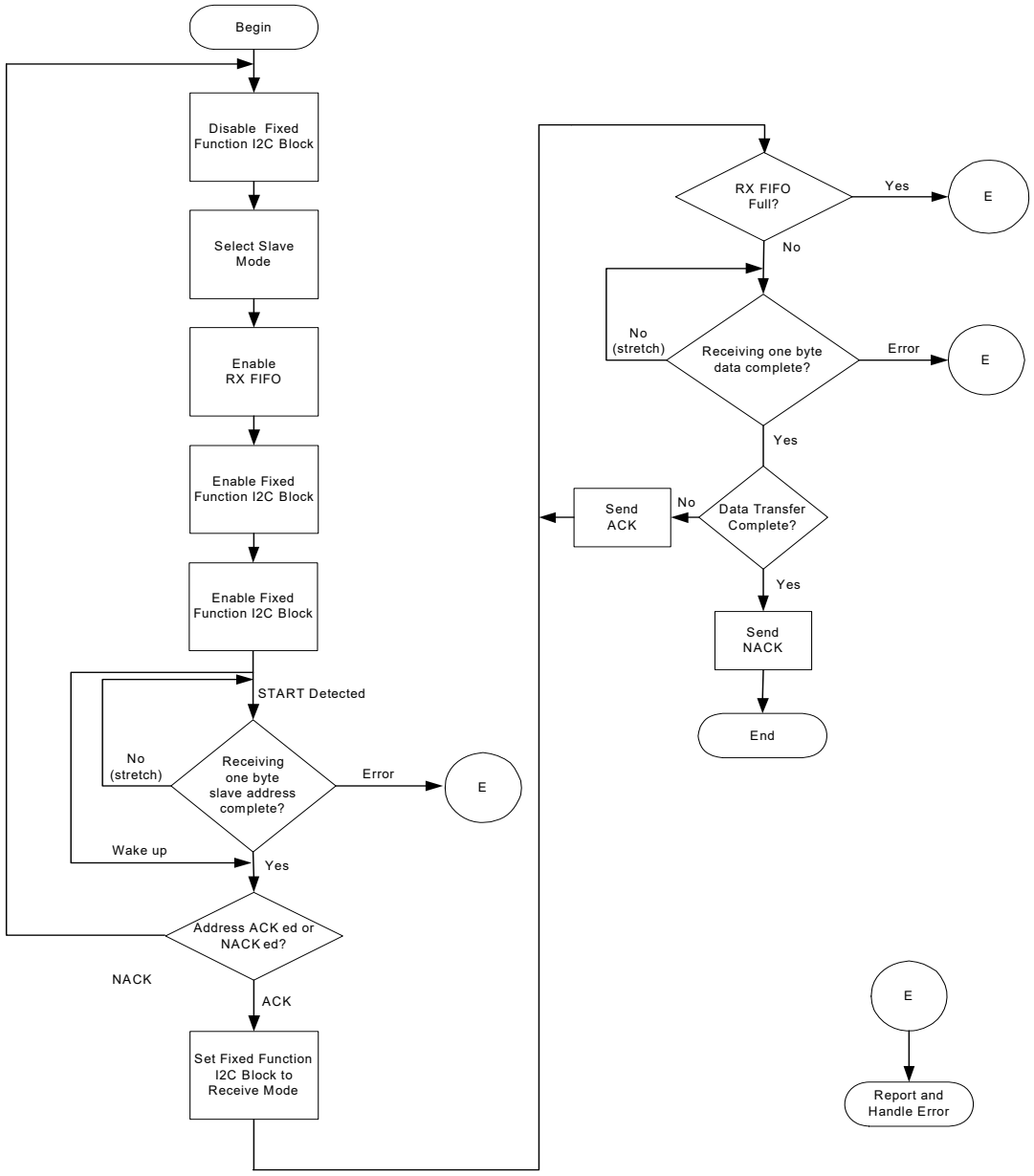
17.4.12.1 Slave Transmit

Figure 17-28. Slave Mode Write Operation Flowchart



17.4.12.2 Slave Receive

Figure 17-29. Slave Mode Read Operation Flowchart

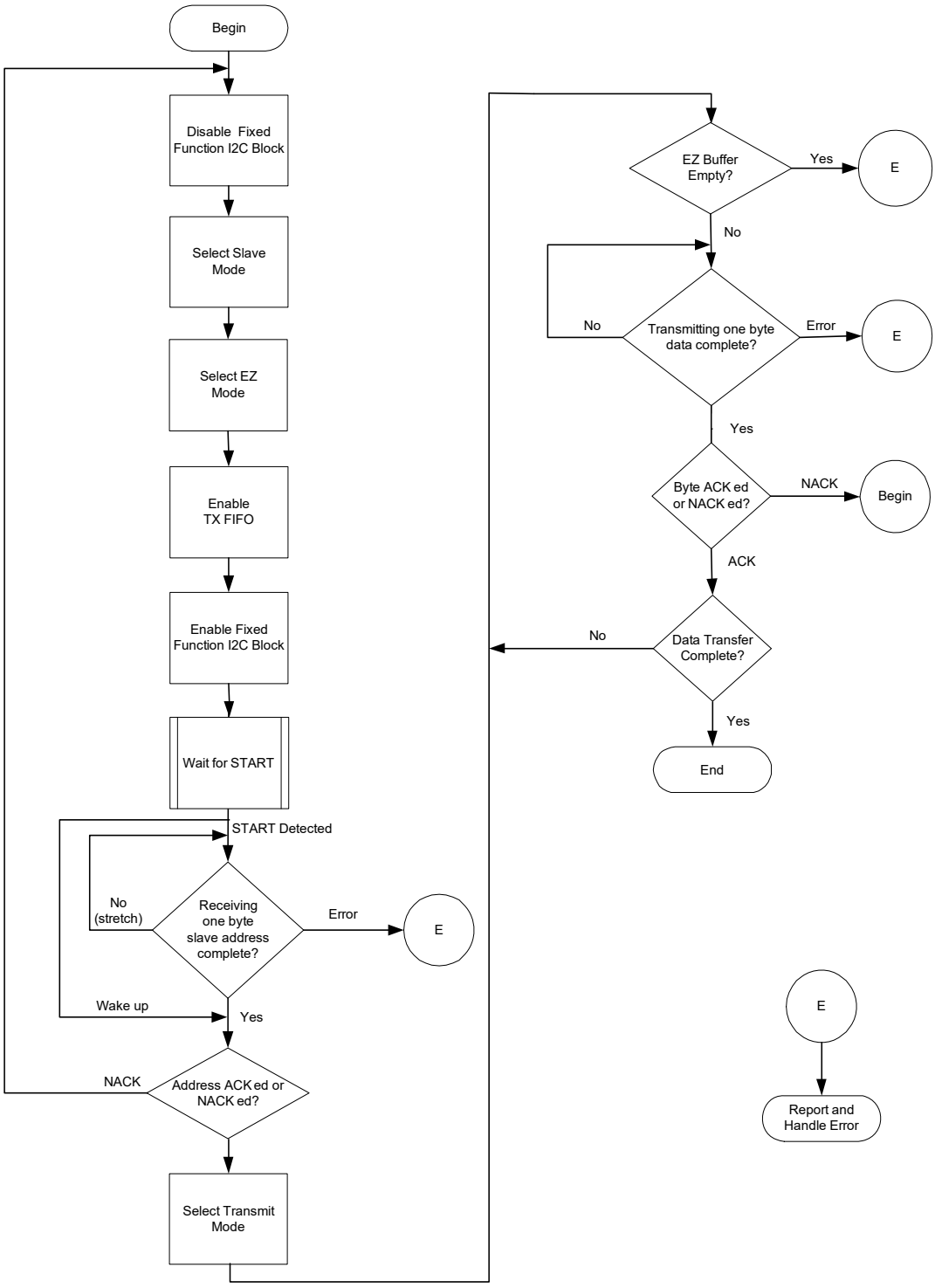


17.4.13 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

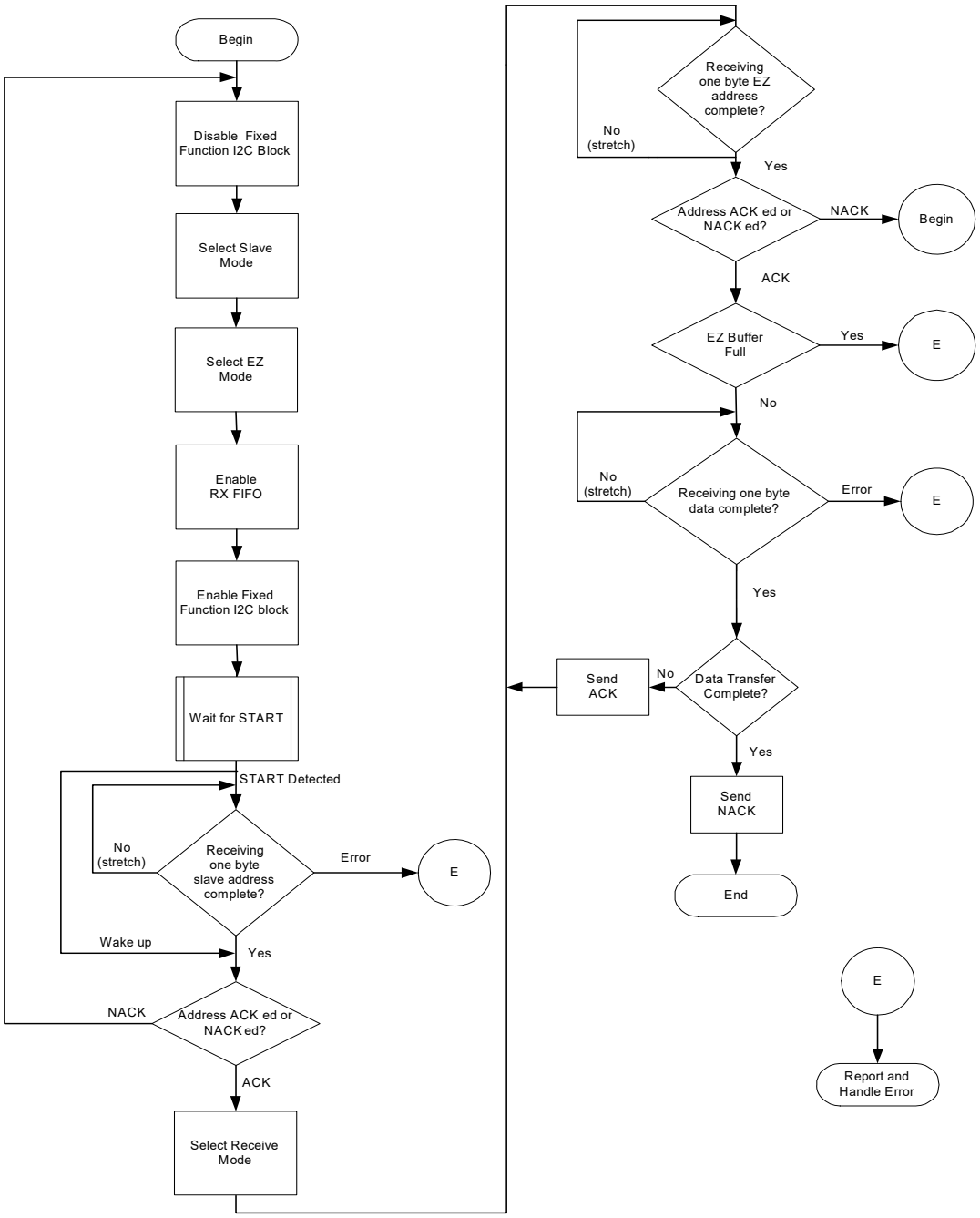
17.4.13.1 EZ Slave Transmit

Figure 17-30. EZI2C Slave Mode Write Operation Flowchart



17.4.13.2 EZ Slave Receive

Figure 17-31. EZI2C Slave Mode Read Operation Flowchart

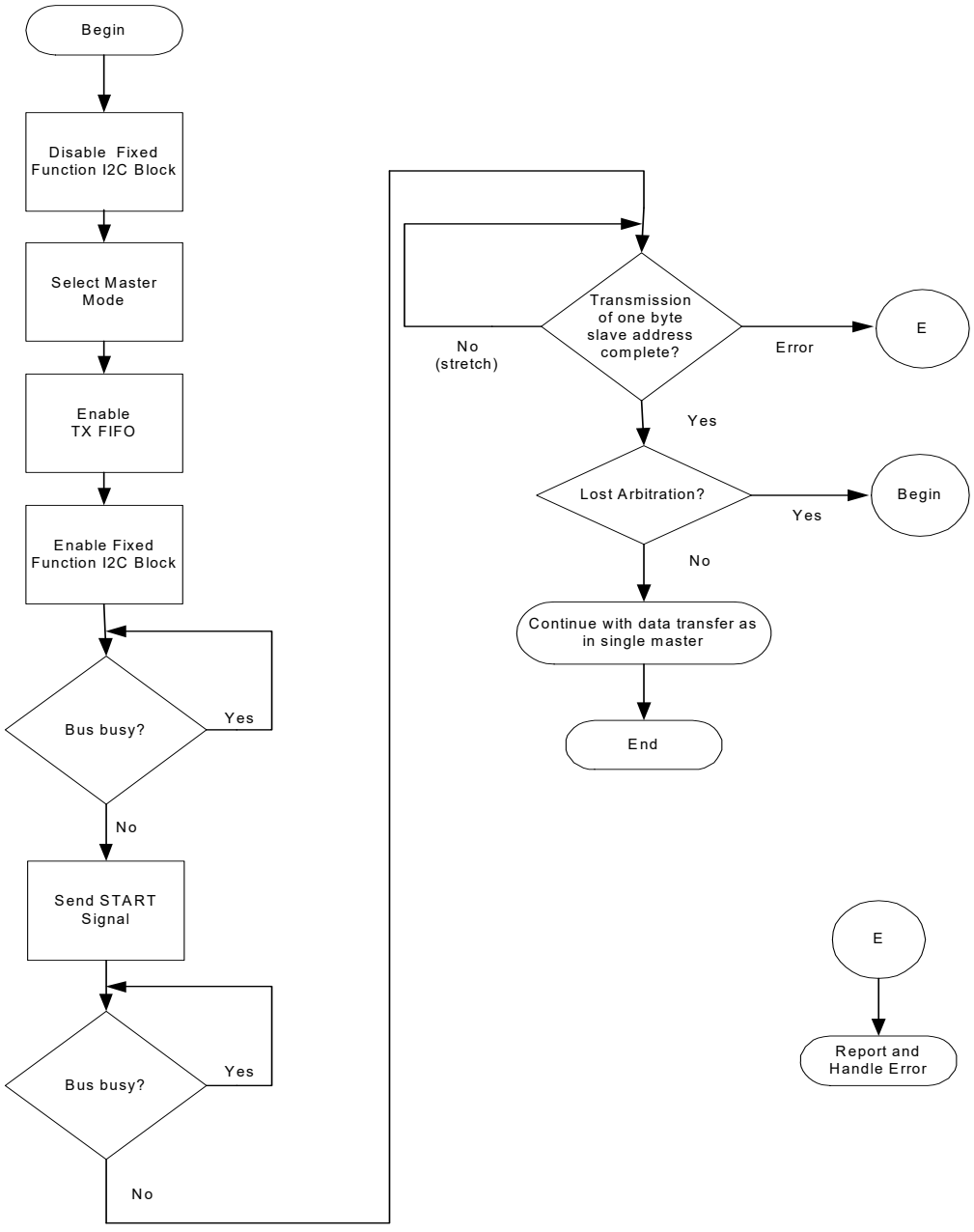


17.4.14 Multi-Master Mode Transfer Example

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

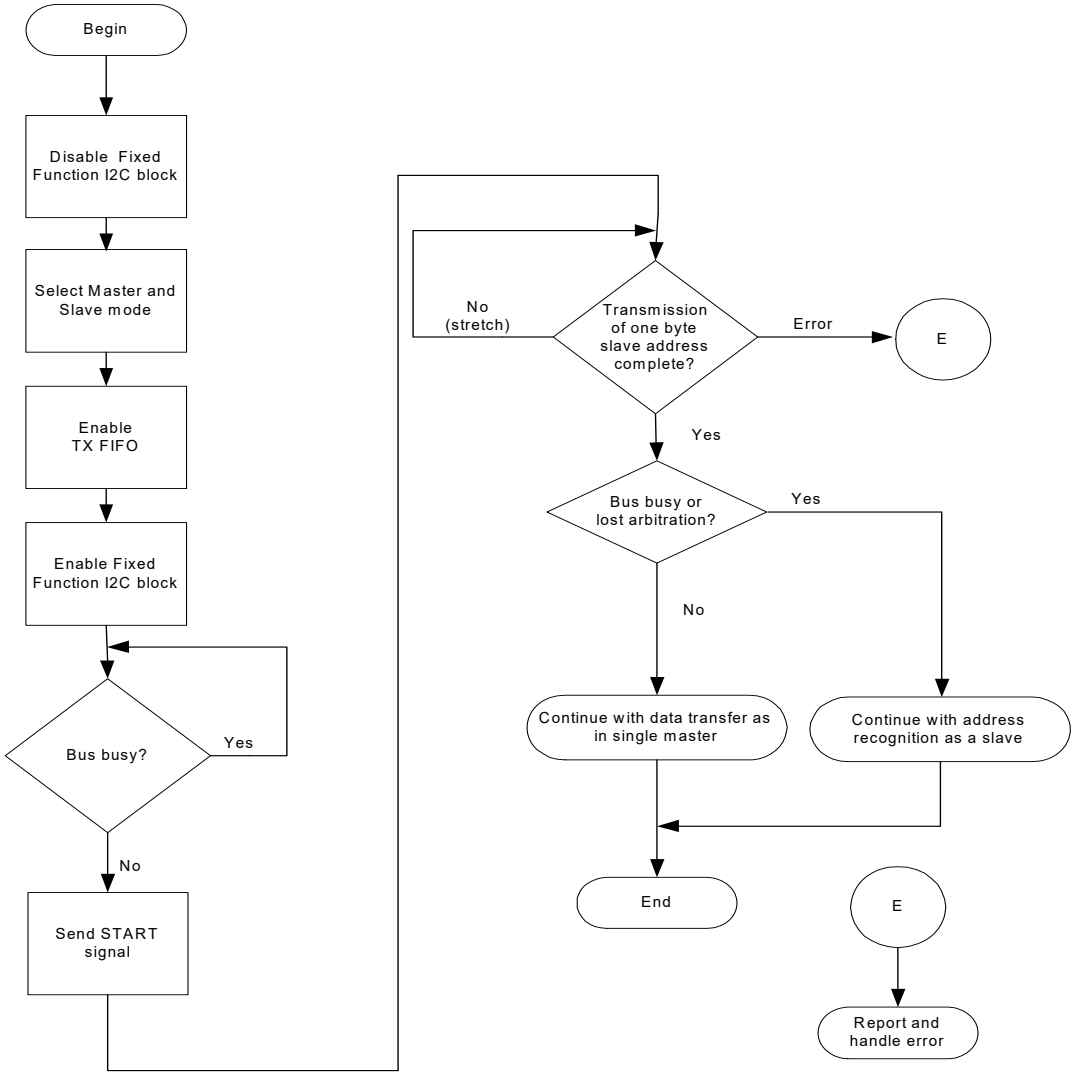
17.4.14.1 Multi-Master - Slave Not Enabled

Figure 17-32. Multi-Master, Slave Not Enabled Flowchart



17.4.14.2 Multi-Master - Slave Enabled

Figure 17-33. Multi-Master, Slave Enabled Flowchart



18. Timer, Counter, and PWM (TCPWM)

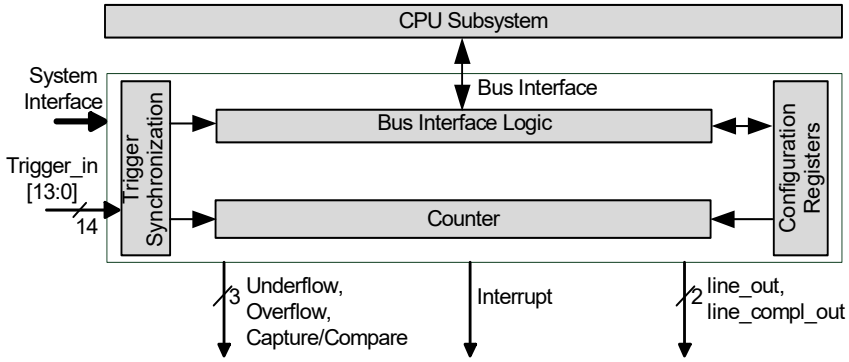
The Timer, Counter, Pulse-Width Modulator (TCPWM) block in PSoC 4 implements the 16-bit timer, counter, PWM, and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular event occurs (counter), generate PWM signals, or decode quadrature signals. This chapter explains the features, implementation, and operational modes of the TCPWM block.

18.1 Features

- Up to eight 16-bit timers, counters, or PWMs
- 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 (TC) – The final value in the counter register is reached
  - Capture/Compare (CC) – The count is captured to the CC register or the counter value equals the compare value
- Underflow, overflow, and CC output signals that can be routed to trigger SAR ADC
- Complementary line output for PWMs
- Selectable start, reload, stop, count, and capture event signals for the TCPWM from other TCPWM's underflow, compare match or overflow signal, SAR ADC's EOC or Sample_Done signal, CTBm comparator output, LPCOMP, or an external clock (EXCO) output signal and from the dedicated GPIOs with rising edge, falling edge, both edges, and level trigger options

18.2 Block Diagram

Figure 18-1. TCPWM Block Diagram



TCPWM 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) to dedicated GPIOs.
- Interrupts: Provides interrupt request signals from the counter, based on TC or CC conditions.
- System Interface: Consists of control signals such as clock and reset from the SRSS.

The TCPWM block can be configured by writing to the TCPWM registers. See "TCPWM Registers" on page 186 for more information on all registers required for this block.

18.2.1 Enabling and Disabling Counter in a TCPWM Block

A 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).

18.2.2 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 18-1.

Table 18-1. 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.

18.2.3 Events Based on Trigger Inputs

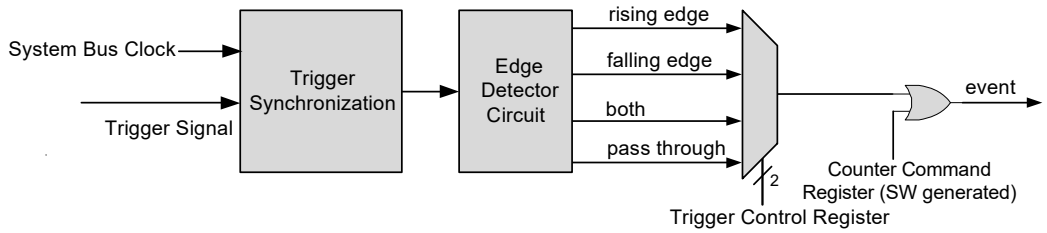
The following events are triggered by hardware or software:

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

Hardware triggers can be level signal, rising edge, falling edge, or both edges. Figure 18-2 shows the selection of edge detection type for any event trigger signal.

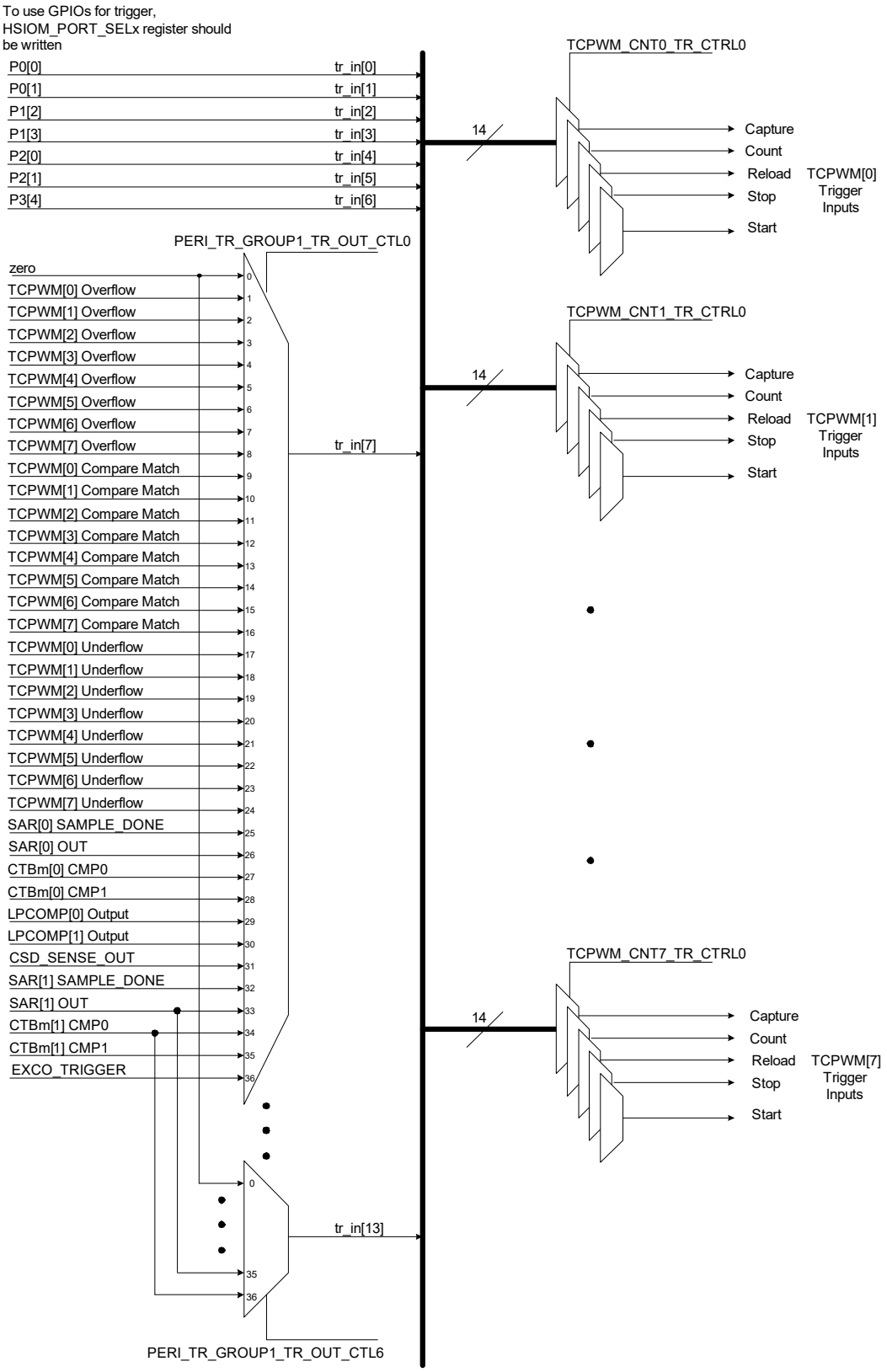
Any edge (rising, falling, or both) or level (HIGH or LOW) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNT_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 18-2.

Figure 18-2. Trigger Signal Edge Detection



The trigger signal to generate an event can be a GPIO signal, TCPWM's underflow, compare match or overflow signal, SAR ADC's EOC or Sample_Done signal, CTBm comparator output, or a LPCOMP output signal. Figure 18-3 shows the trigger signal selection for all events.

Figure 18-3. Trigger Mux in PSoC 4500S



The trigger signal to generate an event can be a GPIO signal, TCPWM's underflow, compare match or overflow signal, SAR ADC's EOC or Sample_Done signal, CTBm comparator output, LPCOMP, or external clock (EXCO) output signal.

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 and DOWN counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with '0'.
  - 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 175.
- **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 CC and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

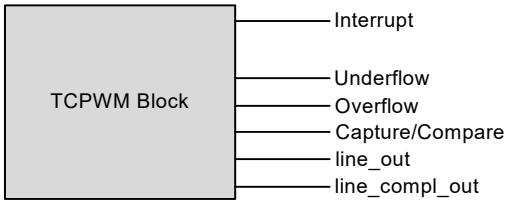
Notes:

- 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.

18.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 18-4.

Figure 18-4. TCPWM Output Signals



18.2.4.1 Signals upon Trigger Conditions

- Counter generates an internal overflow condition when counting up and the count register reaches the period value.
- Counter generates an internal underflow condition when counting down and the count register reaches zero.
- The 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.

18.2.4.2 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 18-2.

Table 18-2. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNT_INTR (Interrupt request register)	0	TC	This bit is set to '1', when a TC is detected. Write '1' to clear this bit.
	1	CC_MATCH	This bit is set to '1' when the counter value matches CC 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.
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.

18.2.4.3 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 (see Table 18-3).

Table 18-3. 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 an overflow 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 an underflow event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	

18.2.5 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 18-4 for details.

Table 18-4. 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.

18.3 Operation Modes

The counter block can function in six operational modes, as shown in Table 18-5. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

Table 18-5. 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 18-6.

Table 18-6. Counting Mode Configuration

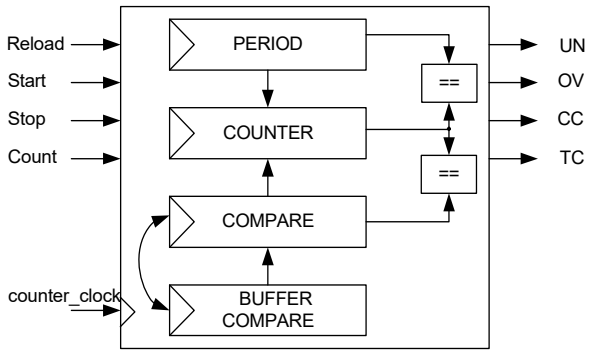
Counting Modes	UP_DOWN_MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. TC condition is generated when the counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until '0' is reached. A TC condition is 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.

18.3.1 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.

18.3.1.1 Block Diagram

Figure 18-5. Timer Mode Block Diagram



18.3.1.2 How it Works

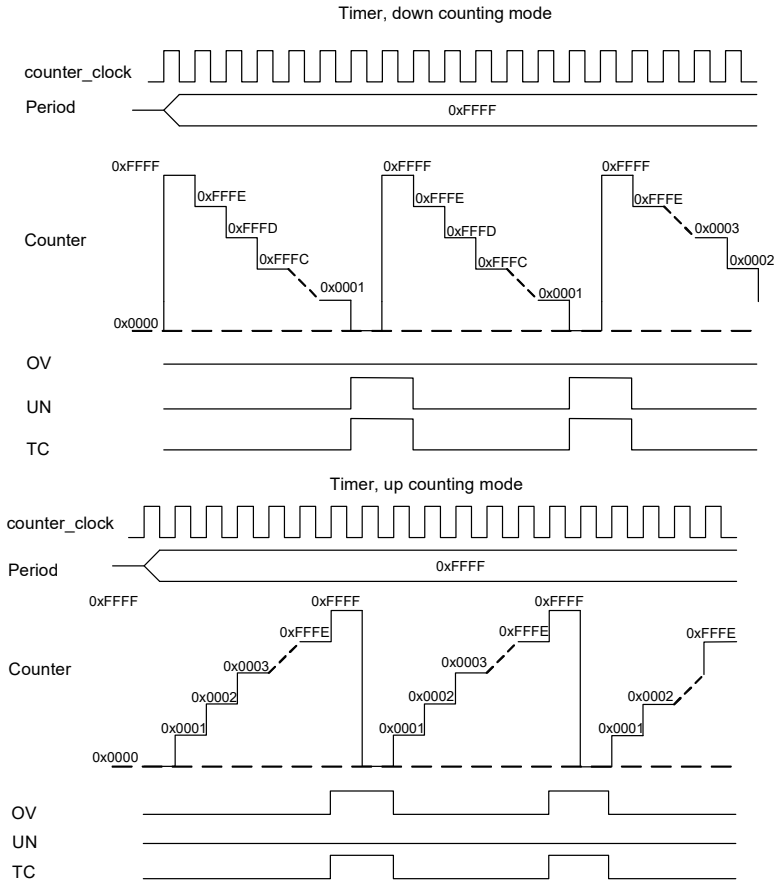
The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode. The following explains the working of the timer:

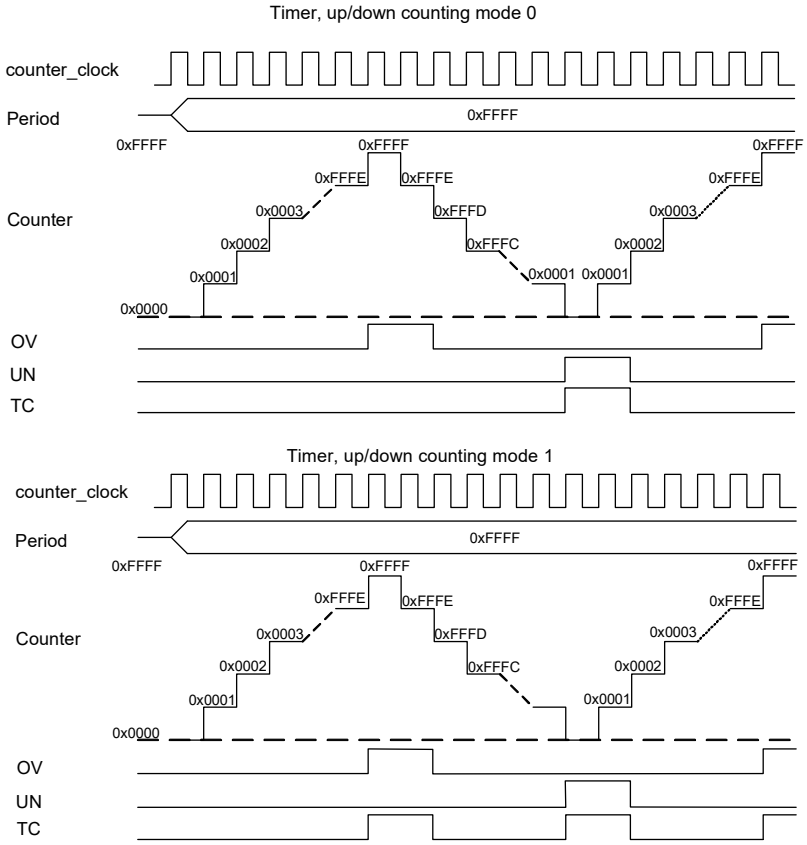
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
    Note: It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The 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 '0', 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 18-6 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 18-6. 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 169. The figures in this chapter assume that HFCLK and counter clock are the same.

18.3.1.3 Configuring Counter for Timer Mode

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

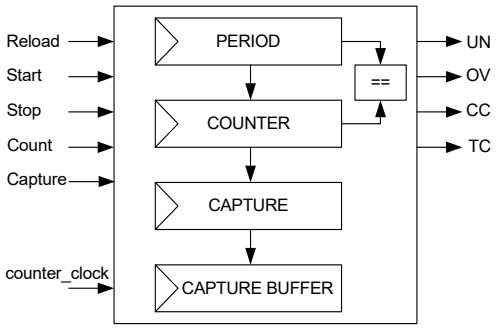
1. Disable the counter by writing '0'' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNT_CC 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 18-1.
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 18-6.
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 170.
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.

18.3.2 Capture Mode

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

18.3.2.1 Block Diagram

Figure 18-7. Capture Mode Block Diagram



18.3.2.2 How it Works

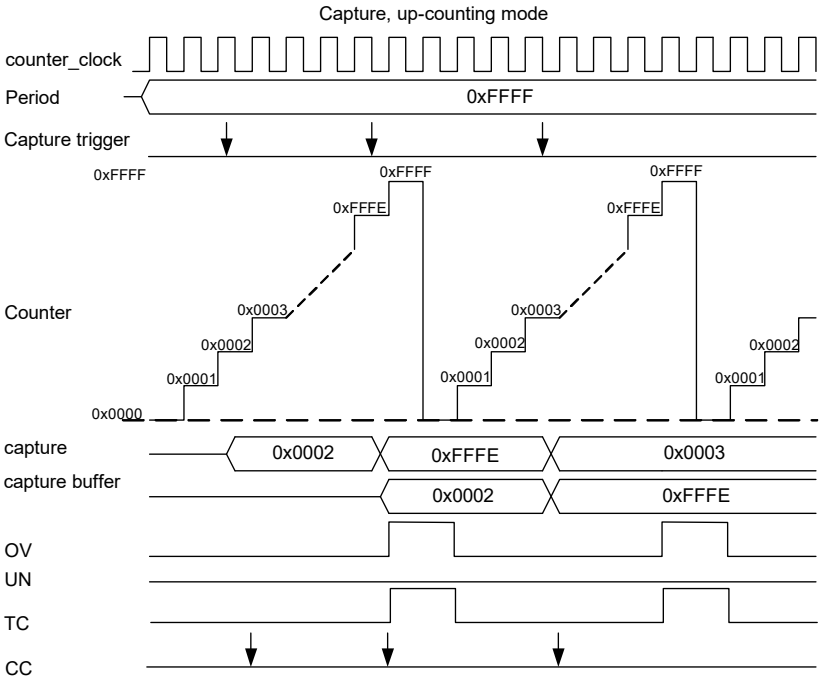
The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNT_CTRL).

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 18-8 illustrates the capture behavior in the up-counting mode.

Figure 18-8. Timing Diagram of Counter in Capture Mode, Up-counting Mode



In the figure, observe that:

- The period register contains the maximum count value.
- Internal 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.

18.3.2.3 Configuring Counter for Capture Mode

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

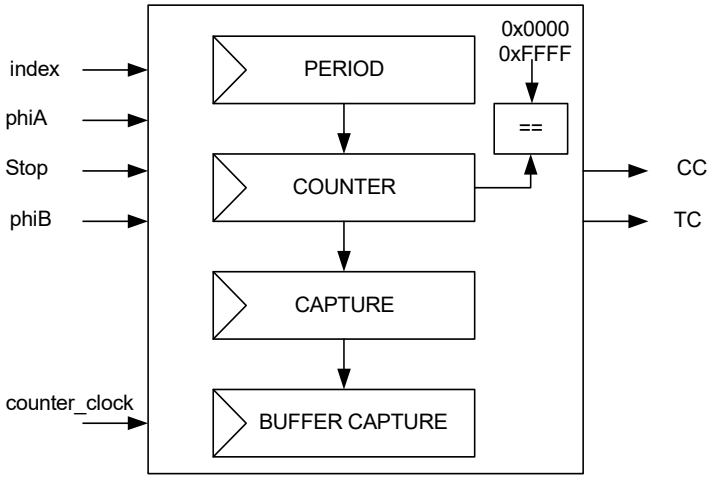
1. Disable the counter by writing ‘0’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing ‘010’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 18-1.
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 18-6.
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 170.
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.

18.3.3 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.

18.3.3.1 Block Diagram

Figure 18-9. Quadrature Mode Block Diagram



18.3.3.2 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.

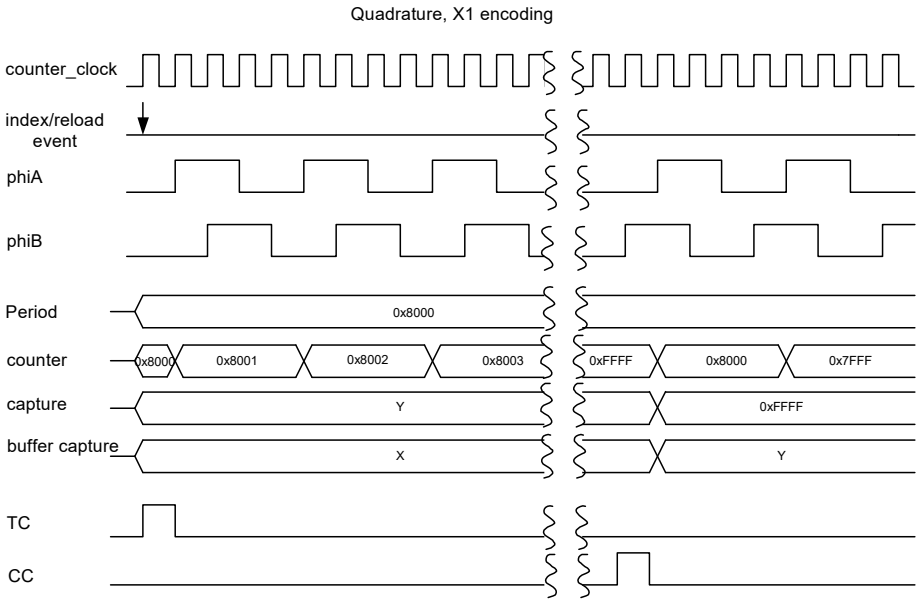
The Quadrature mode operation occurs as follows:

- 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 18-10.

  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 the condition that 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 18-10 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.
  - TC 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).

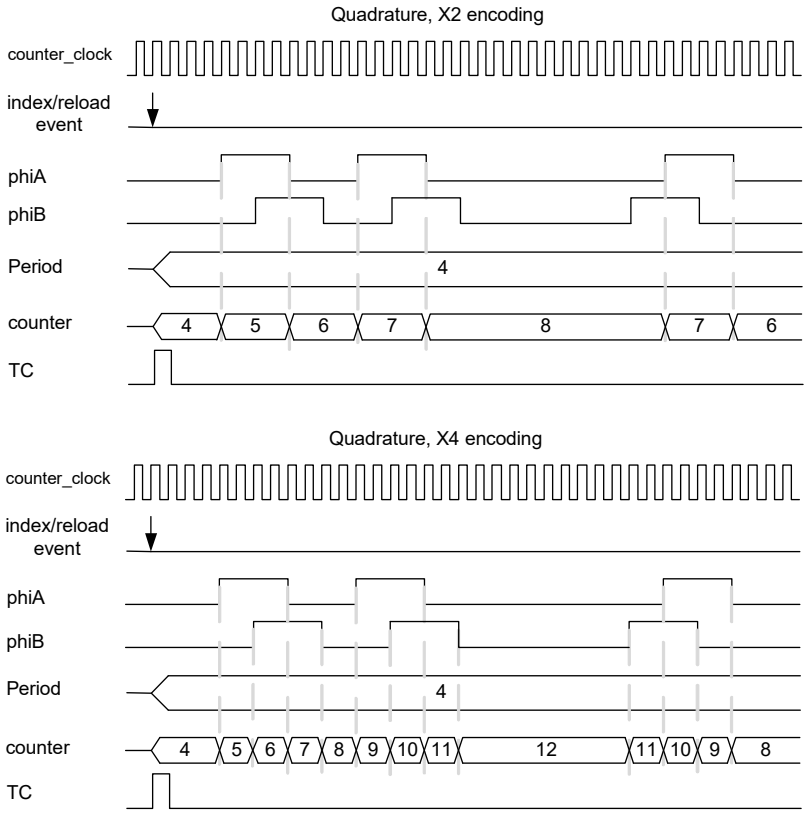
Figure 18-10. Timing Diagram for Quadrature Mode, X1 Encoding



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 18-11 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

Figure 18-11. Timing Diagram for Quadrature Mode, X2 and X4 Encoding



18.3.3.3 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 170.
8. Enable the counter by writing ‘1’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.

18.3.4 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

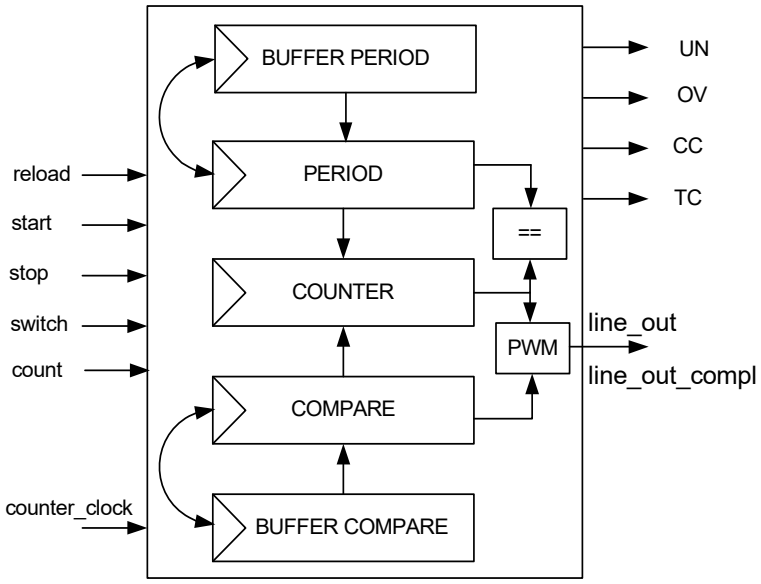
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

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

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

18.3.4.1 Block Diagram

Figure 18-12. PWM Mode Block Diagram



18.3.4.2 How it Works

The PWM mode can be 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 18-6.

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 following are recommended ways to modify the duty cycle:

- 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 will not reflect the register update, as shown in Figure 18-14.

In the center-aligned mode, the output line is set to ‘0’ at TC 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 the compare register value to generate the CC signal on match.

Figure 18-13 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 18-13. Timing Diagram for Center-aligned PWM

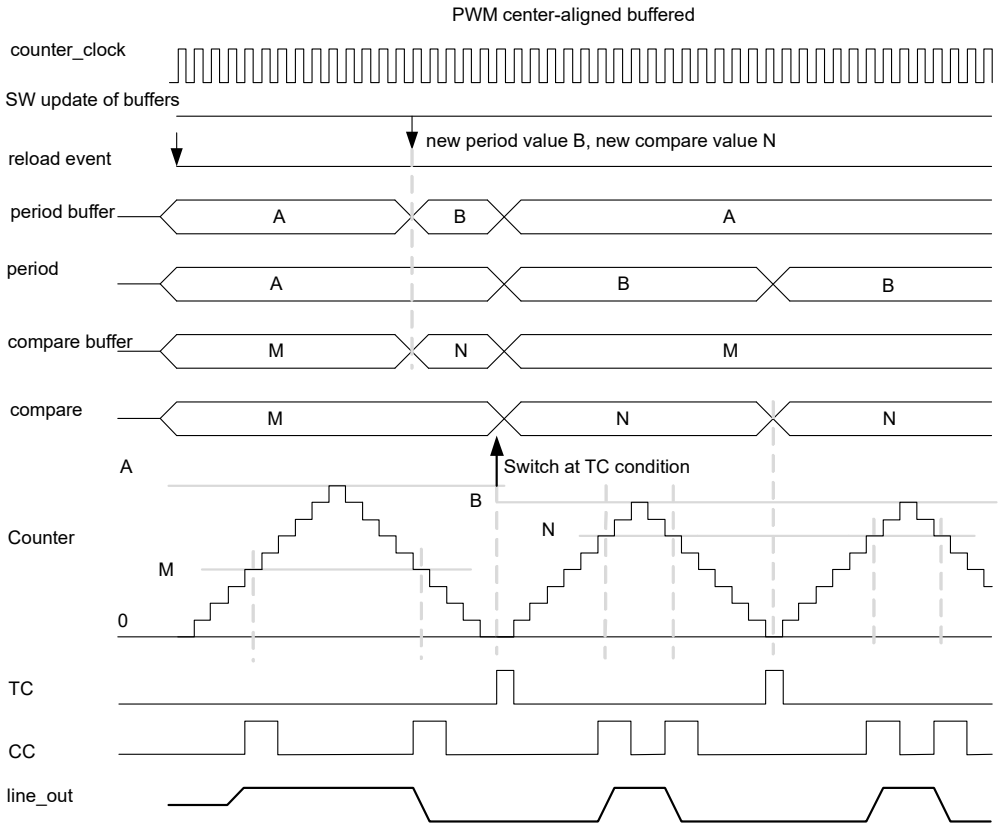
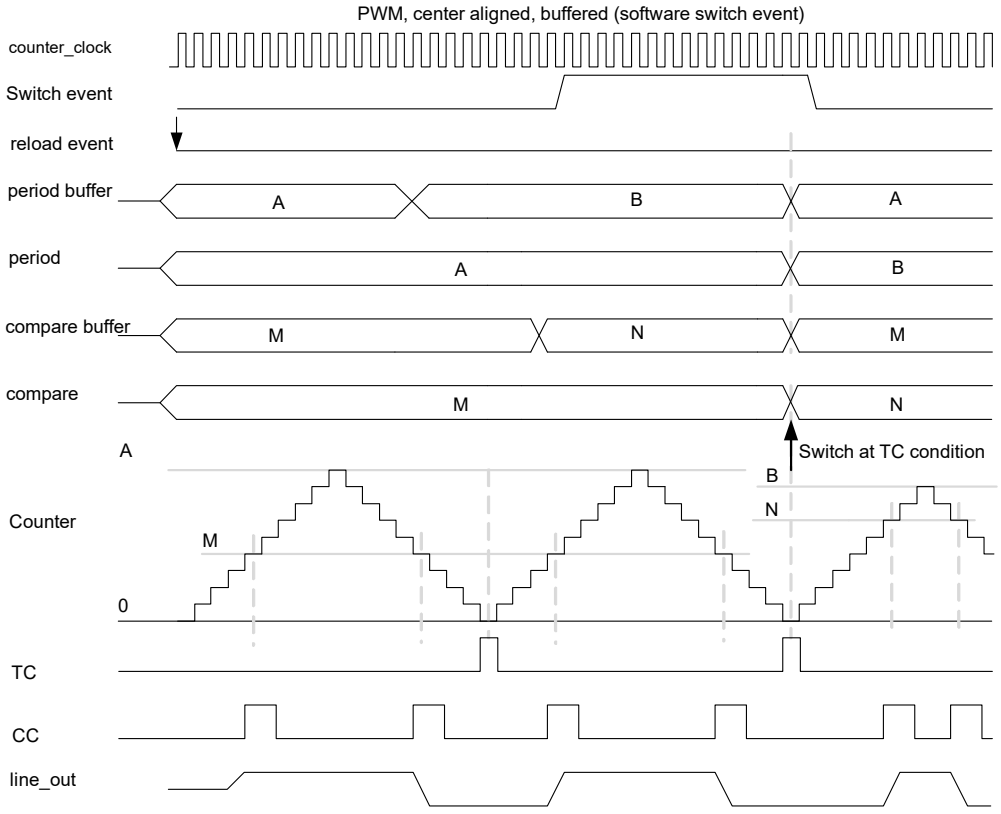


Figure 18-13 illustrates center-aligned PWM with software generated switch events:

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

Figure 18-14. Timing Diagram for Center-aligned PWM (software switch event)



18.3.4.3 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 18-3.
- 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 18-3.

18.3.4.4 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 18-7.

Table 18-7. 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 still runs.
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 18-8.

Table 18-8. 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 the kill input is removed, the kill event should be asynchronous (see Table 18-8). 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.

18.3.4.5 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:

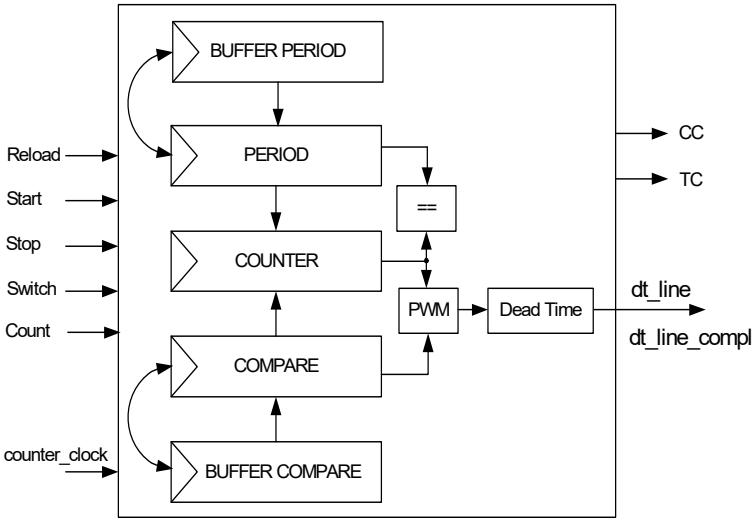
1. Disable the counter by writing ‘0’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing ‘100’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 18-1.
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 18-6.
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 170.
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.

18.3.5 PWM 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.

18.3.5.1 Block Diagram

Figure 18-15. PWM Dead Time Mode Block Diagram



18.3.5.2 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.

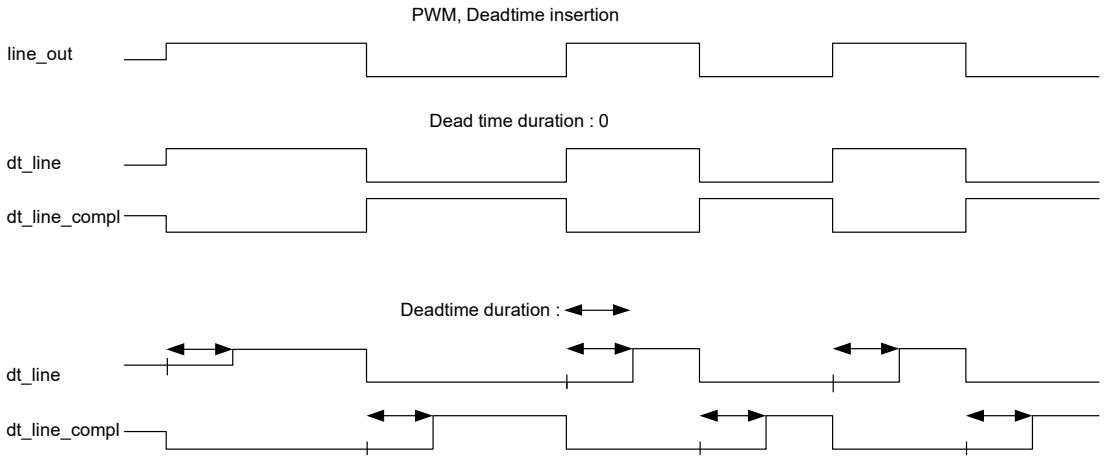
This mode follows the PWM mode and supports the following features available with that mode:

- Various output alignment modes
- Two complementary output lines, dt_line and dt_line_compl, are derived from the 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 18-16 illustrates how the complementary output lines "dt_line" and "dt_line_compl" are generated from the PWM output line, "line_out".

Figure 18-16. Timing Diagram for PWM, with and without Dead Time



18.3.5.3 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:

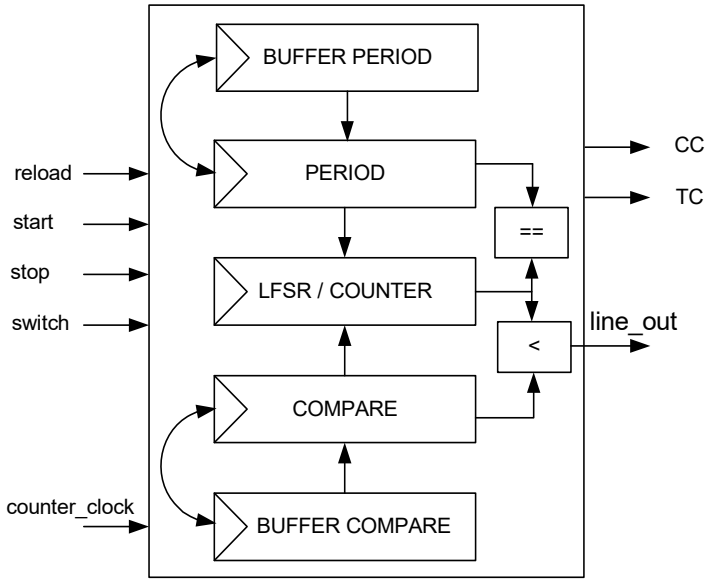
1. Disable the counter by writing ‘0’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing ‘101’ to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 18-1.
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 18-6.
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 178.
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 170.
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.

18.3.6 Pulse-Width Modulation Pseudo-Random Mode (PWM_PR)

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

18.3.6.1 Block Diagram

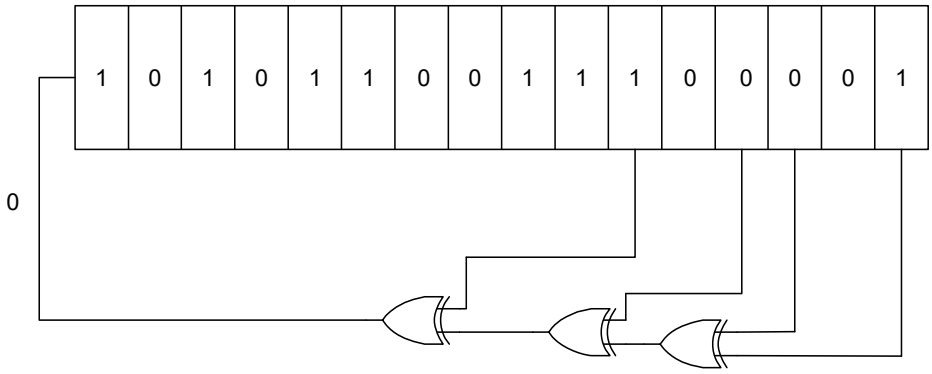
Figure 18-17. PWM Pseudo-Random Mode Block Diagram



18.3.6.2 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 18-18. It generates all numbers in the range [1, 0xFFFF] in a pseudo-random sequence. Note that the counter register should be initialized with a non-zero value.

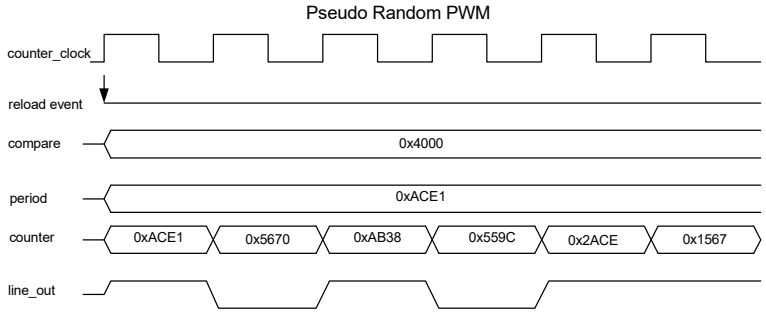
Figure 18-18. Pseudo-Random Sequence Generation using Counter Register



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 lesser 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.
- TC 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 TC, the counter is stopped by hardware.
- In this mode, UN, OV, and trigger condition events do not occur.
- CC condition occurs when the counter is running and its value equals compare value. Figure 18-19 illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50% duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

Figure 18-19. 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 TC.

18.3.6.3 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 170.
10. Enable the counter by writing ‘1’ to the COUNTER_ENABLED field of the TCPWM_CTRL register.

18.4 TCPWM Registers

Table 18-9. 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

19. LCD Direct Drive

The PSoC 4 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.

19.1 Features

The PSoC 4 LCD segment drive block has the following features:

- Supports up to 49 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 four drive methods:
  - PWM at 1/2 bias
  - PWM at 1/3 bias
  - PWM at 1/4 bias
  - PWM at 1/5 bias
- Operates in active and sleep mode
- Digital contrast control

19.2 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:

- **V_{RMSOFF}**: The voltage that the LCD driver can realize on segments that are intended to be OFF.
- **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 4). PSoC 4 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 4 supports two 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.

19.2.1 Drive Modes

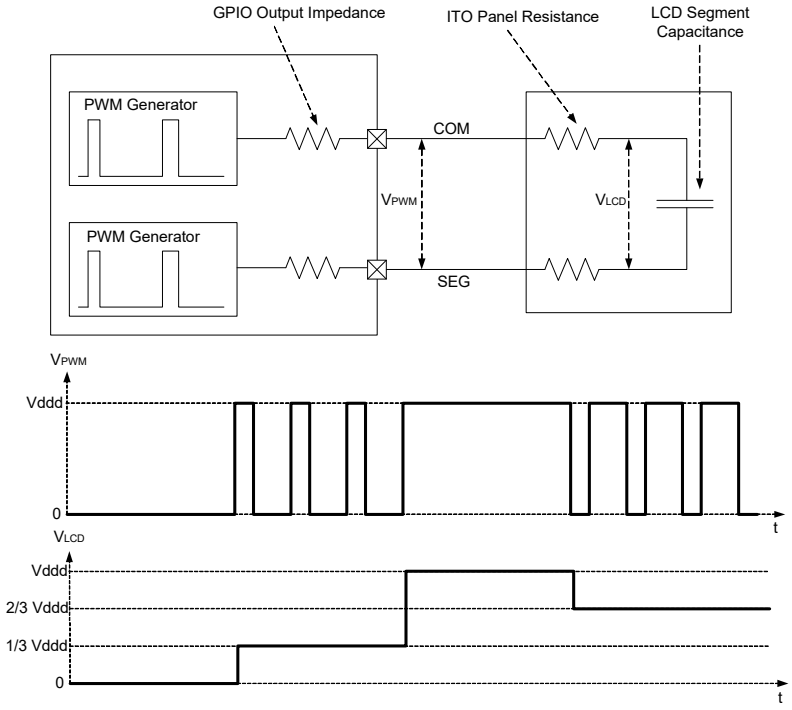
The PSoC 4 supports the following drive modes:

- PWM drive at 1/2 bias
- PWM drive at 1/3 bias
- PWM drive at 1/4 bias
- PWM drive at 1/5 bias

19.2.1.1 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, as illustrated in Figure 19-1.

Figure 19-1. 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 19-1. 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 the IMO clock (high-speed operation). The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 19-2 and Figure 19-3 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 19-4 and Figure 19-5 illustrate the Type-A and Type-B waveforms for COM and SEG electrodes for 1/3 bias and 1/4 duty.

Figure 19-2. PWM1/2 Type-A Waveform Example

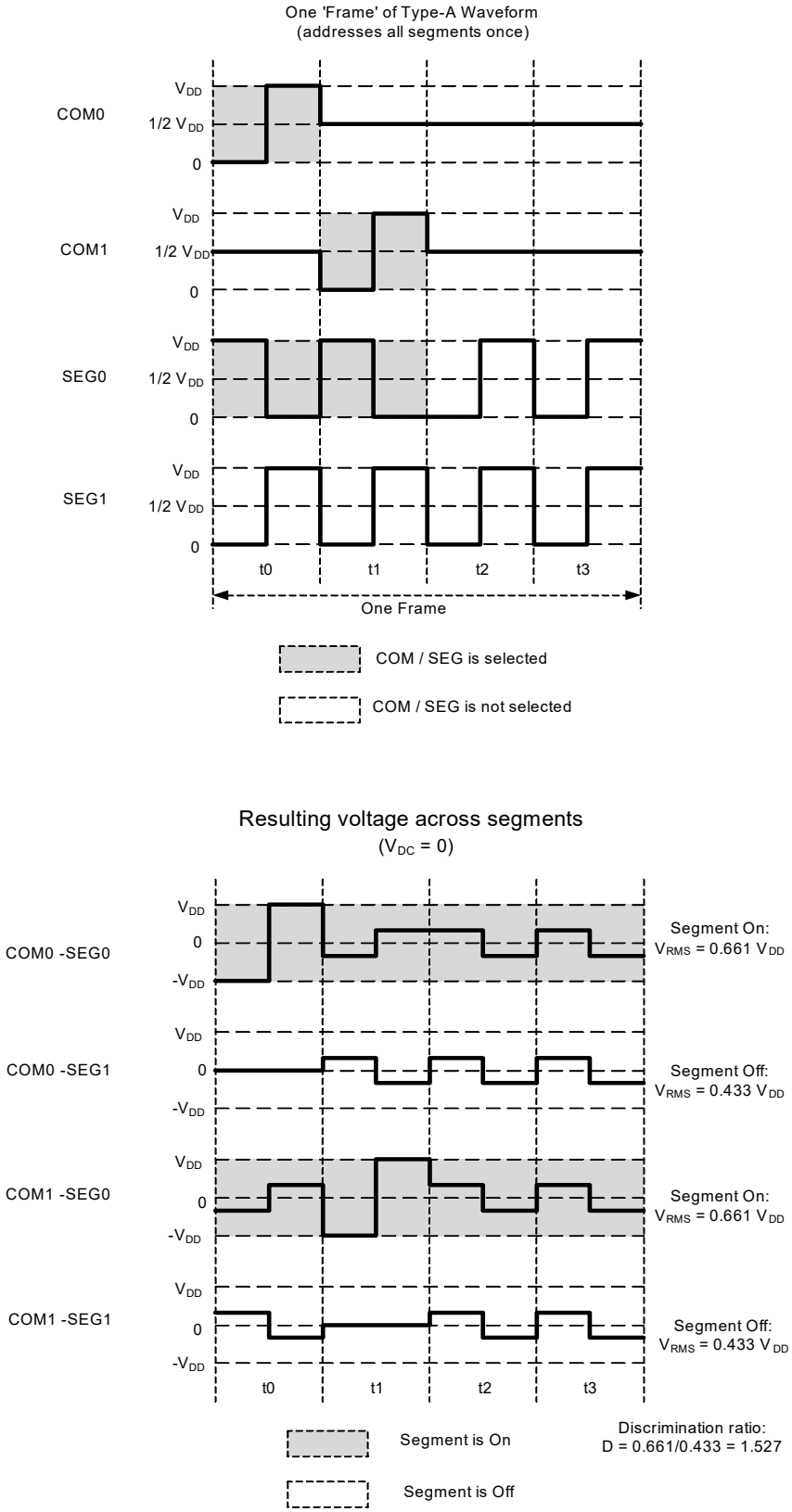


Figure 19-3. PWM1/2 Type-B Waveform Example

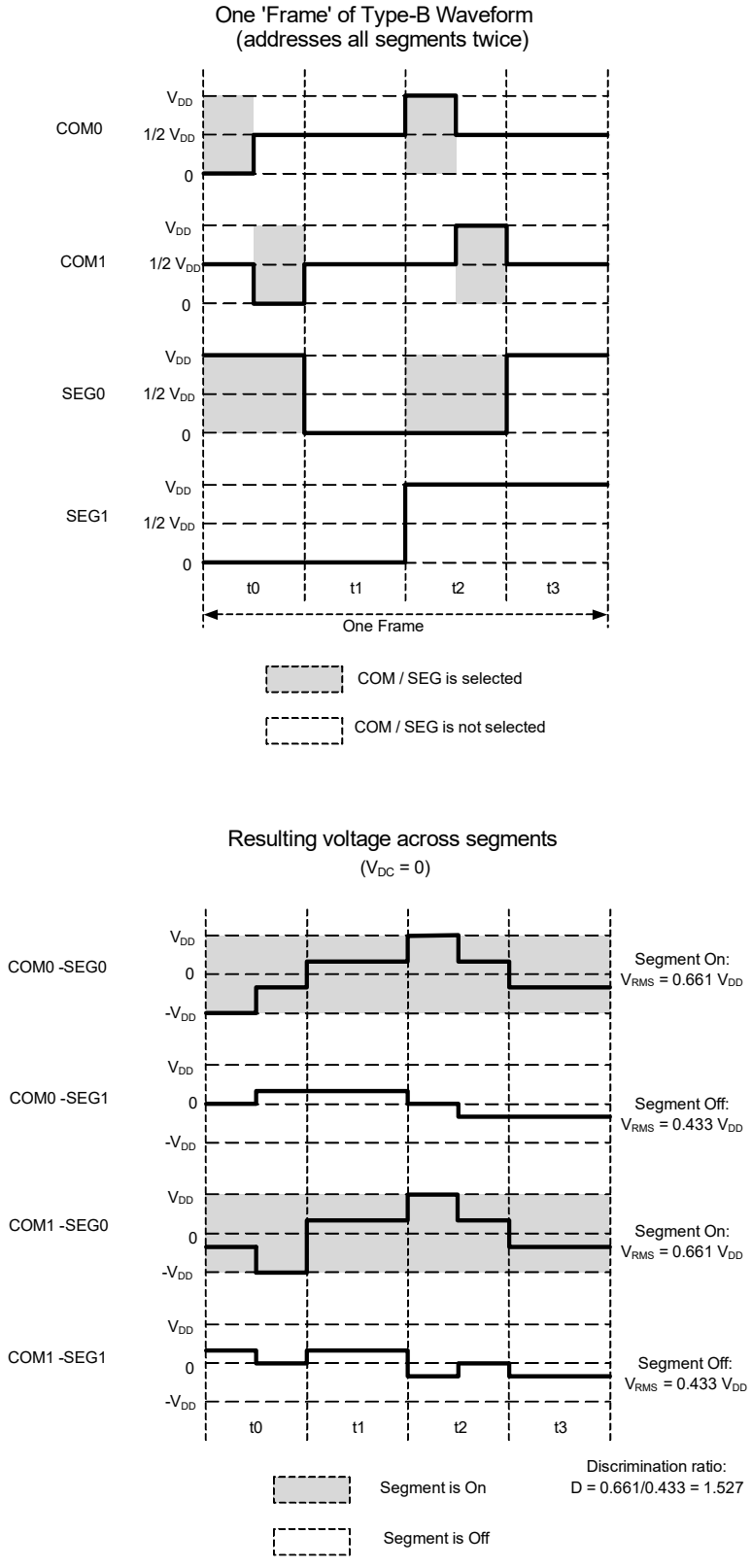


Figure 19-4. PWM1/3 Type-A Waveform Example

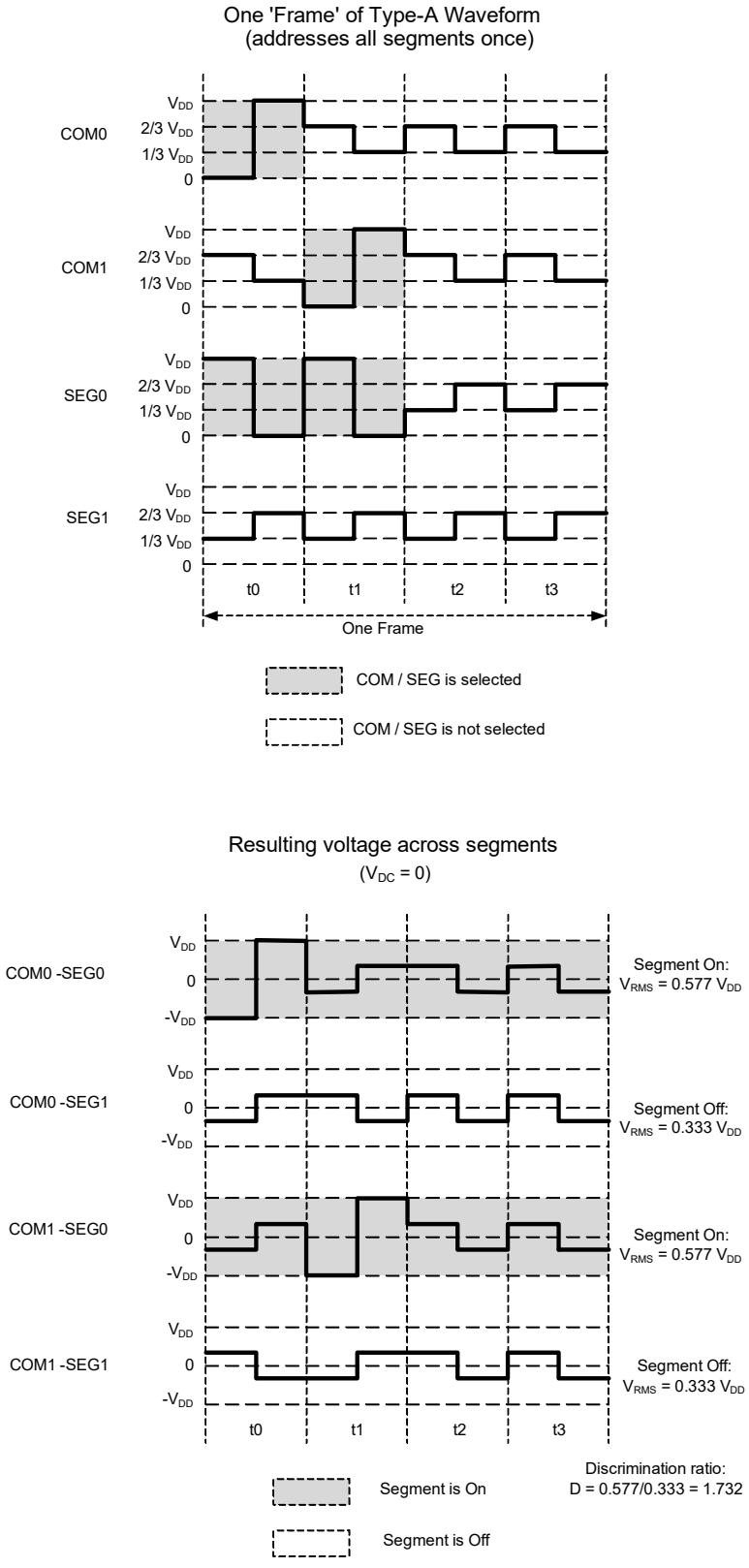
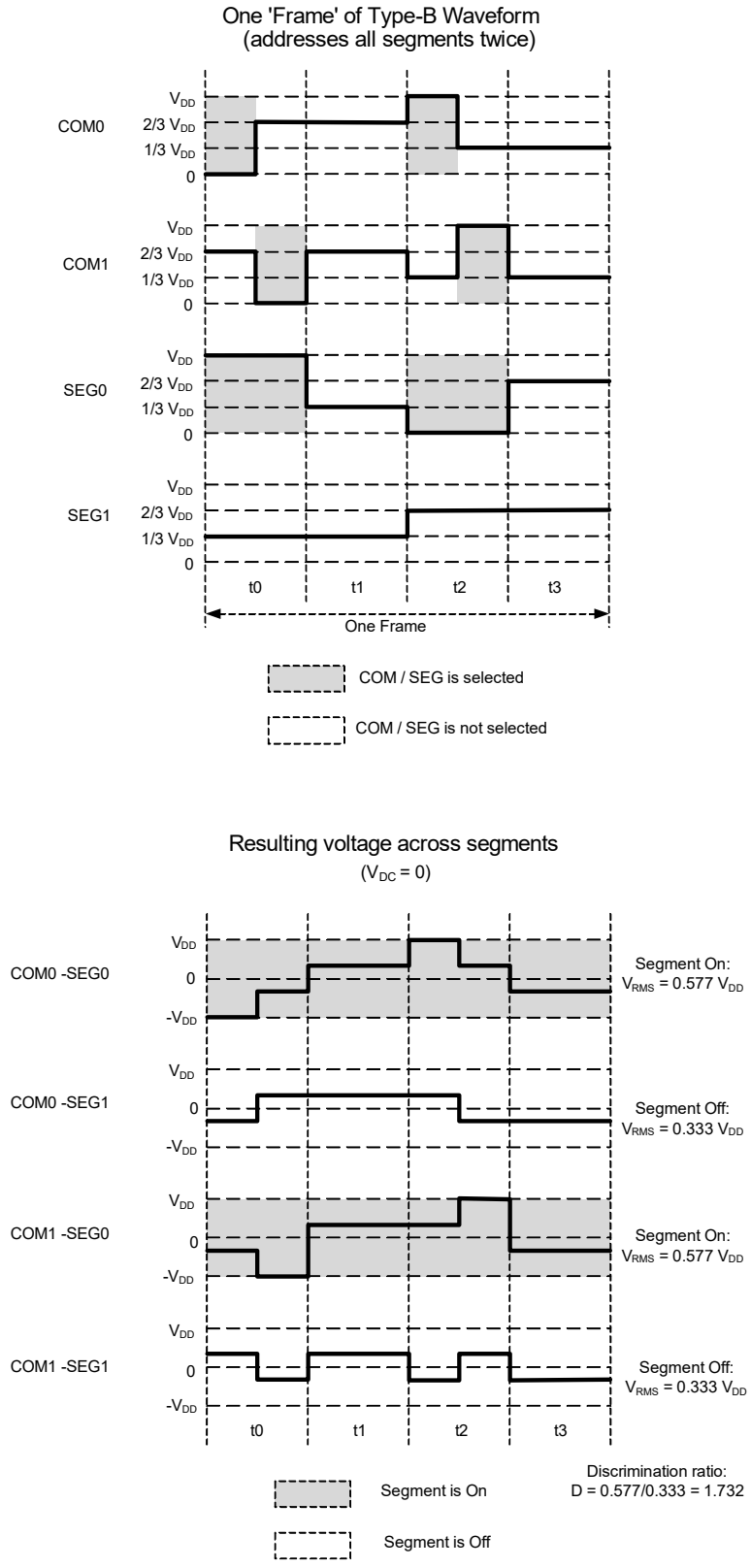


Figure 19-5. PWM1/3 Type-B Waveform Example



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 19-1

$$V_{RMS(ON)} = \sqrt{\frac{2B^2 + 2(M-1)}{2M}} x \left(\frac{V_{DRV}}{B}\right)$$

Equation 19-2

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. They offer better discrimination ratio especially when used in designs with more than 4 COM drivers.

The minimum PWM frequency depends on the capacitance of the display and the internal LCD ITO resistance of the ITO routing traces. In most designs, the minimum PWM frequency required to provide good contrast is 1 MHz.

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 19-2 and Figure 19-3.

19.2.2 Recommended Bias Settings

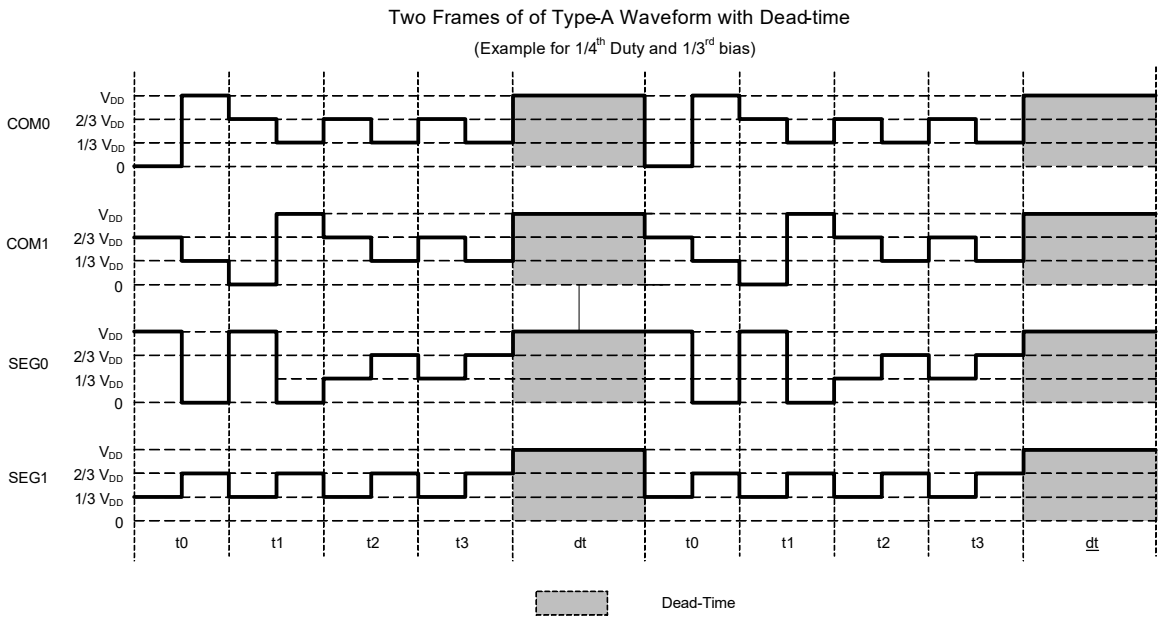
Table 19-1. Recommended Bias Settings

Display Type	Bias Setting
Four COM TN Glass	PWM 1/3 bias
Four COM STN Glass	PWM 1/3 bias
Eight and 16 COM, STN	PWM 1/4 bias and 1/5 bias

19.2.3 Digital Contrast Control

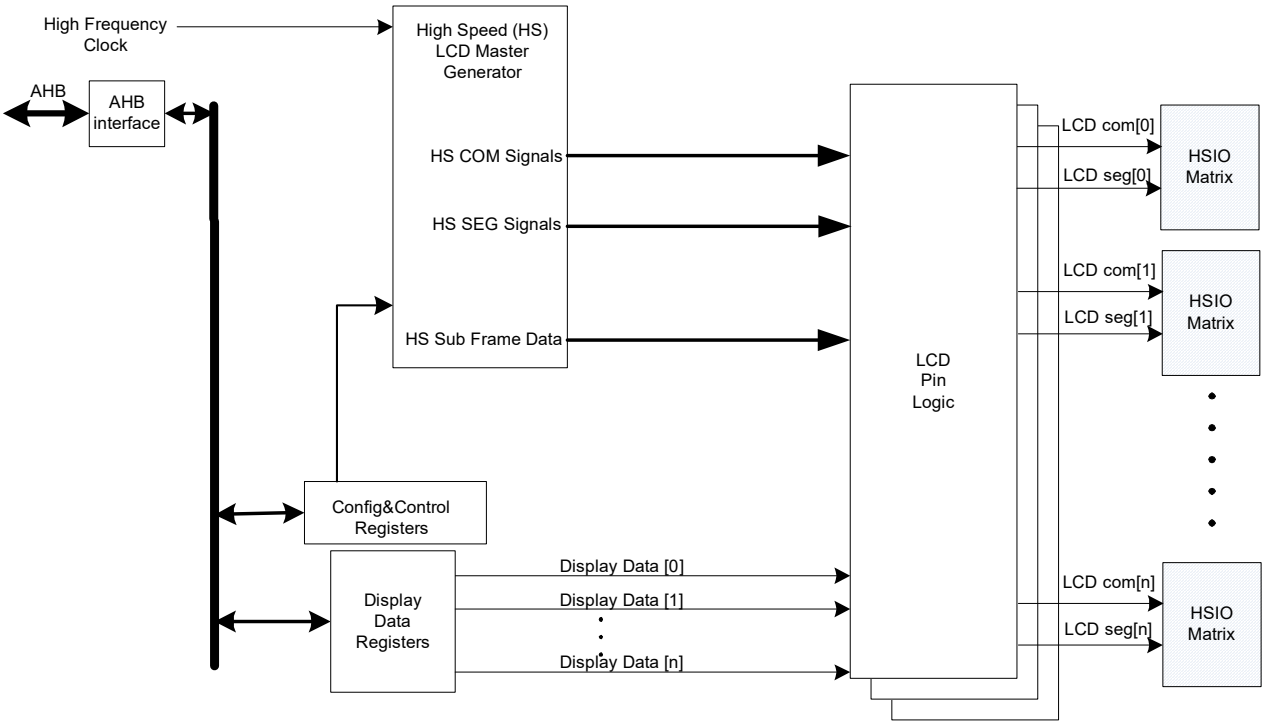
In all drive modes, digital contrast control can be used to change the contrast level of the segments. This method reduces contrast by reducing the driving time of the segments. This is done by inserting a Dead-Time interval after each frame. During dead time, all COM and SEG signals are driven to a logic 1 state. The dead time can be controlled in fine resolution. Figure 19-6 illustrates the dead-time contrast control method for 1/3 bias and 1/4 duty implementation.

Figure 19-6. Dead-Time Contrast Control



19.3 Block Diagram

Figure 19-7. Block Diagram of LCD Direct Drive System



19.3.1 How it Works

The LCD pin logic block routes the COM and SEG outputs from the LCD waveform generators to the corresponding I/O matrices. Any GPIO can be used as either a COM or SEG. This configurable pin assignment for COM or SEG is implemented in the GPIO and I/O matrix; see High-Speed I/O Matrix (HSIOM) on page 70.

The LCD controller operates in two device power modes: Active and Sleep. The LCD controller should not be enabled in Deep Sleep mode because operation is not possible without the IMO. The LCD controller is unpowered in Hibernate and Stop modes.

19.3.2 High-Speed Master Generator

The 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, 4, and 8.
- Operating mode configuration bits enabled to select one of the following:
  - PWM 1/2 bias
  - PWM 1/3 bias
  - PWM 1/4 bias
  - PWM 1/5 bias
  - Off/disabled

  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 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.

19.3.3 LCD Pin Logic

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.

19.3.4 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 19-2. 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 19-2. 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

19.4 Register List

Table 19-3. 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.

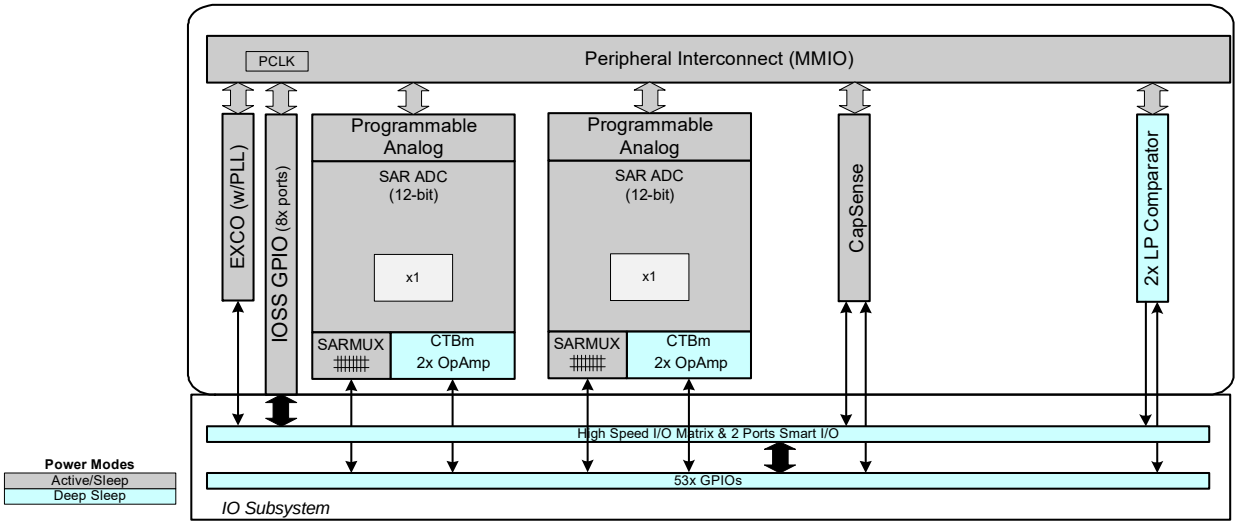
Section E: Analog System

This section comprises the following chapters:

- SAR ADC chapter on page 197
- Low-Power Comparator chapter on page 227
- CapSense chapter on page 233
- Continuous Time Block mini (CTBm) chapter on page 234
- Temperature Sensor chapter on page 245

Top-Level Architecture

Analog System Block Diagram



20. SAR ADC

The PSoC 4 has two successive approximation register analog-to-digital converters (SAR ADCs). Each SAR ADC is designed for applications that require moderate resolution and high data rate.

It consists of the following blocks (see Figure 20-1):

- SARMUX
- SAR ADC core
- SARREF
- SARSEQ

Every SAR ADC core is a fast 12-bit ADC with a sampling rate of 1 Msps. Preceding the SAR ADC is the SARMUX, which can route external pins and internal signals (AMUXBUS-A/-B, CTBm, temperature sensor output) to the SAR ADC. SARREF is used for multiple reference selection. The sequencer controller SARSEQ is used to control SARMUX and SAR ADC to perform an automatic scan on all enabled channels without CPU intervention and for pre-processing, such as averaging the output data.

SAR ADC has 16 individually configurable logical channels that can scan 13 unique input channels. The result from each channel is double-buffered; a complete scan may be configured to generate an interrupt at the end of the scan. The sequencer may also be configured to flag overflow, collision, and saturation errors that can be configured to assert an interrupt.

For more flexibility, it is also possible to control most analog switches, including those in the SARMUX with the firmware. This makes it possible to implement an alternative sequencer with the firmware.

20.1 Features

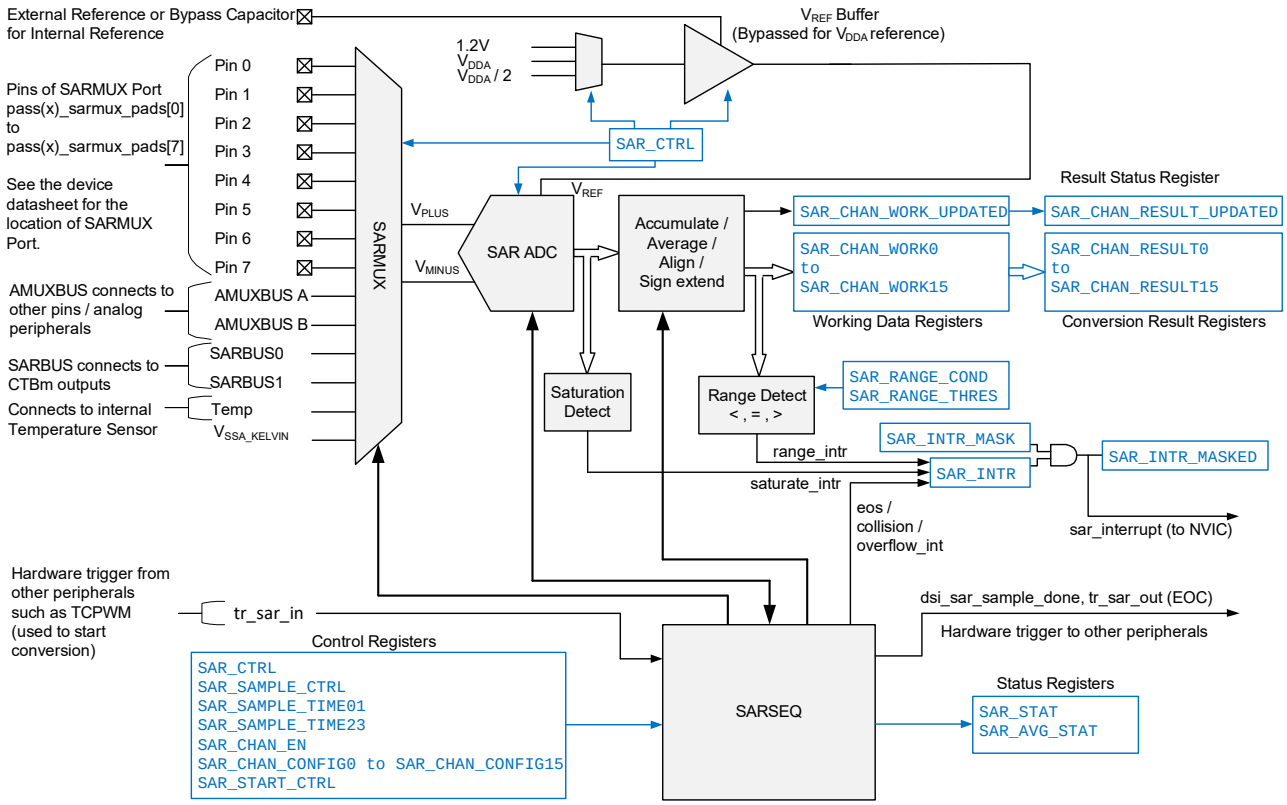
The SAR ADC block provides the following features:

- Operates across the entire device power supply range
- Provides two SAR ADCs
- Maximum 1-Msps sampling rate
- 32 individually configurable channels and two injection channels for two SAR ADCs
- Each channel has the following features:
  - Input from an external pin (only for eight channels in single-ended mode and four channels in differential mode) or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Programmable acquisition times
  - Selectable 8-, 10-, and 12-bit resolution
  - Single-ended or differential measurement
  - Averaging
  - Double-buffered results
  - Left- or right-aligned results
- Scan triggered by firmware, timer, CTBm comparator, LPCOMP, and by SAR end of conversion signal
  - Hardware/firmware trigger (one shot), and free-running (continuous conversion) modes
- Hardware averaging support
  - First order accumulate

- Samples averaging from 2 to 256 (powers of 2)
- Results represented in 16-bit sign extended values
- Selectable voltage references
  - Internal V_{DDA} and $V_{DDA}/2$ references
  - Internal 1.2-V reference with buffer
  - External reference
- Interrupt generation
  - Finished scan conversion
  - Saturation detect and over-range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect
- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single-ended or differential, averaging
- Low-power modes
  - ADC core and reference voltage have dedicated low power modes

20.2 Block Diagram

Figure 20-1. Block Diagram



20.3 How it Works

This section includes the following contents:

- Introduction of each block: SAR ADC core, SARMUX, SARREF, and SARSEQ
- SAR ADC system resource: Interrupt, Low-Power mode, and SAR ADC status
  - System operation
- Configuration examples

20.3.1 SAR ADC Core

The PSoC 4 has two SAR ADC cores; each core is a 12-bit SAR ADC. The maximum sample rate for this ADC is 1 Msps. The SAR ADC core has the following features:

- Fully differential architecture; also supports single-ended mode
- 12-bit resolution and a selectable alternate resolution: either 8-bit or 10-bit
- Programmable acquisition time
- Programmable power mode (full, one-half, one-quarter)
- Supports single and continuous conversion mode

20.3.1.1 Single-ended and Differential Mode

The PSoC 4 SAR ADC can operate in single-ended and differential modes. It is designed in a fully differential architecture, optimized to provide 12-bit accuracy in the differential mode of operation. It gives full range output (0 to 4095) for differential inputs in the range of $-V_{REF}$ to $+V_{REF}$. SAR ADC can be configured in single-ended mode by fixing the negative input. Differential or single-ended mode can be configured by the channel configuration register, SARx_CHAN_CONFIGx.

The single-ended mode options of a negative input include: V_{SSA}, V_{REF}, or an external input from any of the eight pins with SARMUX connectivity. See the *PSoC 4500S datasheet* for the pin details. This mode is configured by the global configuration register SARx_CTRL. When Vminus is connected to these SARMUX pins, the single-ended mode is equivalent to differential mode. However, when the odd pin of each differential pair is connected to the common alternate ground, these conversions are 11-bit, because the measured signal value (SARMUX.vplus) cannot go below ground.

To get a single-ended conversion with 12 bits, it is necessary to connect V_{REF} to the negative input of the SAR ADC; then, the input range can be from 0 to $2 \times V_{REF}$.

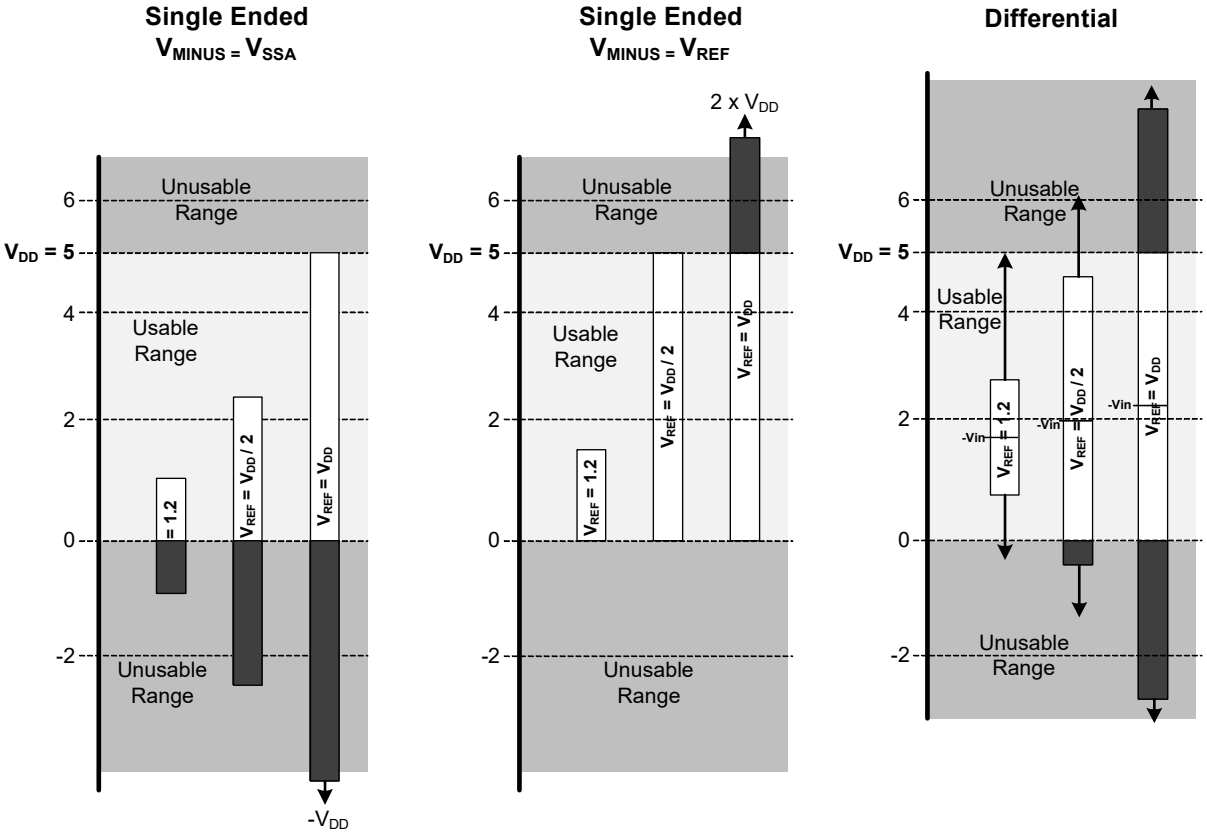
Note that temperature sensor can only be used in single-ended mode; it will override the SARx_CTRL [11:9] to '0'. The differential conversion is not available for temperature sensors; the result is undefined.

20.3.1.2 Input Range

All inputs should be in the range of V_{SSA} to V_{DDA}. Input voltage range is also limited by V_{REF}. If voltage on negative input is Vn and the ADC reference is V_{REF}, the range on the positive input is Vn ± V_{REF}. This criteria applies for both single-ended and differential modes. In single-ended mode, Vn is connected to V_{SSA}, V_{REF} or an external input.

Note that Vn ± V_{REF} should be in the range of V_{SSA} to V_{DDA}. For example, if a negative input is connected to V_{SSA}, the range on the positive input is '0' to V_{REF}, not $-V_{REF}$ to V_{REF}. This is because the signal cannot go below V_{SSA}. Only a half of the ADC range is usable because the positive input signal cannot swing below V_{SS}, which effectively only generates an 11-bit result.

Figure 20-2. SAR Input Range



20.3.1.3 Result Data Format

Result data format is configurable from two aspects:

- Signed/unsigned
- Left/right alignment

When the result is considered signed, the MSb of the conversion is used for sign extension to 16-bit with MSb. For an unsigned conversion, the result is zero extended to 16-bit. It can be configured by SARx_SAMPLE_CTRL [3:2] for differential and single-ended conversion, respectively.

The sample value can either be right-aligned or left-aligned within the 16 bit of the result register. By default, data is right-aligned in data[11:0], with sign extension to 16-bit, if required. A lower resolution combined with left-alignment will cause LSbs to be made zero.

Combined with signed and unsigned, and left and right alignment for 12-, 10-, and 8-bit conversion, the result data format can be as shown in Table 20-1.

Table 20-1. Result Data Format

Alignment	Signed/ Unsigned	Resolution	Result Register															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Right	Unsigned	12	–	–	–	–	11	10	9	8	7	6	5	4	3	2	1	0
		10	–	–	–	–	–	–	9	8	7	6	5	4	3	2	1	0
		8	–	–	–	–	–	–	–	–	7	6	5	4	3	2	1	0
Right	Signed	12	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0
		10	9	9	9	9	9	9	9	8	7	6	5	4	3	2	1	0
		8	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	0
Left	–	12	11	10	9	8	7	6	5	4	3	2	1	0	–	–	–	–
		10	9	8	7	6	5	4	3	2	1	0	–	–	–	–	–	–
		8	7	6	5	4	3	2	1	0	–	–	–	–	–	–	–	–

20.3.1.4 Negative Input Selection

The negative input connection choice affects the voltage range, signal-to-noise ratio (SNR), and effective resolution (see Table 20-2). In single-ended mode, the negative input of the SAR ADC can be connected to V_{SSA}, V_{REF}, or any of the eight pins with SARMUX connectivity.

Table 20-2. Negative Input Selection Comparison

Single-ended/ Differential	Signed/Unsigned	SARMUX Vminus	SARMUX Vplus Range	Result Register	Maximum SNR
Single-ended	N/A[a]	V_{SSA}	$+V_{REF}$ V_{SSA} = 0	0x7FF 0x000	Better
Single-ended	Unsigned	V_{REF}	$+2 \times V_{REF}$ V_{REF} V_{SSA} = 0	0xFFF 0x800 0	Good
Single-ended	Signed	V_{REF}	$+2 \times V_{REF}$ V_{REF} V_{SSA} = 0	0x7FF 0x000 0x800	Good
Single-ended	Unsigned	Vx	Vx + V_{REF} Vx Vx – V_{REF}	0xFFF 0x800 0	Best
Single-ended	Signed	Vx	Vx + V_{REF} Vx Vx – V_{REF}	0x7FF 0x000 0x800	Best
Differential	Unsigned	Vx	Vx + V_{REF} Vx Vx – V_{REF}	0xFFF 0x800 0	Best
Differential	Signed	Vx	Vx + V_{REF} Vx Vx – V_{REF}	0x7FF 0x000 0x800	Best

a. For single-ended mode with Vminus connected to V_{SSA}, conversions are effectively 11-bit because voltages cannot swing below V_{SSA} on any PSoC 4500S pin. Because of this, the global configuration bit SINGLE_ENDED_SIGNED (SARx_SAMPLE_CTRL[2]) will be ignored and the result is always (0x000-0x7FF).

To get a single-ended conversion with 12-bit, it is necessary to connect V_{REF} to the negative input of the SAR ADC; then, the input range can be from 0 to $2 \times V_{REF}$.

Note that single-ended conversions with Vminus connected to the pins with SARMUX connectivity are electrically equivalent to differential mode. However, when the odd pin of each differential pair is connected to the common alternate ground, these conversions are 11-bit, because the measured signal value (SARMUX.vplus) cannot go below ground.

20.3.1.5 Resolution

The PSoC 4 supports 12-bit resolution (default) and a selectable alternate resolution: either 8-bit or 10-bit for each channel. Resolution affects conversion time:

Conversion time (sar_clk) = resolution (bit) + 2

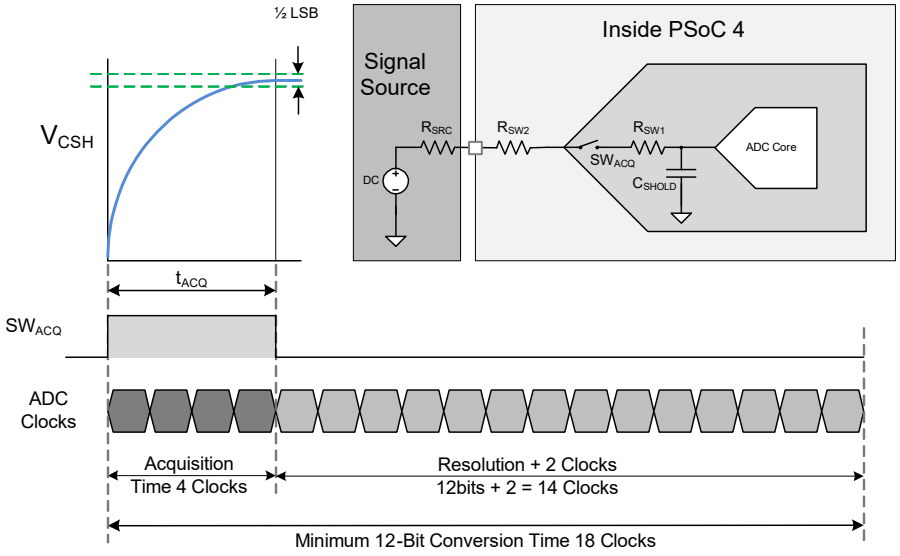
Total acquisition and conversion time (sar_clk) = acquisition time + resolution (bit) + 2

For 12-bit conversion and acquisition time = 4, 18 sar_clk is required. For example, if sar_clk is 18 MHz, 18 sar_clk is required for conversion and you will get 1 Msps conversion rate. A lower resolution results in a higher conversion rate.

20.3.1.6 Acquisition Time

Acquisition time is the time taken by sample and hold (S/H) circuit inside the SAR ADC to settle. After the acquisition time, the input signal source is disconnected from the SARADC core, and the output of the S/H circuit will be used for conversion. Each channel can select one from four acquisition time options, from 4 to 1023 SAR clock cycles defined in global configuration registers SARx_SAMPLE_TIME01 and SARx_SAMPLE_TIME23.

Figure 20-3. Acquisition Time



The acquisition time should be sufficient to charge the internal hold capacitor of the ADC through the resistance of the routing path, as shown in Figure 20-3. The recommended value of acquisition time is:

$t_{ACQ} \geq 9 \times (R_{SRC} + R_{SW2} + R_{SW1}) \times C_{SHOLD}$

Where:

C_{SHOLD} ~= 10 pF

$R_{SW2} + R_{SW1}$ = ~ 1100 to 1600 Ω, 1100 Ω for inputs from SARMUX pins and CTBm outputs, and ~1600Ω for inputs from pins connected via AMUXBUS slice near the SAR ADC. Routing signals to the SAR ADC from other AMUXBUS slices via AMUXBUS split switches is not recommended.

R_{SRC} = series resistance of the signal source

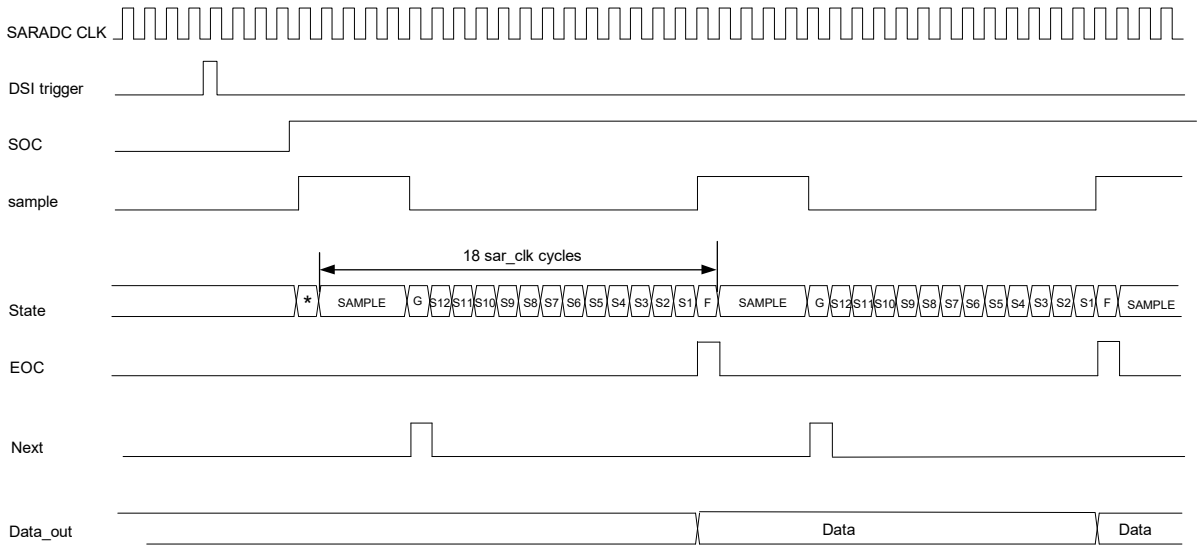
20.3.1.7 SAR ADC Clock

The SAR ADC clock frequency must be between 1 MHz and 18 MHz, which comes from the HFCLK via a clock divider. Note that a fractional divider is not supported for the SAR ADC. To get a 1-Msps sample rate, an 18-MHz SAR ADC clock is required. To achieve this, the system clock (HFCLK) must be set to 36 MHz rather than 48 MHz. A 12-bit ADC conversion with the minimum acquisition time of four clocks (at 18 MHz) requires 18 clocks in which to complete. A 10-bit and 8-bit conversion requires 16 and 14 clocks respectively. Note that the minimum acquisition time of four clock cycles at 18 MHz is based on the minimum acquisition time supported by the SAR block (R_{SW1} and C_{SHOLD} in Figure 20-3), which is 194 ns.

20.3.1.8 SAR ADC Timing

Figure 20-4 shows the SAR ADC timing diagram. A 12-bit resolution conversion needs 14 clocks (one bit needs one sar_clk, plus two excess sar_clk for G and F state). With acquisition time equal to four sar_clk cycles by default, 18 clock sar_clk cycles are required for total ADC acquisition and conversion. After sample (acquisition), it will output the next pulse. The SARMUX can route to another pin and signal; this will be done automatically with sequencer control (see SARSEQ on page 212 for details).

Figure 20-4. SAR ADC Timing

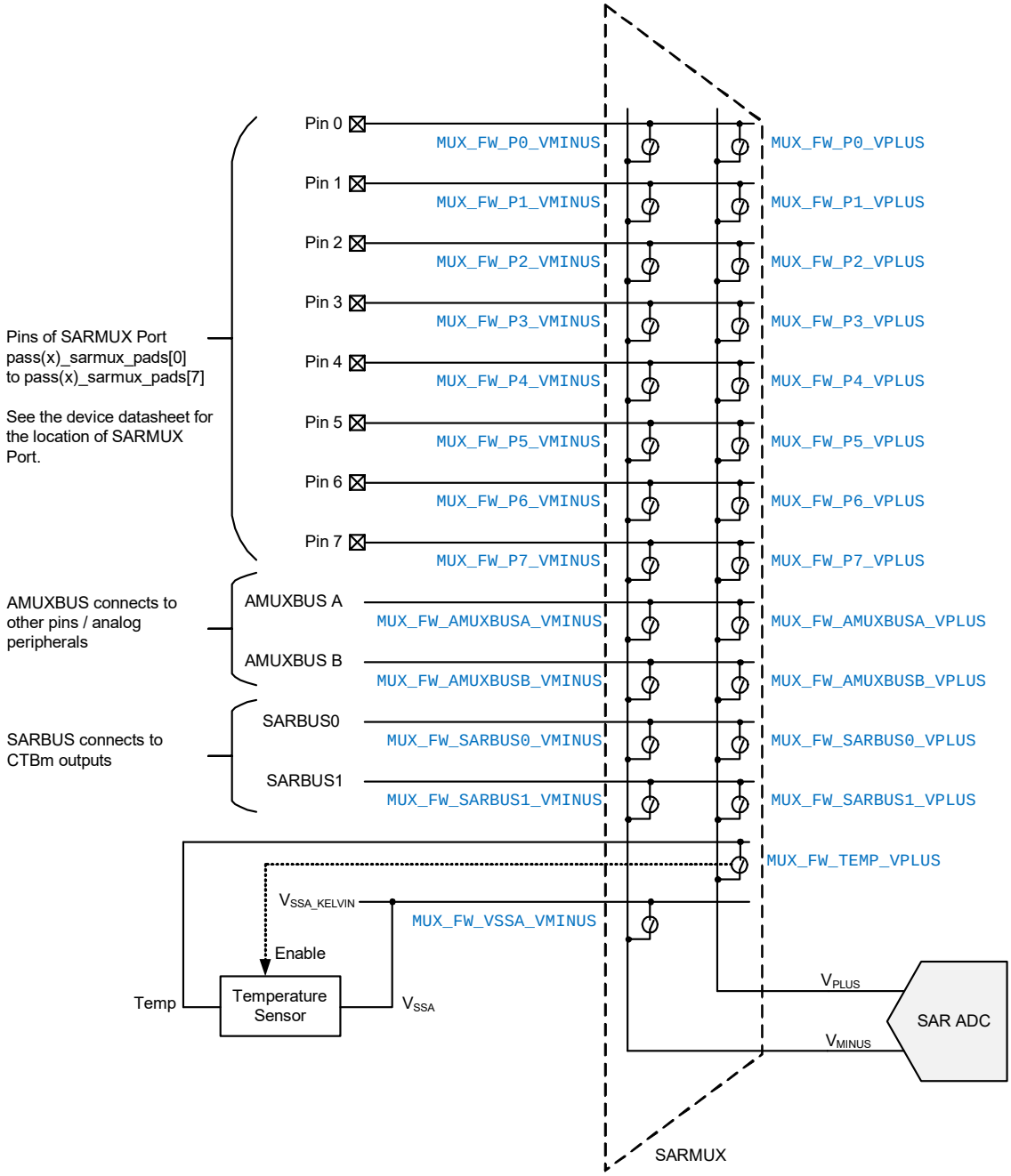


20.3.2 SARMUX

SARMUX is an analog dedicated programmable multiplexer. The main features of SARMUX are:

- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ) or firmware
- Charge pump inside:
  - If $V_{DDA} < 4.0$ V, charge pump should be turned ON to reduce switch resistance
  - If $V_{DDA} \geq 4.0$ V, charge pump is turned OFF and delivers V_{DDA} as its output
- Multiple inputs:
  - Analog signals from pins (SARMUX0's inputs come from port2; SARMUX1's input come from port3)
  - Temperature sensor output
  - CTBm output via sarbus0/1 (not fast enough to sample at 1 Msps)
  - AMUXBUS_A/B (not fast enough to sample at 1 Msps)

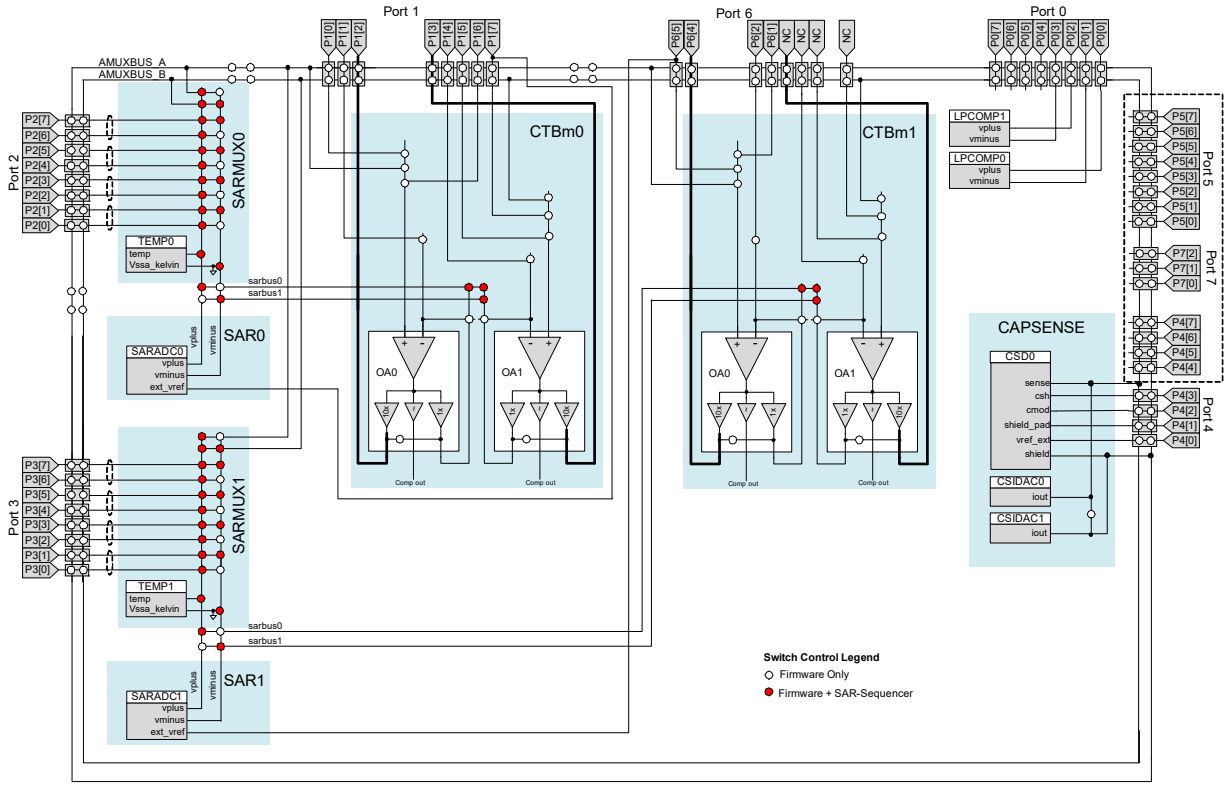
Figure 20-5. SARMUX



20.3.2.1 Analog Routing

SARMUX has many switches that may be controlled by the SARSEQ block (sequencer controller) or firmware. The sequencer is the hardware control method, which can be masked by the hardware control bit in the register, SARx_MUX_SWITCH_HW_CTRL. Different control methods have different control capability on the switches. See Figure 20-6.

Figure 20-6. SARMUX Switches and Control Capability



Sequencer control: The switches are controlled by the sequencer in the SARSEQ block. After configuring each channel's analog routing, it enables multi-channel automatic scan in a round-robin fashion, without CPU intervention. Not every switch can be controlled by the sequencer; see Figure 20-6. The corresponding registers are: SARx_CHAN_CONFIGx, SARx_MUX_SWITCH0, SARx_CTRL, and SARx_MUX_SWITCH_HW_CTRL.

Firmware control: Programmable registers directly define the VPLUS/VMINUS connection. It can control every switch in SARMUX; see Figure 20-6. For example, in firmware control, it is possible to do a differential measurement between any two pins or signals, not just two adjacent pins (as in sequencer control). However, it needs CPU intervention for multi-channel acquisition. The corresponding registers are: SARx_MUX_SWITCH0, SARx_MUX_SWITCH_HW_CTRL, and SARx_CTRL.

20.3.2.2 Analog Interconnection

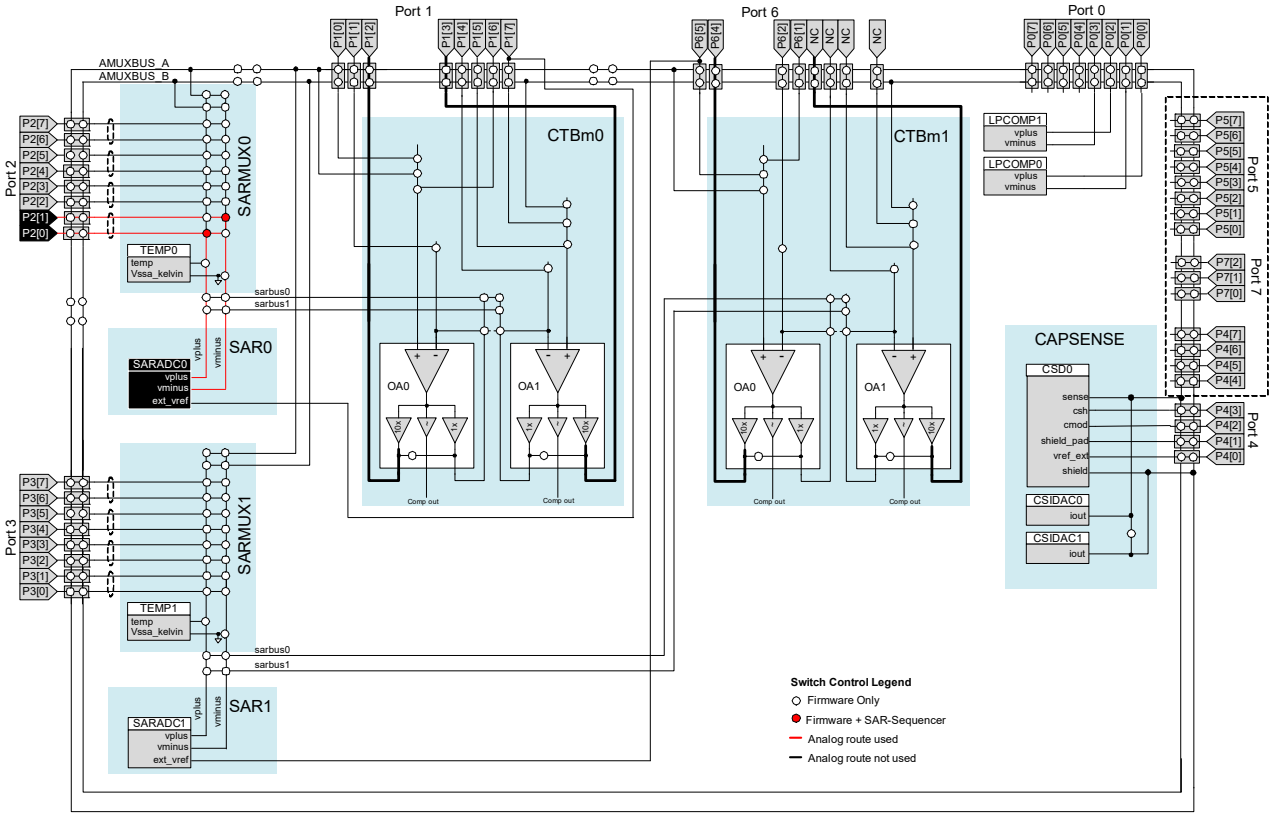
PSoC 4 analog interconnection is very flexible. The SAR ADC can be connected to multiple inputs via SARMUX, including both external pins and internal signals. For example, it can connect to a neighboring block such as CTBm. It can also connect to other pins except port 2 through AMUXBUS_A/B, at the expense of scanning performance (more parasitic coupling, longer RC time to settle).

Several cases are discussed here to provide a better understanding of the analog interconnection.

Input from External Pins

Figure 20-7 shows how two GPIOs that support the SARMUX are connected to the SAR ADC as a differential pair (Vplus/Vminus) via switches. These two switches can be controlled by the sequencer or firmware. The pins are arranged in adjacent pairs; for example, in SARMUX port P2[0] and P2[1], P2[2] and P2[3], and so on. If you need to use pins that are not paired as a differential pair, such as P2[1] and P2[2], the sequencer will not work; use firmware.

Figure 20-7. Input from External Pins

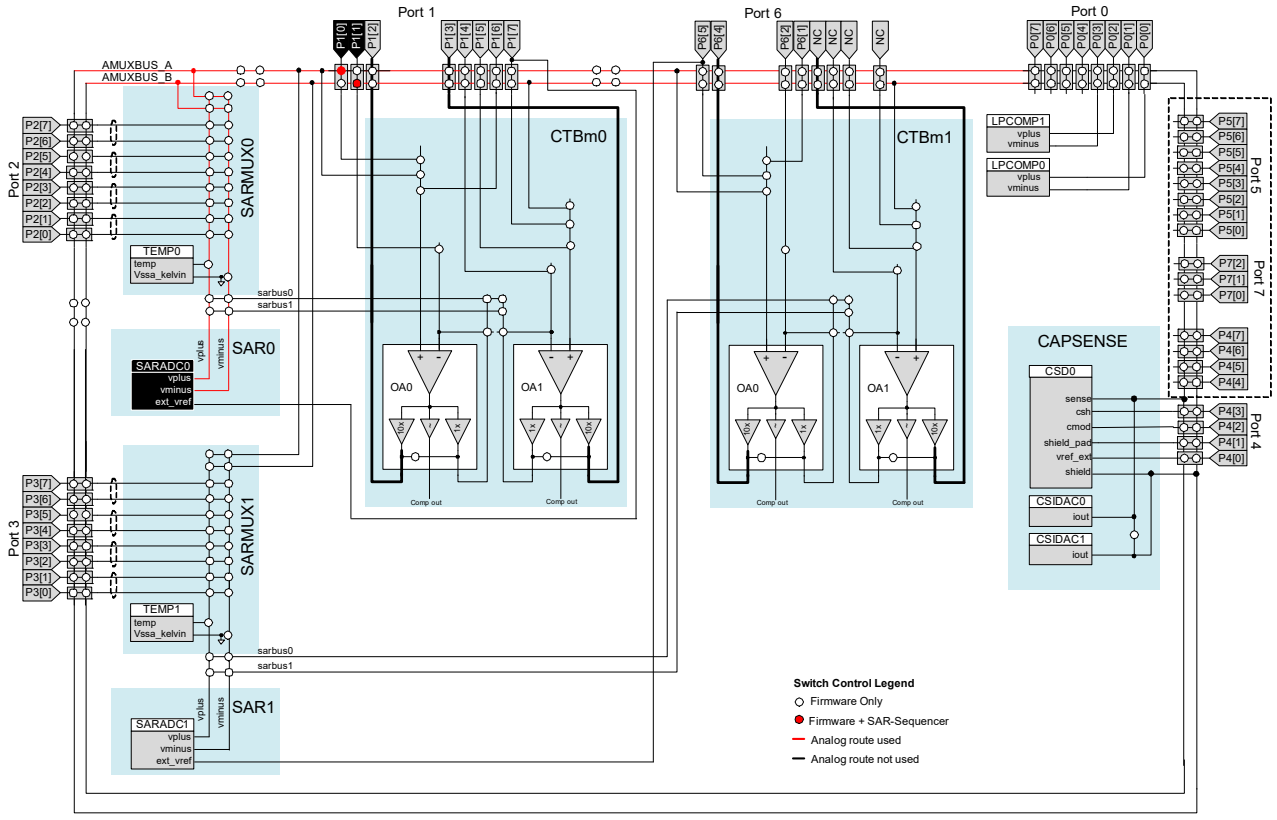


Input from Analog Bus (AMUXBUS_A/B)

Figure 20-8 shows how two pins that do not support SARMUX connectivity are connected to the ADC as a differential pair. Additional switches must connect these two pins to AMUXBUS_A and AMUX-BUS_B, and then connect AMUXBUS_A and AMUXBUS_B to ADC.

The additional switches reduce the scanning performance (more parasitic coupling, longer RC time to settle) – it is not fast enough to sample at 1 Msps. This is not recommended for external signals; the dedicated SARMUX port should be used, if possible.

Figure 20-8. Input from Analog Bus



Input from CTBm Output via sarbus

SAR ADC can be connected to the CTBm output via sarbus 0/1. Figure 20-9 shows how to connect an opamp (configured as a follower) output to a single-ended SAR ADC. The negative terminal is connected to V_{REF}. Figure 20-10 shows how to connect two opamp outputs to the SAR ADC as a differential pair. It must connect the opamp output to sarbus 0/1, then connect the SAR ADC input to sarbus 0/1. There are also additional switches, so it is not fast enough to sample at 1 Msps. However, the on-chip opamps add value for many applications.

Figure 20-9. Input from CTBm Output via sarbus

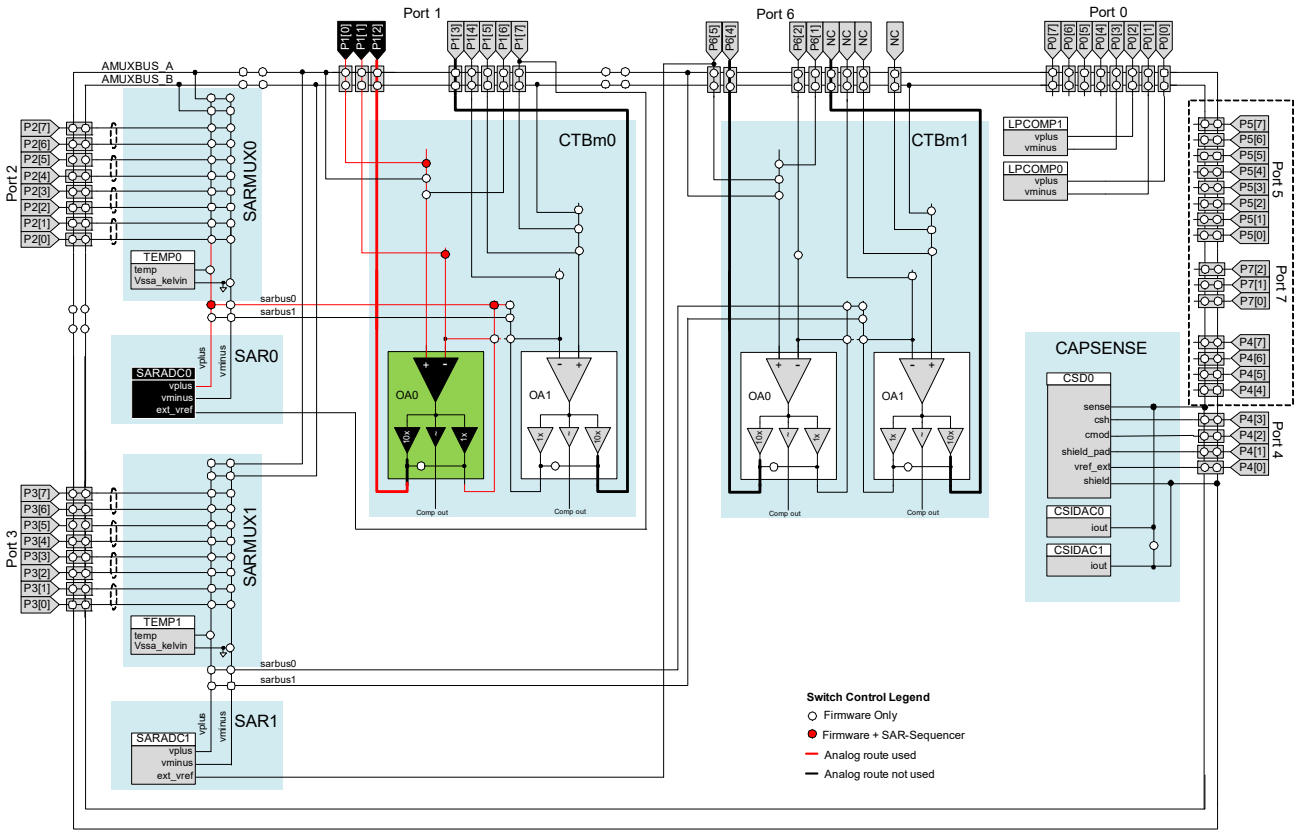
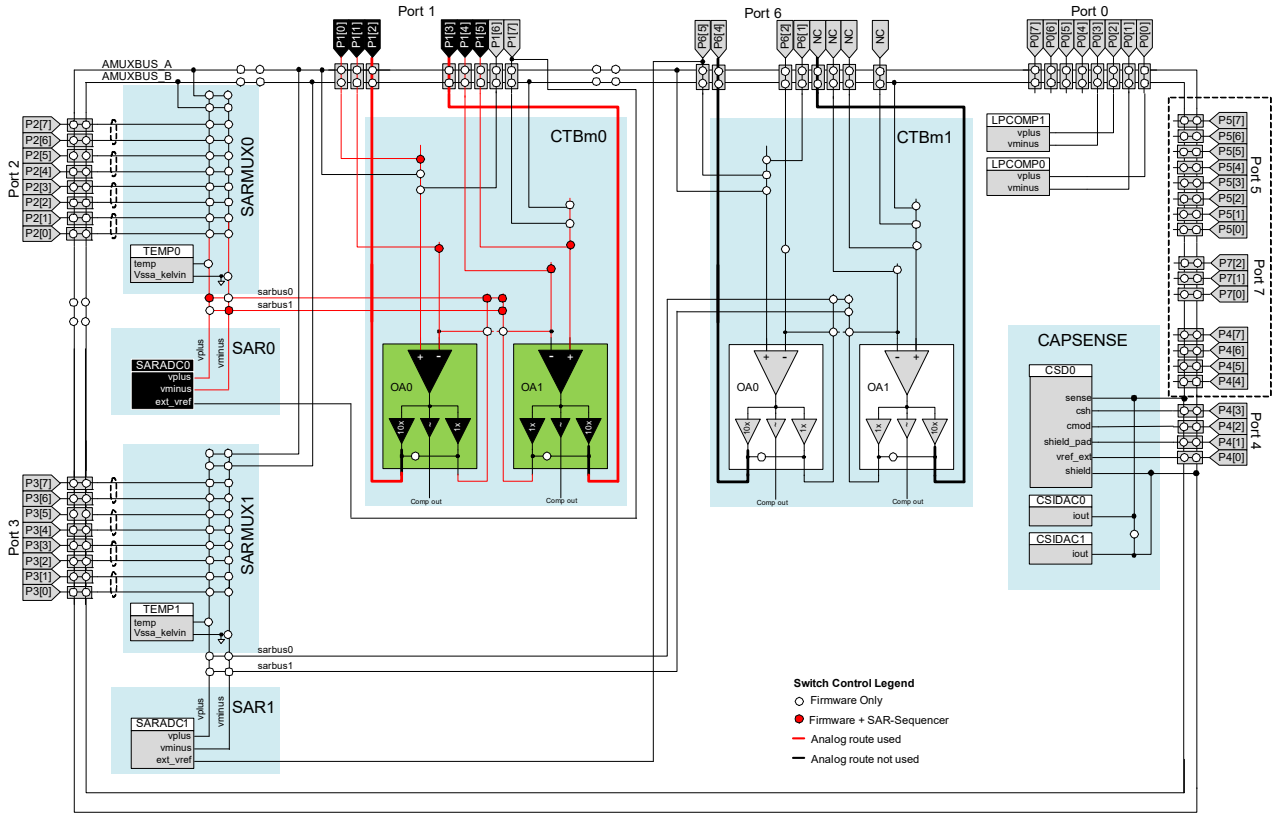


Figure 20-10. Inputs from CTBm Output via sarbus0 and sarbus1

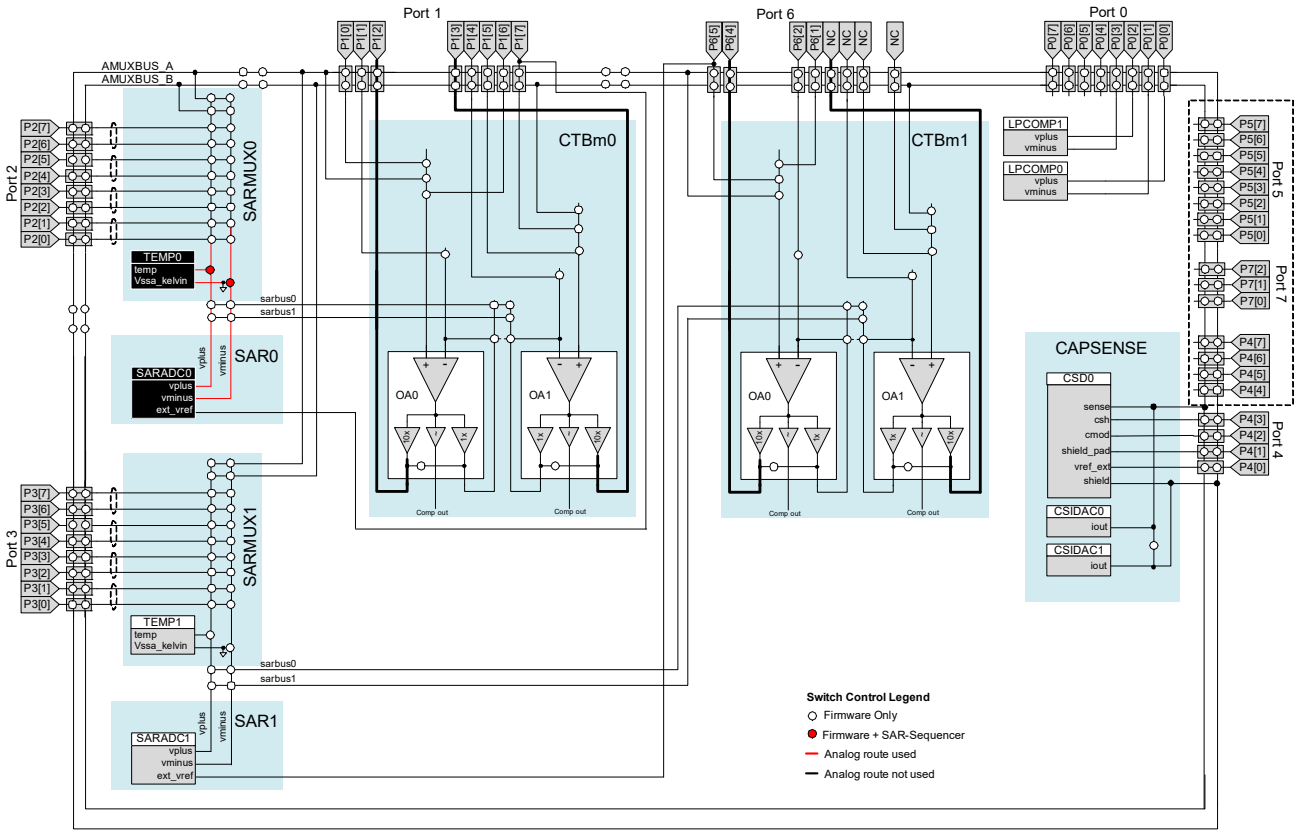


Input from Temperature Sensor

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Note that for temperature sensor, differential conversions are not available (conversion result is undefined); thus always use it in singled-ended mode.

As Figure 20-11 shows, the temperature sensor can be routed to positive input of the SAR ADC via switch, which can be controlled by the sequencer or firmware. Setting the MUX_FW_TEMP_VPLUS bit (SARx_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit will disable the temperature sensor by cutting its bias current.

Figure 20-11. Inputs from Temperature Sensor

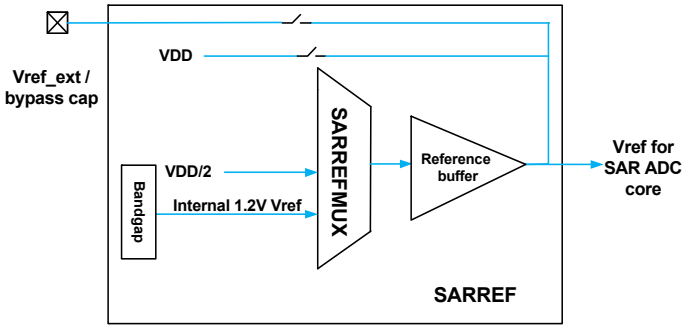


20.3.3 SARREF

The main features of the SAR reference block (SARREF) are:

- Reference options: V_{DDA}, $V_{DDA}/2$, 1.2-V bandgap (±1%), external reference
- Reference buffer + bypass cap to enhance internal reference drive capability

Figure 20-12. SARREF Block Diagram



20.3.3.1 Reference Options

Reference voltage selection for the SAR ADC consists of a reference MUX and switches inside the SARREF. The selection allows connecting V_{DDA}, $V_{DDA}/2$, and 1.2-V internal reference from a bandgap or an external V_{REF} connected to an Ext Vref/SAR bypass pin (see the *PSoC 4500S datasheet* for details). The control for the reference MUX in SARREF is in the global configuration register SAR_CTRL [6:4].

20.3.3.2 Bypass Capacitors

The internal references, 1.2 V from bandgap and $V_{DDA}/2$ are buffered with the reference buffer. This reference may be routed to the Ext Vref/SAR bypass pin, where an external capacitor can be used to filter internal noise that may exist on the reference signal. The SAR ADC sample rate is limited to 100 ksps (at 12-bit) without an external reference bypass capacitor. For example, without a bypass capacitor and with 1.2-V internal V_{REF}, the maximum SAR ADC clock frequency is 1.6 MHz. When using an external reference, it is recommended that an external capacitor is used. Bypass capacitors can be enabled by setting SARx_CTRL [7]. Table 20-3 lists different reference modes and their maximum frequency/sample rates for 12-bit continuous mode operation.

Table 20-3. Reference Modes

Reference Mode	Reference SARx_CTRL [6:4]	Bypass Cap SARx_CTRL[7]	Buffer	Max Frequency	Max Sample Rate (12-bit)
1.2 V internal V_{REF} without bypass cap	4	0	Yes	1.6 MHz	100 ksps
1.2 V internal V_{REF} with bypass cap	4	1	Yes	18 MHz	1 Msps
External V_{REF} (low-impedance path)	5	X	No	18 MHz	1 Msps
$V_{DDA}/2$ without bypass cap	6	0	Yes	1.6 MHz	100 ksps
$V_{DDA}/2$ with bypass cap	6	1	Yes	18 MHz	1 Msps
V_{DDA}	7	X	No	9 MHz	500 ksps

1.2-V internal V_{REF} startup time varies with different bypass capacitor size. Table 20-4 lists two common values for the bypass capacitor and its startup time specification. If reference selection is changed between scans or when scanning after Sleep/Deep Sleep, make sure that the 1.2-V internal V_{REF} is settled when the SAR ADC starts sampling. The worst-case settling time (when V_{REF} is completely discharged) is the same as the startup time.

Table 20-4. Bypass Capacitor Values

Internal V_{REF} Startup Time	Maximum Specification
Startup time for reference with external capacitor (1 µF)	2 ms
Startup time for reference with external capacitor (100 nF)	200 µs

20.3.3.3 Input Range versus Reference

All inputs should be between V_{SSA} and V_{DDA}. The ADC input range is limited by the V_{REF} selection. If the negative input is Vn and the ADC reference is V_{REF}, the range on the positive input is Vn ± V_{REF}. This criteria applies for both single-ended and differential modes as long as both negative and positive inputs stay within V_{SSA} to V_{DDA}.

20.3.4 SARSEQ

SARSEQ is a dedicated sequencer controller that automatically sequences the input MUX from one channel to the next, while placing the result in an array of registers, one per channel.

SARSEQ has the following functions:

- Controls SARMUX analog routing automatically without CPU intervention
- Controls the SAR ADC core (such as resolution, acquisition time, and reference)
- Receives data from the SAR ADC and does pre-processing (average, range detect)
- Uses double buffer to store the result so that the CPU can safely read the results of the last scan while the next scan is in progress.

The features of SARSEQ are:

- Sixteen channels can be individually enabled as an automatic scan without CPU intervention.
- An additional channel (injection channel) for inserting an infrequent signal in an automatic scan.
- Each channel has the following features:
  - Single-ended or differential mode
  - Input from an external pin (only for eight channels in single-ended mode and four channels in differential mode) or an internal signal (AMUXBUS/CTBm/temperature sensor)
  - Up to four programmable acquisition times
  - Default 12-bit resolution, selectable alternate resolution: either 8-bit or 10-bit
  - Result averaging
- Scan triggering
  - One shot, periodic, or continuous mode
  - Triggered by the TCPWM block, CTBm comparator, LPCOMP, SAR ADC EOC signal, and by firmware
- Hardware averaging support
  - First-order accumulate
  - From 2 to 256 samples averaging (powers of 2)
  - Results in 16-bit representation
- Double buffering of output data
  - Left- or right-adjusted results
  - Results in working register and result register
- Interrupt generation
  - Finished scan conversion
  - Channel saturation detect in all control modes
  - Over range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect
- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single-ended/differential, averaging

20.3.4.1 Averaging

The SARSEQ block has a 20-bit accumulator and shift register to implement averaging. Averaging is after the signed extension. The global configuration SARx_SAMPLE_CTRL register specifies the details of averaging.

Channel configuration SARx_CHAN_CONFIG register has an enable bit (AVG_EN) to enable averaging.

In global configuration, AVG_CNT (SARx_SMAPLE_CTRL [6:4]) specifies the number of samples (N) according to this formula:

N=2^(AVG_CNT+1) N range = [2..256]

For example, if AVG_CNT (SARx_SMAPLE_CTRL [6:4]) = 3, then N = 16.

AVG_SHIFT bit (SARx_SAMPLE_CTRL[7]) is used to shift the result to get averaged; it should be set if averaging is enabled.

If a channel is configured for averaging, the SARSEQ will take 'N' consecutive samples of the specified channel in every scan. The conversion result is 12-bit and the maximum value of 'N' is 256 (left shift 8 bits), so the 20-bit accumulator will never overflow.

If AVG_SHIFT in the SARx_SAMPLE_CTRL register is set, the SAR sequencer performs sign extension and then accumulation. The accumulated result is shifted right AVG_CNT + 1 bits to get averaged. If not, the result is forced to shift right to ensure that it fits in 16 bits. Right-shift is done by maximum (0, AVG_CNT-3) – if the number of samples is more than 16 (AVG_CNT >3), then the accumulation result is shifted right AVG_CNT 3bits; it AVG_CNT < 3, the result is not shifted. Note that in this case, the average result is bigger than expected; it is recommended to set AVG_SHIFT. This mode always uses the selected resolution of ADC (12, 10, or 8 bits).

20.3.4.2 Range Detection

The SARSEQ supports range detection to allow automatic detection of result values compared to two programmable thresholds without CPU involvement. Range detection is defined by the SARx_RANGE_THRES register. The RANGE_LOW field (SARx_RANGE_THRES [15:0]) value defines the lower threshold and RANGE_HIGH field (SARx_RANGE_THRES [31:16]) defines the upper threshold of the range.

The SARx_RANGE_COND bits define the condition that triggers a channel maskable range detect interrupt (RANGE_INTR). The following conditions can be selected:

0: result < RANGE_LOW (below range)

1: RANGE_LOW ≤ result < RANGE_HIGH (inside range)

2: RANGE_HIGH ≤ result (above range)

3: result <RANGE_LOW || RANGE_HIGH <= result (outside range)

See Range Detection Interrupts on page 217 for details.

20.3.4.3 Double Buffer

Double buffering is used so that firmware can read the results of a complete scan while the next scan is in progress. The SAR ADC results are written to a set of working registers until the scan is complete, at which time the data is copied to a second set of registers where the data can be read by the user application. This action allows sufficient time for the firmware to read the previous scan before the present scan is completed. All input channels are double-buffered with 16 registers, except the injection channel. The injection channel is not required to be double-buffered because it is not normally part of a normal channel scan.

20.3.4.4 Injection Channel

The injection channel is similar to the other channels, with the exception that it is not part of a regular scan. The injection channel is used for incidental or rare conversions; for example, sampling the temperature sensor every two seconds. Note that if the SAR ADC is operating in continuous mode, enabling the injection channel will change the sample rate.

The injection channel can only be controlled by firmware with a firmware trigger (one-shot). This means the injection channel does not support continuous trigger. As the only trigger is one-shot, there is no need for double-buffering or an overflow interrupt.

The conversions for the injection channel can be configured in the same way as the regular channels by setting the SARx_INJ_CHAN_CONFIG register; it supports:

- Pin or signal selection
- Single-ended or differential selection
- Choice of resolution between 12-bit or the globally specified SUB_RESOLUTION
- Sample time select from one of the four globally specified sample times
- Averaging select

It supports the same interrupts as the regular channel except the overflow interrupt:

- Maskable EOC interrupt INJ_EOC_INTR
- Maskable range detect interrupt INJ_RANGE_INTR
- Maskable saturation detect interrupt INJ_SATURATE_INTR
- Maskable collision interrupt INJ_COLLISION_INTR

SARx_INTR, SARx_INTR_MASK, SARx_INTR_MASKED, and SARx_INTR_SET are the corresponding registers.

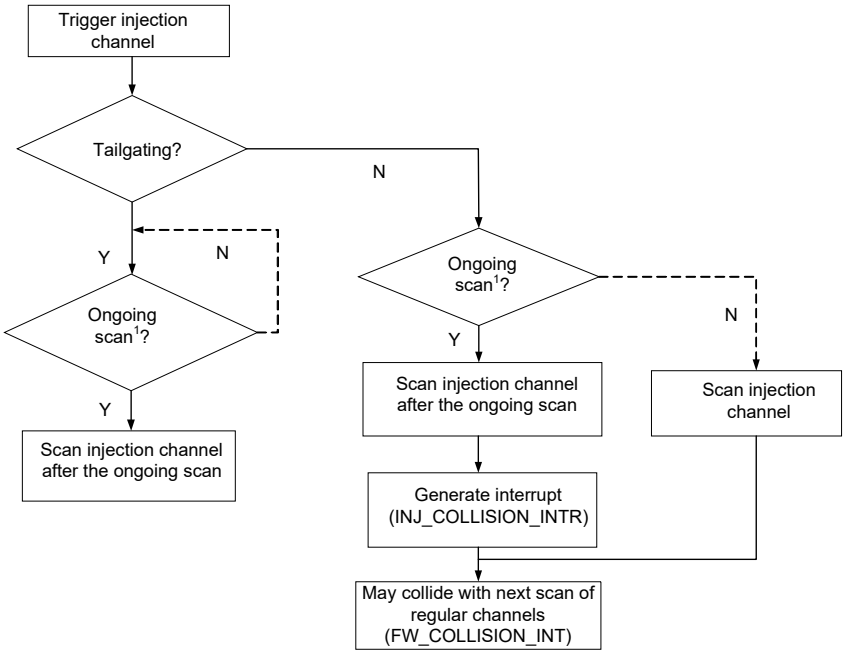
These features are described in detail in Global SARSEQ Configuration on page 222, Channel Configurations on page 223, and SAR Interrupts on page 216.

Tailgating

The injection channel conversion can be triggered by setting the start or enable bit INJ_START_EN (SARx_INJ_CHAN_CONFIG [31]). It is recommended to select tailgating by setting INJ_TAILGATING=1 (SARx_INJ_CHAN_CONFIG [30]). The injection channel will be scanned at the end of the ongoing scan of regular channels without any collision. However, if there is no ongoing scan or the SAR ADC is idle, and tailgating is selected, INJ_START_EN will enable the injection channel to be scanned at the end of the next scan of regular channels.

If tailgating is not selected, setting the INJ_START_EN bit immediately starts the conversion of the injection channel provided there is no ongoing scan or SAR ADC is idle. If a scan of the regular channels is ongoing, the injection channel will be scanned at the end of the ongoing scan, but it will cause a collision and generate a collision interrupt (INJ_COLLISION_INTR). Another potential problem without tailgating is that it can cause the next scan of the regular channels to collide with the injection channel conversion (FW_COLLISION_INTR is raised). As a result, the next scan of regular channels is postponed until the injection scan is finished, thus causing jitter on a regular scan.

Figure 20-13. Injection Channel Flow Chart



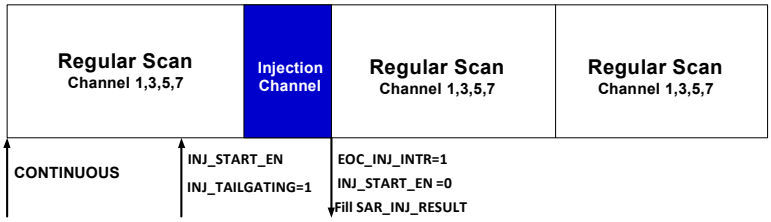
[1] *scan here means scan of ALL the regular channels*

The disadvantage of tailgating is that it may be a long time before the next trigger occurs. If there is no risk of colliding or causing jitter on the regular channels, the injection channel can be used safely without tailgating.

After completing the conversion for the injection channel, the EOC interrupt (INJ_EOC_INTR) is set and the INJ_START_EN bit is cleared. The conversion data of the injection is put in the SARx_INJ_RESULT register. Similar to SARx_CHAN_RESULT, the registers contain mirror bits for "valid" (=INJ_EOC_INTR), range detect, saturation detect interrupt, and a mirror bit of the collision interrupt (INJ_COLLISSION_INTR).

Figure 20-14 is an example when injection channel is enabled during a continuous scan (channel 1, 3, 5, and 7 are enabled), and tailgating is enabled. Note that the INJ_START_EN bit is immediately cleared when the SAR is disabled (but only if it was enabled before).

Figure 20-14. Injection Channel Enabled with Tailgating



20.3.5 SAR Interrupts

An interrupt can be generated on different events:

- End of Scan – When scanning is complete for all enabled channels.
- Overflow – When the result register is updated before the previous result is read.
- Collision – When a new trigger is received while the SAR ADC is still processing the previous trigger.
- Injection EOC – When the injection channel is converted.
- Range Detection – When the channel result meets the threshold value.
- Saturation Detection – When the channel result is equal to the minimum or maximum value of the set resolution.

This section describes each interrupt in detail. These interrupts have an interrupt mask in the SARx_INTR_MASK register. By making the interrupt mask LOW, the corresponding interrupt source is ignored. The SAR interrupt is generated if the interrupt mask bit is HIGH and the corresponding interrupt source is pending.

When servicing an interrupt, the ISR clears the interrupt source by writing a ‘1’ to the interrupt bit after reading the data.

The SARx_INTR_MASKED register is the logical AND between the interrupt sources and the interrupt mask. This register provides a convenient way for the firmware to determine the source of the interrupt.

For verification and debug purposes, a set bit (such as EOS_SET in the SARx_INTR_SET register) is used to trigger each interrupt. This action allows the firmware to generate an interrupt without the actual event occurring.

20.3.5.1 End-of-Scan Interrupt (EOS_INTR)

After completing a scan, the end-of-scan interrupt (EOS_INTR) is raised. Firmware should clear this interrupt after picking up the data from the RESULT registers.

Optionally, EOS_INTR can also be sent out on the GPIO by setting the EOS_DSI_OUT_EN bit in SARx_SAMPLE_CTRL [31]. The EOS_INTR signal is maintained for two system clock cycles. These cycles coincide with the data_valid signal for the last channel of the scan (if selected).

EOS_INTR can be masked by making the EOS_MASK bit 0 in the SARx_INTR_MASK register. EOS_MASKED bit of the SARx_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a ‘1’ to EOS_SET bit in the SARx_INTR_SET register can set EOS_INTR, which is intended for debug and verification.

20.3.5.2 Overflow Interrupt

If a new scan completes and the hardware tries to set EOS_INTR and EOS_INTR is still HIGH (firmware does not clear it fast enough), then an overflow interrupt (OVERFLOW_INTR) is generated by the hardware. This usually means that the firmware is unable to read the previous results before the current scan completes. In this case, the old data is overwritten.

OVERFLOW_INTR can be masked by making the OVERFLOW_MASK bit ‘0’ in the SARx_INTR_MASK register. The OVERFLOW_MASKED bit of the SARx_INTR_MASKED register is the logic AND of interrupt flags and interrupt masks, which is for firmware convenience. Writing a ‘1’ to the OVERFLOW_SET bit in the SARx_INTR_SET register can set OVERFLOW_INTR, which is intended for debug and verification.

20.3.5.3 Collision Interrupt

It is possible that a new trigger is generated while the SARSEQ is still busy with the scan started by the previous trigger. Therefore, the scan for the new trigger is delayed until after the ongoing scan is completed. It is important to notify the firmware that the new sample is invalid. This is done through the collision interrupt, which is raised any time a new trigger, other than the continuous trigger, is received.

There are three collision interrupts: for the firmware trigger (FW_COLLISION_INTR), for the external trigger (DSI_COLLISION_INTR), and for the injection channel (INJ_COLLISION_INTR). These interrupts allow the firmware to identify which trigger collided with an ongoing scan.

When the external trigger is used in level mode, DSI_COLLISION_INTR will never be set.

The three collision interrupts can be masked by making the corresponding bit ‘0’ in the SARx_INTR_MASK register. The corresponding bit in the SARx_INTR_MASKED register is the logic AND of interrupt flags and interrupt masks. Writing a ‘1’ to the corresponding bit in the SARx_INTR_SET register can set the collision interrupt, which is intended for debug and verification.

20.3.5.4 Injection EOC Interrupt (INJ_EOC_INTR)

After completing a conversion for the injection channel, the injection EOC interrupt is raised (INJ_EOC_INTR). The firmware clears this interrupt after picking up the data from the INJ_RESULT register.

Note that if the injection channel is tailgating a scan, EOS_INTR is raised in parallel to starting the injection channel conversion. The injection channel is not considered part of the scan.

INJ_EOC_INTR can be masked by making the INJ_EOC_MASK bit ‘0’ in the SARx_INTR_MASK register. The INJ_EOC_MASKED bit of the SARx_INTR_MASKED register is the logic AND of interrupt flags and interrupt masks. Writing a ‘1’ to the INJ_EOC_SET bit in the SARx_INTR_SET register can set INJ_EOC_INTR, which is intended for debug and verification.

20.3.5.5 Range Detection Interrupts

The range detection interrupt flag can be set after averaging, alignment, and sign extension (if applicable). This means it is not required to wait for the entire scan to complete to determine whether a channel conversion is over-range. The threshold values need to have the same data format as the result data.

The range detection interrupt for a specified channel can be masked by setting the SARx_RANGE_INTR_MASK register specified bit to ‘0’. The SARx_RANGE_INTR_MASKED register reflects a bitwise AND between the interrupt request and mask registers. If the value is not zero, the SAR interrupt signal to the NVIC is HIGH.

SARx_RANGE_INTR_SET can be used for debug/verification. Write a ‘1’ to set the corresponding bit in the interrupt request register; when read, this register reflects the interrupt request register.

There is a range detect interrupt for each channel (RANGE_INTR and INJ_RANGE_INTR).

20.3.5.6 Saturate Detection Interrupts

Saturation detection is always applied to every conversion. This feature detects whether a sample value is equal to the minimum or maximum value for the specific resolution and sets a maskable interrupt flag for the corresponding channel. This action allows the firmware to take action, such as discarding the result, when the SAR ADC saturates. The sample value is tested immediately after conversion, before averaging. This means that the interrupt is set while the averaged result in the data register is not equal to the minimum or maximum.

When a 10-bit or 8-bit resolution is selected for the channel, saturate detection is done on 10-bit or 8-bit data.

The saturation interrupt flag is set immediately to enable a fast response to saturation, before the full scan and averaging. The saturation detection interrupt for a specified channel can be masked by setting the SARx_SATURATE_INTR_MASK register specified bit to ‘0’. The SARx_SATURATE_INTR_MASKED register reflects a bit-wise AND between the interrupt request and mask registers. If the value is not zero, then the SAR interrupt signal to the NVIC is HIGH.

SARx_SARTURATE_INTR_SET can be used for debug/verification. Write a ‘1’ to set the corresponding bit in the interrupt request register; when read, this register reflects the interrupt request register.

20.3.5.7 Interrupt Cause Overview

The INTR_CAUSE register contains an overview of all pending SAR interrupts. It allows the ISR to determine the cause of the interrupt. The register consists of a mirror copy of SARx_INTR_MASKED. In addition, it has two bits that aggregate the range and saturate detection interrupts of all channels. It includes a logical OR of all the bits in RANGE_INTR_MASKED and SATURATE_INTR_MASKED registers (does not include INJ_RANGE_INTR and INJ_SATURATE_INTR).

20.3.6 Trigger

A scan can be triggered in the following ways:

- A firmware or one-shot trigger is generated when the firmware writes to the FW_TRIGGER bit of the SARx_START_CTRL register. After the scan is completed, the SARSEQ clears the FW_TRIGGER bit and goes back to idle mode waiting for the next trigger. The FW_TRIGGER bit is cleared immediately after the SAR is disabled.
- An external trigger can be TCPWM outputs, CTBm comparator outputs, LPCOMP outputs, and SAR ADC's end-of-sampling and EOC signals. Hardware trigger is enabled by writing '1' to the DSI_TRIGGER_EN bit of the SARx_SAMPLE_CTRL register. The signal for the trigger is selected using the PERI_TR_GROUP2_TR_OUT_CTL0 register in PSoC 4.

Table 20-5. Hardware Trigger Source Selection in PSoC 4

PERI_TR_GROUP2_TR_OUT_CTL0[6:0]	Trigger Source
0	Hardwired to 0 (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
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	TCPWM 0 Line Output
26	TCPWM 1 Line Output
27	TCPWM 2 Line Output
28	TCPWM 3 Line Output
29	TCPWM 4 Line Output
30	TCPWM 5 Line Output
31	TCPWM 6 Line Output
32	TCPWM 7 Line Output
33	PASS[x].SAR ADC Sample Done(sdone) Signal

Table 20-5. Hardware Trigger Source Selection in PSoC 4<*Italic*> *(continued)*

PERI_TR_GROUP2_TR_OUT_CTL0[6:0]	Trigger Source
34	PASS[x].SAR ADC EOC Signal
35	PASS[x].CTBm Comparator 0 Output
36	PASS[x].CTBm Comparator 1 Output
37	LPCOMP 0 Output
38	LPCOMP 1 Output

- A continuous trigger is activated by setting the CONTINUOUS bit in the SARx_SAMPLE_CTRL register. In this mode, after completing a scan, the SARSEQ starts the next scan immediately; therefore, the SARSEQ is always BUSY. As a result, all other triggers are essentially ignored. Note that FW_TRIGGER will still get cleared by hardware- on the next completion.

The three triggers are mutually exclusive, although there is no hardware requirement. If an external trigger coincides with a firmware trigger, the external trigger is handled first and a separate scan is done for the firmware trigger (and a collision interrupt is set). When an external trigger coincides with a continuous trigger, both triggers are effectively handled at the same time (a collision interrupt may be set for the external trigger).

For firmware continuous trigger, it takes only one SAR ADC clock cycle before the sequencer informs the SAR ADC to start sampling (provided the sequencer is idle). For the external trigger, it depends on the trigger configuration setting.

20.3.6.1 External Trigger Configuration

- Synchronization

If the incoming external trigger signal is not synchronous to the AHB clock, the signal needs to be synchronized by double flopping it (default). However, if the trigger signal is already synchronized with the AHB clock, then these two flops can be bypassed. The configuration bit, DSI_SYNC_TRIGGER in the SARx_SAMPLE_CTRL register, controls the double flop bypass. DSI_SYNC_TRIGGER affects the trigger width (TW) and trigger interval (TI) requirement of the pulse trigger signal.

- Trigger Level

The trigger can either be a pulse or a level; this is indicated by the configuration bit, DSI_TRIGGER_LEVEL in the SARx_SAMPLE_CTRL register. If it is a level, the SAR starts new scans for as long as the trigger signal remains HIGH. When the trigger signal is a pulse input, a positive edge detected on the trigger signal triggers a new scan.

- Transmission Time

After the ‘dsi_trigger’ is raised, it takes some transmission time before the SAR ADC is told to start sampling. With different DSI_SYNC_TRIGGER and DSI_TRIGGER_LEVEL configurations, the transmission time is different; Table 20-6 shows the maximum time. Two trigger pulse intervals should be longer than the transmission time; otherwise, the second trigger is ignored.

When the SAR is disabled (ENABLED=0), the trigger is ignored.

Table 20-6. External Trigger Maximum Time

Maximum External_TRIGGER Transmission Time	Bypass Sync DSI_SYNC_TRIGGER=0	Enable Sync DSI_SYNC_TRIGGER=1 (by default)
Pulse trigger: DSI_TRIGGER_LEVEL=0 (by default)	1 clk_sys+2 clk_sar	3 clk_sys+2 clk_sar
Level Trigger: DSI_TRIGGER_LEVEL=1	2 clk_sar	2 clk_sys+2 clk_sar

Table 20-7. Trigger Signal Requirement

Trigger Specification	Requirement
Trigger Width (TW)	TW should be large enough so that a trigger can be locked. If DSI_SYNC_TRIGGER=1, TW ≥ 2 clk_sys cycle. If DSI_SYNC_TRIGGER=0, TW ≥ 1 SAR clock cycle.
Trigger interval (TI)	Trigger interval should be longer than the transmission time (as specified in Table 20-6); otherwise, the second trigger pulse will be ignored.

20.3.7 SAR ADC Status

The current SAR status can be observed from the BUSY and CUR_CHAN fields in the SARx_STATUS register. The BUSY bit is HIGH whenever the SAR is busy sampling or converting a channel; the CUR_CHAN [4:0] bits indicate the number of the current channel being sampled (channel 16 indicates the injection channel). The SW_VREF_NEG bit indicates the current switch status that shorts NEG with V_{REF} input.

CHAN_WORK_VALID in the CHAN_WORK_VALID register will be set if the WORK data that was sampled during the last scan is valid. When CHAN_RESULT_VALID is set in the CHAN_RESULT_VALID register, indicating that the RESULT data is valid, the corresponding CHAN_WORK_VALID bit is cleared. The CUR_AVG_ACCU and CUR_AVG_CNT fields in the SARx_AVG_STAT register indicate the current averaging accumulator contents and the current sample counter value for averaging (counts down).

The SARx_MUX_SWITCH_STATUS register gives the current switch status of the MUX_SWITCH0 register. These status registers help to debug SAR behavior.

20.3.8 Low-Power Mode

The current consumption of the SAR ADC can be divided into two parts: SAR ADC core and SARREF. There are several methods to reduce the power consumption of the SAR ADC core. The easiest way is to reduce the trigger frequency; that is, reduce the number of conversions per second. Another option is to use lower resolution for channels that do not need high accuracy. This action shortens the conversion by up to four out of 18 cycles (for 8-bit resolution and minimum sample time). In addition, the SAR ADC offers the ICONT_LV[1:0] configuration bits, which control overall power of the SAR ADC. Maximum clock rates for each power setting should be observed.

Table 20-8. ICONT_LV for Low Power Consumption

ICONT_LV[1:0]	Relative Power of SAR ADC Core [%]	Maximum Frequency [MHz]	Minimum Sample Time [cycles]	Maximum Sample Speed (at 12-bit) [ksps]
0	100	18	4	1000
1	50	9	3	529
2	133	18	4	1000
3	25	4.5	2	281

In addition to controlling the power of the SAR ADC core, the power consumed by the V_{REF} buffer (if used) can also be configured. Note that for a full V_{DDA} range (1.7 V to 5.5 V) operation without external bypass capacitor, the VREF buffer should be operated in 2x power mode. However, the maximum sample rate supported without an external bypass capacitor remains at 100 ksps. For a 1-Msps sample rate, an external bypass capacitor and an 18-MHz clock are required. See Table 20-9 for details.

Table 20-9. SAR VREF Power Options

PWR_CTRL_VREF [1:0]	External Bypass Capacitor Required	Relative VREF Power [%]	Maximum Frequency [MHz]	Minimum Sample Time [cycles]	Maximum Sample Speed (at 12-bit) [ksps]	VDDA Range
0	Yes	100	18	4	1000	1.7 V - 5.5 V
0	No	100	1.6	2	100	2.7 V - 5.5 V
2	No	200	1.6	2	100	1.7 V - 5.5 V
1 or 3	Invalid setting - Should not be used					

Using an external VREF eliminates the need for the VREF buffer and bypass capacitor, which in turn reduces overall power consumption of the SAR ADC block.

20.3.9 System Operation

After the SAR analog is enabled by setting the ENABLED bit (SARx_CTRL [31]), follow these steps to start ADC conversions with the SARSEQ:

1. Set SARMUX analog routing (pin/signal selection) via sequencer/firmware.
2. Set the global SARSEQ conversion configurations.
3. Configure each channel source (such as pin address).
4. Enable channels.
5. Set the trigger type.
6. Set interrupt masks.
7. Start the trigger source.
8. Retrieve data after each EOC interrupt.
9. Perform injection conversions if needed.

Use registers to configure the SAR ADC; this is the most common usage. Detailed register bit definition is available in the *PSoC 4500S: PSoC 4 Registers TRM*.

20.3.9.1 SARMUX Analog Routing

There are two ways to control the SARMUX analog routing: sequencer and firmware.

Sequencer Control

It is essential that the appropriate hardware control bits in the MUX_SWITCH_HW_CTRL register and the firmware control bits in the MUX_SWITCH0 register are both set to '1'. Ensure that SWITCH_DISABLE=0; setting SWITCH_DISABLE disables sequencer control.

With sequencer control, the pin or internal signal a channel converts is specified by the combination of port and pin address. The PORT_ADDR bits are SARx_CHANx_CONFIG [6:4] and PIN_ADDR bits are SARx_CHANx_CONFIG [2:0]. Table 20-10 shows the PORT_ADDR and PIN_ADDR setup with corresponding SARMUX selection. The unused port/pins are reserved for other products in the PSoC 4500S series.

Table 20-10. PORT_ADDR and PIN_ADDR

PORT_ADDR	PIN_ADDR	Description
0	0..7	8 dedicated pins of the SARMUX
1	X	sarbus0[a]
1	X	sarbus1[a]
7	0	Temperature sensor
7	2	AMUXBUS-A
7	3	AMUXBUS-B

a. sarbus0 and sarbus1 connect to the output of the CTBm block, which contains opamp0/1. See the Continuous Time Block mini (CTBm) chapter on page 234 for more information. When PORT_ADDR=1, sarbus0 connects to positive terminal of SAR ADC regardless of the value of PIN_ADDR; sarbus1 can only connect to the negative terminal of SAR ADC when differential mode is enabled and PORT_ADDR=1.

For differential conversion, the negative terminal connection is dependent on the positive terminal connection, which is defined by PORT_ADDR and PIN_ADDR. By setting DIFFERENTIAL_EN, the channel will do a differential conversion on the even/odd pin pair specified by the pin address with PIN_ADDR [0] ignored. P.0/P.1, P.2/P.3, P.4/P.5, and P.6/P.7 are valid differential pairs for sequencer control. More flexible analog can be implemented by firmware.

For single-ended conversions, NEG_SEL (SARx_CTRL [11:9]) is intended to decide which signal is connected to the negative input. In differential mode, these bits are ignored. The negative input choice affects the input voltage range and effective resolution. See Negative Input Selection on page 201 for details. The options include: V_{SSA}, V_{REF}, or an external input from any of the eight pins with SARMUX connectivity. To connect the negative input to V_{REF}, an additional bit, SARx_HW_CTRL_NEGVREF (SARx_CTRL[13]) must be set, because the MUX_SWITCH_HW_CTRL register does not have that hardware control bit.

Firmware Control

By default, SARMUX operates in firmware control. VPLUS (positive) and VMINUS (negative) inputs of the SAR ADC can be controlled separately by setting the appropriate bits in SARx_MUX_SWITCH0 [29:0]. Clear appropriate bits in the hardware switch control register (SARx_MUX_SWITCH_HW_CTR[n]=0). Otherwise, the hardware control method (sequencer) will control SARMUX analog routing.

The SARx_CTRL register bit SWITCH_DISABLE is used to disable the SAR sequencer from enabling routing switches. Note that firmware control mode can always close switches independent of this bit value; however, it is recommended to set it to '1'.

NEG_SEL (SARx_CTRL [11:9]) decides which signal is connected to the negative terminal (VMINUS) of the SAR ADC in single-ended mode. In differential mode, these bits are ignored. In single-ended mode, when using sequencer control, you must set these bits. When using firmware control, NEG_SEL is ignored and SARx_MUX_SWITCH0 should be set to control the negative input. A special case is when SARx_MUX_SWITCH0 does not connect internal V_{REF} to VMINUS; then, set NEG_SEL to '7'. Negative input choice affects the input voltage range, SNR, and effective resolution. See Negative Input Selection on page 201 for details.

20.3.9.2 Global SARSEQ Configuration

A number of conversion options that apply to all channels are configured globally. In several cases, the channel configuration has bits to choose what parts of the global configuration to use.

SARx_CTRL, SARx_SAMPLE_CTRL, SARx_SAMPLE01, SARx_SAMPLE23, SARx_RANGE_THES, and SARx_RANGE_COND are global configuration registers. Typically, these configurations should not be modified while a scan is in progress. If configuration settings that are in use are changed, the results are undefined. Configuration settings that are not currently in use can be changed without affecting the ongoing scan.

Table 20-11. Global Configuration Registers

Configurations	Control Registers	Detailed Reference
Reference selection	SARx_CTRL[6:4]	20.3.3.1 Reference Options
Signed/unsigned selection	SARx_SAMPLE_CTRL[3:2]	20.3.1.3 Result Data Format
Data left/right alignment	SARx_SAMPLE_CTRL[1]	20.3.1.3 Result Data Format
Negative input selection in single-ended mode	SARx_CTRL[11:9]	20.3.1.4 Negative Input Selection
Resolution	SARx_SAMPLE_CTRL[0][a]	20.3.1.5 Resolution
Acquisition time	SARx_SAMPLE_TIME01[25:0] SARx_SAMPLE_TIME32[25:0]	20.3.1.6 Acquisition Time
Averaging count	SARx_SAMPLE_CTRL[7:4]	20.3.4.1 Averaging
Range detection	SARx_RANGE_THRES[31:0] SARx_RANGE_COND[31:30]	20.3.4.2 Range Detection

a. The alternate resolution should be enabled by the SAR_RESOLUTION bit in the SAR_CHAN_CONFIG register. If the alternate resolution is not enabled, the ADC operates at 12-bit resolution, irrespective of the resolution set by the SAR_SAMPLE_CTRL register.

20.3.9.3 Channel Configurations

Channel configuration includes:

- Differential or single-ended mode selection
- Global configuration selection: sample time, resolution, averaging enable
- DSI output enable

As a general rule, channel configurations should only be updated between scans (same as global configurations). However, if a channel is not enabled for the ongoing scan, the configuration for that channel can be changed freely without affecting the ongoing scan. If this rule is violated, the results are undefined. The channels that enable themselves are the only exception to this rule; enabled channels can be changed during the on-going scan, and it will be effective in the next scan. Changing the enabled channels may change the sample rate.

Table 20-12. Channel Configuration Registers

Configurations	Registers	Detailed Reference
Single-ended/differential	SARx_CHAN_CONFIGx [8]	20.3.1.1 Single-ended and Differential Mode
Acquisition time selection	SARx_CHAN_CONFIGx [13:12]	20.3.1.6 Acquisition Time
Resolution selection	SARx_CHAN_CONFIGx [9]	20.3.1.5 Resolution
Average enable	SARx_CHAN_CONFIGx [10]	20.3.4.1 Averaging

SUB_RESOLUTION (SARx_SAMPLE_CTRL[0]) can choose which alternate resolution will be used, either 8-bit or 10-bit. Resolution (SARx_CHAN_CONFIGx [9]) can determine whether the default resolution (12-bit) or alternate resolution is used. When averaging is enabled, the SUB_RESOLUTION is ignored; the resolution will be fixed to the maximum 12-bit.

Table 20-13. Resolution

Average	SUB_RESOLUTION SARx_SAMPLE_CTRL[0]	Register Mode Resolution SARx_CHAN_CONFIGx [9]	Channel Resolution
OFF	0	1	8-bit
OFF	1	1	10-bit
OFF	0	0	12-bit
OFF	1	0	12-bit
ON	X	X	12-bit

20.3.9.4 Channel Enables

A CHAN_EN register is available to individually enable each channel. All enabled channels are scanned when the next trigger happens. After a trigger, the channel enables can immediately be updated to prepare for the next scan. This action does not affect the ongoing scan. Note that this is an exception to the rule; all other configurations (global or channel) should not be changed while a scan is in progress.

20.3.9.5 Interrupt Masks

There are six interrupt sources; all have an interrupt mask:

- End-of-scan interrupt
- Overflow interrupt
- Collision interrupt
- Injection EOC interrupt
- Range detection interrupt
- Saturate detection interrupt

Each interrupt has an interrupt request register (INTR, SATURATE_INTR, RANGE_INTR), a software interrupt set register (INTR_SET, SATURATE_INTR_SET, RANGE_INTR_SET), an interrupt mask register (INTR_MASK, SATURATE_INTR_MASK, RANGE_INTR_MASK), and an interrupt request masked result register (INTR_MASKED, SATURATE_INTR_MASKED, RANGE_INTR_MASKED). An interrupt cause register is also added to have an overview of all the currently pending SAR interrupts and allows the ISR to determine the interrupt cause by just reading this register.

See 20.3.5 SAR Interrupts for details.

20.3.9.6 Trigger

The three ways to start an A/D conversion are:

- Firmware trigger: SARx_START_CTRL [0]
- External trigger: dsi_trigger
- Continuous trigger: SARx_SAMPLE_CTRL [16]

See 20.3.6 Trigger for details.

20.3.9.7 Retrieve Data after Each Interrupt

Make sure that you read the data from the result register after each scan; otherwise, the data may change because of the next scan's configuration.

The 16-bit data registers are used to implement double buffering for up to eight channels (injection channel does not have double buffer). Double buffering means that there is one working register and one result register for each channel. Data is written to the working register immediately after sampling this channel. It is then copied to the result register from the working register after all enabled channels in this scan have been sampled.

The CHAN_WORK_VALID bit is set after the corresponding WORK data is valid; that is, it was already sampled during the current scan. Corresponding CHAN_RESULT_VALID is set after completed scan. When CHAN_RESULT_VALID is set, the corresponding CHAN_WORK_VALID bit is cleared.

For firmware convenience, bit [31] in the SARx_CHAN_WORK register is the mirror bit of the corresponding bit in the SARx_CHAN_WORK_VALID register. Bit[29], bit [30], and bit[31] in SARx_CHAN_RESULT are the mirror bits of the corresponding bit in SARx_SATURATE_INTR, SARx_RANGE_INTR, and SARx_CHAN_RESULT_VALID registers. Note that the interrupt bits mirrored here are the raw (unmasked) interrupt bits. It helps firmware to check whether the data is valid by just reading the data register.

20.3.9.8 Injection Conversions

The injection channel can be triggered by setting the start bit INJ_START_EN (INJ_CHAN_CONFIG [31]). To prevent the collision of a regular automatic scan, it is recommended to enable tailgating by setting INJ_CHAN_CONFIG [30]. When it is enabled, INJ_START_EN will enable the injection channel to be scanned at the end of next scan of regular channels. See 20.3.4.4 Injection Channel for details.

20.3.10 Temperature Sensor Configuration

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Differential conversions are not available for temperature sensors (conversion result is undefined). Therefore, always use it in single-ended mode. The reference is from internal 1.2 V.

A pin or signal can be routed to the SAR ADC in three ways. Table 20-14 lists the methods to route temperature sensors to the SAR ADC. Setting the MUX_FW_TEMP_VPLUS bit (SARx_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of the SAR ADC; clearing this bit disables the temperature sensor by cutting its bias current.

Table 20-14. Route Temperature to SAR ADC

Control Methods	Setup
Sequencer	DIFFERENTIAL_EN = 0 (SARx_CHAN_CONFIGx[8]) VREF_SEL = 0 (SARx_CTRL[6:4]) PORT_ADDR = 7 (SARx_CHAN_CONFIGx[6:4]) PIN_ADDR = 0 (SARx_CHAN_CONFIGx[2:0]) SWITCH_DISABLE = 0 (SARx_CTRL[30]) SARx_MUX_SWITCH0[16] = 1 SARx_MUX_SWITCH0[17] = 1 SARx_MUX_SWITCH_HW_CTRL[16]= 1 SARx_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
Firmware	DIFFERENTIAL_EN = 0 (SARx_CHAN_CONFIGx[8]) VREF_SEL = 0 (SARx_CTRL[6:4]) SWITCH_DISABLE = 1 (SARx_CTRL[30]) SARx_MUX_SWITCH0[16] = 1 SARx_MUX_SWITCH0[17] = 1 SARx_MUX_SWITCH_HW_CTRL[16]= 0 SARx_MUX_SWITCH_HW_CTRL[17]= 0 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]

a. For temperature sensor, override NEL_SEG (SARx_CTRL [11:9]) to ‘0’.

20.4 Registers

Name	Offset	Qty.	Width	Description
SARx_CTRL	0x0000	1	32	Analog control register.
SARx_SAMPLE_CTRL	0x0004	1	32	Sample control register.
SARx_SAMPLE_TIME01	0x0010	1	32	Sample time specification ST0 and ST1
SARx_SAMPLE_TIME23	0x0014	1	32	Sample time specification ST2 and ST3
SARx_RANGE_THRES	0x0018	1	32	Global range detect threshold register.
SARx_RANGE_COND	0x001C	1	32	Global range detect mode register.
SARx_CHAN_EN	0x0020	1	32	Enable bits for the channels
SARx_START_CTRL	0x0024	1	32	Start control register (firmware trigger).
SARx_CHAN_CONFIG	0x0080	16	32	Channel configuration register.
SARx_CHAN_WORK	0x0100	16	32	Channel working data register
SARx_CHAN_RESULT	0x0180	16	32	Channel result data register
SARx_CHAN_WORK_VALID	0x0200	1	32	Channel working data register valid bits
SARx_CHAN_RESULT_VALID	0x0204	1	32	Channel result data register valid bits
SARx_STATUS	0x0208	1	32	Current status of internal SAR registers (mostly for debug)
SARx_AVG_STAT	0x020C	1	32	Current averaging status (for debug)
SARx_INTR	0x0210	1	32	Interrupt request register.
SARx_INTR_SET	0x0214	1	32	Interrupt set request register
SARx_INTR_MASK	0x0218	1	32	Interrupt mask register.
SARx_INTR_MASKED	0x021C	1	32	Interrupt masked request register
SARx_SATURATE_INTR	0x0220	1	32	Saturate interrupt request register.
SARx_SATURATE_INTR_SET	0x0224	1	32	Saturate interrupt set request register
SARx_SATURATE_INTR_MASK	0x0228	1	32	Saturate interrupt mask register.
SARx_SATURATE_INTR_MASKED	0x022C	1	32	Saturate interrupt masked request register
SARx_RANGE_INTR	0x0230	1	32	Range detect interrupt request register.
SARx_RANGE_INTR_SET	0x0234	1	32	Range detect interrupt set request register
SARx_RANGE_INTR_MASK	0x0238	1	32	Range detect interrupt mask register.
SARx_RANGE_INTR_MASKED	0x023C	1	32	Range interrupt masked request register
SARx_INTR_CAUSE	0x0240	1	32	Interrupt cause register
SARx_INJ_CHAN_CONFIG	0x0280	1	32	Injection channel configuration register.
SARx_INJ_RESULT	0x0290	1	32	Injection channel result register
SARx_MUX_SWITCH0	0x0300	1	32	SARMUX Firmware switch controls
SARx_MUX_SWITCH_CLEAR0	0x0304	1	32	SARMUX Firmware switch control clear
SARx_MUX_SWITCH_HW_CTRL	0x0340	1	32	SARMUX switch hardware control
SARx_MUX_SWITCH_STATUS	0x0348	1	32	SARMUX switch status
SARx_PUMP_CTRL	0x0380	1	32	Switch pump control

21. Low-Power Comparator

PSoC 4 devices have two Low-Power comparators. These comparators can perform fast analog signal comparison in all system power modes. See the Power Modes chapter on page 101 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.

21.1 Features

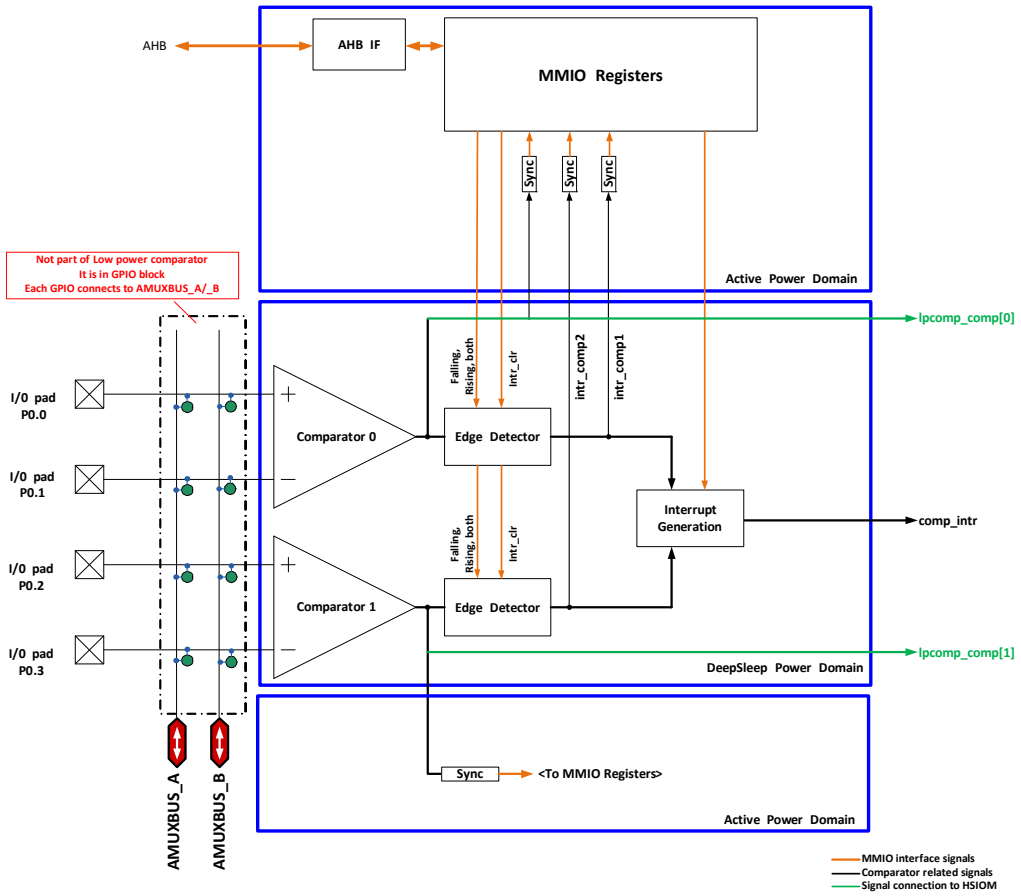
The PSoC 4 comparators support the following features:

- Configurable positive and negative inputs
- Programmable power and speed
- Ultra-Low-Power (ULP) mode support (< 4 µA)
- Optional 10-mV input hysteresis
- Low input offset voltage (< 4 mV after trim)
- Wakeup source in Deep Sleep mode

21.2 Block Diagram

Figure 21-1 shows the block diagram for the Low-Power comparator.

Figure 21-1. Low-Power Comparator Block Diagram



21.3 How it Works

The following sections describe the operation of the PSoC 4 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.

21.3.1 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 21-1, note that P0.0 and P0.1 connect to positive and negative inputs of Comparator 0; P0.2 and P0.3 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 LPCOMP 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 65 for more details on connecting the GPIO to AMUXBUS A/B or setting up the GPIO for comparator input.

21.3.2 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 (see Table 21-1). 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 21-1.

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. Refer to I/O System on page 65 for details on HSIOM. For details on the pins that support the LPCOMP output, refer to the *PSoC 4500S 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 21-1). 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 21-1), which is a logical OR of the two interrupts and mapped as the LPCOMP block's interrupt in the CPU NVIC. See the Interrupts chapter on page 52 for details. If both the comparators are used in a design, the COMP1, COMP2, or both bits of the LPCOMP_INTR register need to be read in the ISR 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 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 21-1. 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

21.3.3 Power Mode and Speed Configuration

The Low-Power comparators can operate in three power modes:

- Fast
- Slow
- ULP

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 ULP mode; response time is fastest in fast mode and slowest in ULP mode. See the *PSoC 4500S 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 21-2.

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 21-2. 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: ULP 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: ULP 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

21.3.4 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 21-3.

Table 21-3. 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

21.3.5 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 Deep Sleep 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.

21.3.6 Comparator Clock

The comparator uses the system main clock SYSCLK as the clock for interrupt synchronization.

21.3.7 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 the 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}.

21.4 Register Summary

Table 21-4. LPCOMP 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

22. CapSense

The CapSense system can measure the self-capacitance of an electrode or the mutual capacitance between a pair of electrodes. In addition to capacitive sensing, the CapSense system can function as an ADC to measure voltage on any GPIO pin that supports the CapSense functionality.

The CapSense touch sensing method in PSoC 4, which senses self-capacitance, is known as CapSense Sigma Delta (CSD). Similarly, the mutual-capacitance sensing method is known as CapSense Cross-point (CSX). The CSD and CSX touch sensing methods provide the industry's best-in-class SNR, high touch sensitivity, low-power operation, and superior EMI performance.

CapSense touch sensing is a combination of hardware and firmware techniques. Therefore, use the CapSense component provided by the PSoC Creator IDE to implement CapSense designs. See the *PSoC 4 and PSoC 6 MCU CapSense Design Guide* for more details.

23. Continuous Time Block mini (CTBm)

The CTBm provides discrete operational amplifiers (opamps) inside the chip for use in continuous-time signal chains. The PSoC 4500S device has two CTBm blocks (CTBm 0 and CTBm 1). Each CTBm has a switch matrix for input/output configuration, two identical opamps, which are also configurable as two comparators, a charge pump inside each opamp, and a digital interface for comparator output routing, switch controls, and interrupts. CTBm blocks can be operational in Deep Sleep power mode.

23.1 Features

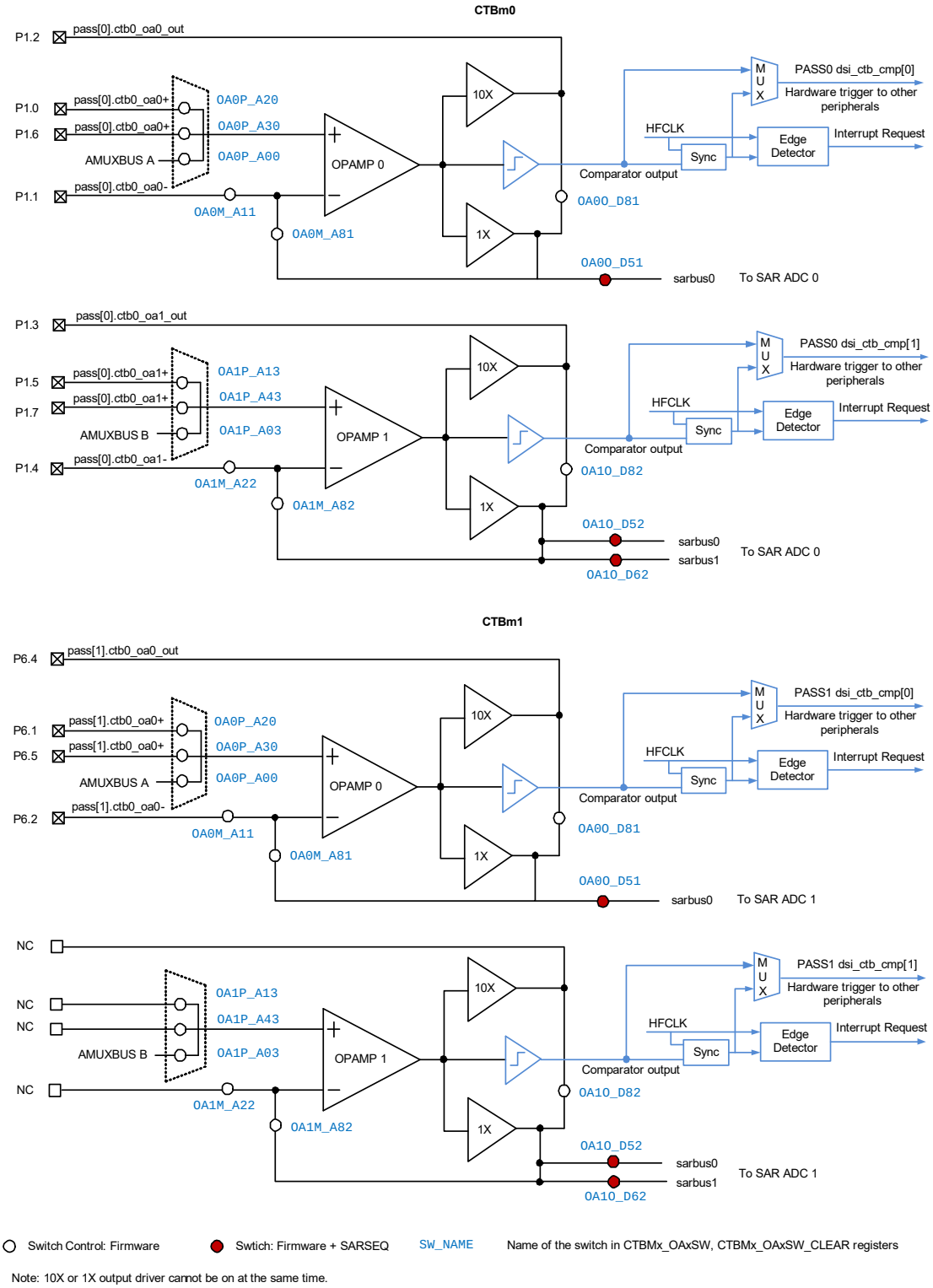
The opamps in the PSoC 4500S CTBm block have the following features:

- Discrete, high-performance, and highly configurable on-chip amplifiers
- Programmable power, bandwidth, compensation, and output drive strength
- 1-mA or 10-mA selectable output current drive capability
- 6-MHz gain bandwidth for 20-pF load
- Less than 1-mV offset with trim
- Support for opamp follower mode
- Comparator mode with optional 10-mV hysteresis
- Buffer/pre-amplifier for SAR inputs
- Support in Deep Sleep device power mode

23.2 Block Diagram

Figure 23-1 illustrates the block diagram for the CTBm block available in the PSoC 4500S devices.

Figure 23-1. CTBm Block Diagram



Note: 10X or 1X output driver cannot be on at the same time.

23.3 How it Works

As the block diagram shows, each CTBm has two identical opamps. Each opamp has one input and three output stages, all of which share a common input stage, as shown in Figure 23-1; only one of them can be selected at a time. The output stage can be operated as Class-A(1X), Class-AB(10X), or comparator. The other configurable features are power and speed, compensation, and switch routing control.

To use the CTBm block, the first step is to set up external components (such as resistors), if required. Then, enable the block by setting the CTBMx_CTB_CTRL [31] bit. To have almost rail-to-rail input range and minimal distortion common mode input, there is one charge pump inside each opamp. The charge pump can be enabled by setting the CTBMx_OA_RES0_CTRL [11] bit for opamp0 and CTBMx_OA_RES1_CTRL [11] bit for opamp1.

After enabling the opamps and charge pumps, follow these steps to set up the amplifier:

1. Configure power mode.
2. Configure output strength.
3. Configure compensation.
4. Configure input switch.
5. Configure output switch, especially when opamp output needs to be connected to SAR ADC.

Follow these steps to set up a comparator:

1. Configure the power mode.
2. Configure the input switch.
3. Configure the comparator output circuitry, as required - interrupt generation, output, and so on.
4. Configure hysteresis and enable the comparator.

23.3.1 Power Mode Configuration

The opamp can operate in three power modes – low, medium, and high. CTBm adjusts the power consumed by adjusting the reference currents coming into the opamp. Power modes are configured using the PWR_MODE bits [1:0] in CTBMx_OA_RESx_CTRL. The slew rate and gain bandwidth are maximum in high-power mode and minimum in Low-Power mode. Note that power mode configuration also affects the maximum output drive capability (I_{OUT}) in 1X mode. See Table 23-1 for details. See the *PSoC 4500S datasheet* for gain bandwidth, slew rate, and I_{OUT} specifications in various power modes.

23.3.2 Output Strength Configuration

The output driver of each opamp can be configured to internal driver (Class A/1X driver) or external driver (Class AB/10X driver). 1X and 10X drivers are mutually exclusive – they cannot be active at the same time. 1X output driver is suited to drive smaller on-chip capacitive and resistive loads at higher speeds. The 10X output driver is useful for driving large off-chip capacitive and resistive loads. The 1X driver output is routed to sarbus 0/1 and 10X driver output is routed to an external pin. Each driver mode has a low, medium, or high power mode, as shown in Table 23-1.

Table 23-1. Output Driver versus Power Mode

Power Mode I_{OUT} Drive Capability	CTBM_OA_RESx_CTRL[1:0]			
	00 (disable)	01 (low)	10 (medium)	11 (high)
External Driver (10X)	OFF	10 mA	10 mA	10 mA
Internal Driver (1X)	OFF	100 µA	400 µA	1 mA

The CTBMx_OA_RESx_CTRL[2] bit is used to select between the 10X and 1X output capability (0: 1X, 1: 10X). If the output of the opamp is connected to the SAR ADC, it is recommended to choose the 1X output driver. If the output of the opamp is connected to an external pin, choose the 10X output driver. In special instances, to connect the output to an external pin with 1X output driver or an internal load (for example, SAR ADC) with 10X output driver, set CTBMx_OAx_SW [21] to '1'. However, Cypress does not guarantee performance in this case.

Table 23-2 summarizes the bits used to configure the opamp output drive strength and power modes.

Table 23-2. Output Strength and Power Mode Configuration in CTBM Registers

Register[Bit_Pos]	Bit_Name	Description
CTBM0_CTB_CTRL[31]	ENABLE	CTBm power mode selection
		0: CTBm is disabled
		1: CTBm is enabled
CTBM0_OA_RES0_CTRL [11]	OA0_PUMP_EN	Opamp0 pump enable bit
		0: Opamp0 pump is disabled
		1: Opamp0 pump is enabled
CTBM0_OA_RES1_CTRL [11]	OA1_PUMP_EN	Opamp1 pump enable bit
		0: Opamp1 pump is disabled
		1: Opamp1 pump is enabled
CTBM0_OA_RES0_CTRL [1:0]	OA0_PWR_MODE	Opamp0 power mode select bits
		00: Opamp0 is OFF
		01: Opamp0 is in Low-Power mode
		10: Opamp0 is in medium-power mode
		11: Opamp0 is in high-power mode
CTBM0_OA_RES1_CTRL [1:0]	OA1_PWR_MODE	Opamp1 power mode select bits
		00: Opamp1 is OFF
		01: Opamp1 is in Low-Power mode
		10: Opamp1 is in medium-power mode
		11: Opamp1 is in high-power mode
CTBM0_OA_RES0_CTRL [2]	OA0_DRIVE_STR_SEL	Opamp0 output drive strength select bits
		0: Opamp0 output drive strength is 1X
		1: Opamp0 output drive strength is 10X
CTBM0_OA_RES1_CTRL [2]	OA1_DRIVE_STR_SEL	Opamp1 output drive strength select bits
		0: Opamp1 output drive strength is 1X
		1: Opamp1 output drive strength is 10X
CTBM1_CTB_CTRL[31]	ENABLE	CTBm power mode selection
		0: CTBm is disabled
		1: CTBm is enabled
CTBM1_OA_RES0_CTRL [11]	OA0_PUMP_EN	Opamp0 pump enable bit
		0: Opamp0 pump is disabled
		1: Opamp0 pump is enabled
CTBM1_OA_RES1_CTRL [11]	OA1_PUMP_EN	Opamp1 pump enable bit
		0: Opamp1 pump is disabled
		1: Opamp1 pump is enabled
CTBM1_OA_RES0_CTRL [1:0]	OA0_PWR_MODE	Opamp0 power mode select bits
		00: Opamp0 is OFF
		01: Opamp0 is in Low-Power mode
		10: Opamp0 is in medium-power mode
		11: Opamp0 is in high-power mode

Table 23-2. Output Strength and Power Mode Configuration in CTBM Registers *(continued)*

Register[Bit_Pos]	Bit_Name	Description
CTBM1_OA_RES1_CTRL [1:0]	OA1_PWR_MODE	Opamp1 power mode select bits
		00: Opamp1 is OFF
		01: Opamp1 is in Low-Power mode
		10: Opamp1 is in medium-power mode
		11: Opamp1 is in high-power mode
CTBM1_OA_RES0_CTRL [2]	OA0_DRIVE_STR_SEL	Opamp0 output drive strength select bits
		0: Opamp0 output drive strength is 1X
		1: Opamp0 output drive strength is 10X
CTBM1_OA_RES1_CTRL [2]	OA1_DRIVE_STR_SEL	Opamp1 output drive strength select bits
		0: Opamp1 output drive strength is 1X
		1: Opamp1 output drive strength is 10X

23.3.3 Compensation

Each opamp also has a programmable compensation capacitor block, which allows optimizing the stability of the opamp performance based on output load. The compensation of each opamp is controlled by the respective CTBM_OAx_COMP_TRIM register, as explained in Table 23-3. Note that all GBW slew rate specifications in the *PSoC 4500S datasheet* are applied for all compensation trims.

Table 23-3. Opampx (Opamp0 ~ Opamp3) Compensation Bits in CTBm

Register[Bit_Pos]	Bit_Name	Description
CTBMx_OAx_COMP_TRIM[1:0]	OAx_COMP_TRIM	Opampx compensation trim bits
		00: No compensation
		01: Minimum compensation, high speed, and low stability
		10: Medium compensation, balanced speed, and stability
		11: Maximum compensation, low speed, and high stability

23.3.4 Switch Control

The CTBm has many switches to configure the opamp input and output. Most of them are controlled by configuring CTBm registers (CTBMx_OA0_SW, CTBMx_OA1_SW), except three switches, which are used to connect the output of opamps to SAR ADC through sarbus0 and sarbus1. They must be controlled by SAR ADC registers, and CTBm registers.

Switches can be closed by setting the corresponding bit in register CTBMx_OAx_SW; clearing the bit will cause the corresponding switches to open. To open the switch, write '1' to CTBMx_OAx_SW_CLEAR, which clears the corresponding bit in CTBMx_OAx_SW. See the *PSoC 4500S: PSoC 4 Registers TRM* for details on the switches and the connections they enable.

23.3.4.1 Input Configuration

Positive and negative inputs to the operational amplifier can be selected from several options through analog switches. These switches serve to connect the opamp inputs from the external pins or AMUX buses, or to form a local feedback loop (for buffer function). Each opamp has a switch connecting to one of the two AMUXBUS line: Opamp0 connects to AMUXBUS-A and Opamp1 connects to AMUXBUS-B.

Note: Only one switch should be closed for the positive and negative input paths; otherwise, different input sources may short together.

- Positive input: Both opamp0 ~ opamp3 have three positive input options through analog switches: two external pins and one AMUXBUS line. See Table 23-4 for details.

Table 23-4. Positive Input Selection

CTBm	Opamp	Positive Input	Switch Control Bit	Description
CTBm0	Opamp0	AMUXBUSA	CTBM_OA0_SW [0]	0: open 1: close switch
		P1.0	CTBM_OA0_SW [2]	0: open 1: close switch
		P1.6	CTBM_OA0_SW [3]	0: open 1: close switch
	Opamp1	AMUXBUSB	CTBM_OA1_SW [0]	0: open 1: close switch
		P1.5	CTBM_OA1_SW [1]	0: open 1: close switch
		P1.7	CTBM_OA1_SW [4]	0: open 1: close switch
CTBm1	Opamp0	AMUXBUSA	CTBM_OA0_SW [0]	0: open 1: close switch
		P6.1	CTBM_OA0_SW [2]	0: open 1: close switch
		P6.5	CTBM_OA0_SW [3]	0: open 1: close switch
	Opamp1	AMUXBUSB	CTBM_OA1_SW [0]	0: open 1: close switch
		NC	CTBM_OA1_SW [1]	0: open 1: close switch
		NC	CTBM_OA1_SW [4]	0: open 1: close switch

- Negative input: Both opamp0 ~ opamp3 have two negative input options through analog switches: one external pin or output feedback, which is controlled by the CTBMx_OAx_SW register. Table 23-5 shows the control bits.

Table 23-5. Negative Input Selection

CTBm	Opamp	Positive Input	Switch Control Bit	Description
CTBm0	Opamp0	P1.1	CTBM_OA0_SW [8]	0: open 1: close switch
		Opamp0 output feedback through 1X output driver	CTBM_OA0_SW [14]	0: open 1: close switch
	Opamp1	P1.4	CTBM_OA1_SW [8]	0: open 1: close switch
		Opamp1 output feedback through 1X output driver	CTBM_OA1_SW [14]	0: open 1: close switch
CTBm1	Opamp0	P6.2	CTBM_OA0_SW [8]	0: open 1: close switch
		Opamp0 output feedback through 1X output driver	CTBM_OA0_SW [14]	0: open 1: close switch
	Opamp1	NC	CTBM_OA1_SW [8]	0: open 1: close switch
		Opamp1 output feedback through 1X output driver	CTBM_OA1_SW [14]	0: open 1: close switch

23.3.4.2 Output Configuration

Each opamp's output is connected directly to a fixed pin; no additional setup is needed. Optionally, it can be connected to sarbus0 or sarbus1 through three switches (SW1/2/3). The opamp0 output can be connected to sarbus0 and opamp1 can be connected to sarbus0 or sarbus1. sarbus0 ~ sarbus3 are intended to connect the opamp output to the SAR ADC input MUX. The three output routing switches to sarbus are controlled by the SAR ADC registers and the CTBm register together; other switches can be controlled only by the CTBm register.

The truth tables (Table 23-6, Table 23-7, and Table 23-8) show the control logic of the three switches. PORT_ADDR, PIN_ADDR, and DIFFERENTIAL_EN are from SARx_CHAN_CONFIGx [6:4], SARx_CHAN_CONFIGx [2:0], and SARx_CHAN_CONFIGx [2:0], respectively. Either PORT_ADDR = 0 or PIN_ADDR = 0 will set SW[n]=0. CTBMx_SW_HW_CTRL bit [2] or [3] should be set when using the SAR register to control switches. CTBMx_OAx_SW[18]/[19] can mask the other control bits – if CTBMx_OAx_SW[18]/[19] = 0, SW[n] = 0.

The CTBMx_SW_STATUS [30:28] register gives the current switch status of SW1/2/3.

Table 23-6. Truth Table of SW1 Control Logic

PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[2]	CTBMx_OA0_SW[18]	SW1
X	X	X	0	0
X	0	1	1	0
0	X	1	1	0
X	X	X	1	1
X	X	0	1	1
1	2	X	1	1

Table 23-7. Truth Table of SW2 Control Logic

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[3]	CTBMx_OA0_SW[18]	SW2
X	X	X	X	0	0
X	X	0	1	1	0
X	0	X	1	1	0
1	X	X	X	1	0
X	X	X	0	1	1
X	X	X	X	1	1
0	1	3	X	1	1

23.3.4.3 Comparator Mode

Each opamp can be configured as a comparator by setting the respective CTBMx_OA_RESx_CTRL[4] bit. Note that enabling the comparator completely disables the compensation capacitors and shuts down the Class A (1X) and Class AB (10X) output drivers. The comparator has the following features:

- Optional 10-mV input hysteresis
- Configurable power/speed
- Optional comparator output synchronization
- Offset trimmed to less than 1 mV
- Configurable edge detection (rising/falling/both/disable)

23.3.4.4 Comparator Configuration

The hysteresis of 10 mV ±5% can be enabled in one direction (LOW to HIGH). Input hysteresis can be enabled by setting CTBMx_OA_RESx_CTRL[5]. The four comparators also have three power modes: low, medium, and high, controlled by setting CTBMx_OA_RESx_CTRL [1:0]. Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ULP mode. Exact specifications for power consumption and response time are provided in the datasheet.

The synchronization of the comparator output with the system AHB clock can be configured in CTBMx_OA_RESx_CTRL[6].

The output state of comparator0 ~ comparator3 are stored in CTBMx_COMP_STAT[0] and CTBMx_COMP_STAT[16], respectively.

Table 23-8 summarizes various bits used to configure the comparator mode in the CTBM block.

Table 23-8. Comparator Mode and Configuration Register Settings

Register[Bit_Pos]	Bit_Name	Description
CTBMx_OA_RESyx_CTRL[4]	OAx_COMP_EN	Opampx comparator enable bit 0: Comparator mode is disabled in opampx 1: Comparator mode is enabled in opampx
CTBMx_OA_RESx_CTRL[5]	OAx_HYST_EN	Opampx Comparator hysteresis enable bit 0: Hysteresis is disabled in opampx 1: Hysteresis is enabled in opampx
CTBMx_OA_RESx_CTRL[6]	OAx_BYPASS_DSI_SYNC	Opampx bypass comparator output synchronization for DSI (trigger) output 0: Synchronize (level or pulse) 1: Bypass
CTBMx_OA_RESx_CTRL[7]	OAx_ DSI_LEVEL	Opampx comparator DSI (trigger) output synchronization level 0: Pulse 1: Level

23.3.4.5 Comparator Interrupt

The comparator output is connected to an edge detector block, which is used to detect the edge (disable/rising/falling/both) that generates interrupt. It can be configured by the CTBMx_OA_RESx_CTRL[9:8] bits.

Each comparator has a separate IRQ. CTBMx_INTR [0] is for comparator0 IRQ, CTBMx_INTR [1] is for comparator1 IRQ. CTBMx_INTR [2] is for comparator2 IRQ and CTBMx_INTR [3] is for comparator3 IRQ. Though each comparator have different IRQ bits, they all share a single CTBM ISR mapped in the CPU NVIC. See the Interrupts chapter on page 52 for details. You can check which comparator(s) triggered the ISR by polling the CTBMx_INTR bits.

Each interrupt has an interrupt mask bit in the CTBMx_INTR_MASK register. By setting the interrupt mask LOW, the corresponding interrupt source is ignored. The CTBm comparator interrupt to the NVIC will be raised if the logic AND of the interrupt flags in CTBMx_INTR registers and the corresponding interrupt masks in CTBMx_INTR_MASK register is ‘1’.

Writing a ‘1’ to the CTBMx_INTR bit [1:0] can clear the corresponding interrupt.

For firmware convenience, the intersection (logic AND) of the interrupt flags and the interrupt masks is also made available in the CTBMx_INTR_MASKED register.

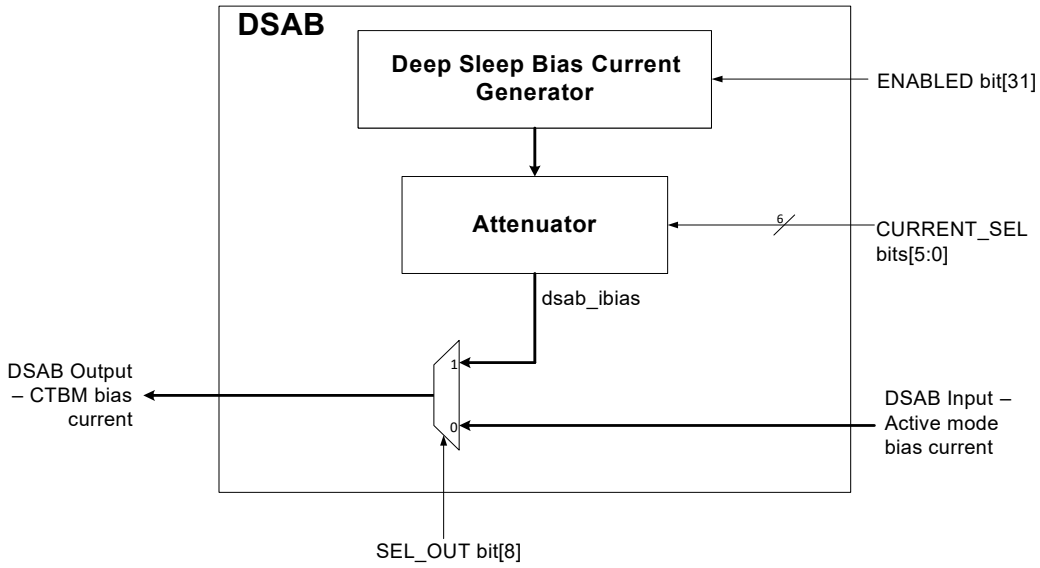
For verification and debug purposes, a set bit is provided for each interrupt in the CTBMx_INTR_SET register. This bit allows the firmware to raise the interrupt without a real comparator switch event.

23.3.4.6 Deep Sleep Mode Operation

In Deep Sleep mode, the block that provides the bias current, reference voltage, and IMO clock is turned off. As a result, the CTBm functionality, which relies on the bias current and IMO clock for its operation is not available. See the Power Modes chapter on page 101 for details on various power modes and blocks available in each mode. To support the functionality of the CTBm during Deep Sleep, an alternate bias current is generated by a special block called the Deep Sleep Amplifier Bias (DSAB) block. This current allows the opamps in the CTBm to be functional in Deep Sleep mode.

Figure 23-2 shows the architecture of the DSAB block. This block receives the Active mode bias current as input. It outputs the bias current that is fed to the opamp bias circuitry. In Active mode, the DSAB block acts similar to a pass-through block and routes the bias current from the input to the output. In Deep Sleep mode, if enabled, the DSAB generates the alternate bias current, attenuates the output to a user-selected value, and provides the bias current for the CTBm at its output. If the DSAB block is disabled, the output is always connected to the input bias current and the alternate bias current is not generated during Deep Sleep. The opamps will not be functional in Deep Sleep mode if the DSAB block is disabled. The ENABLED bit [31] of the PASS_DSAB_DSAB_CTRL register enables/disables the block; the CURRENT_SEL bits [5:0] selects the output bias current value. The value selected is CURRENT_SEL × 0.075 µA (±5%). The SEL_OUT bit[8] is used to control the selection between the two bias currents, which can be routed to the CTBm bias. Table 23-9 summarizes the bit configuration settings of the PASS_DSAB_DSAB_CTRL register.

Figure 23-2. Deep Sleep Amplifier Bias Block Diagram



This feature is useful in designs that require the opamp-based circuitry to remain active in Low-Power modes, such as Deep Sleep, to save power. For instance, in a battery-operated system (such as a heart-rate monitor) that requires always-on opamps, substantial power savings can be achieved if the rest of the chip can go into Deep Sleep mode and only wake up as needed. Note that the bias current provided by the DSAB block does not meet the accuracy and stability of the Active mode bias current. In addition, the DSAB does not generate an alternate clock. As a result, none of the switch or opamp-related charge pumps are activated. Consequently, the highest input common-mode voltage of the opamps is limited to approximately V_{DDA} – 1.3 V. In addition, because of the unavailability of switch pumps (required for analog switches when operating below 3.3 V), the on-resistance of the analog switches increase beyond normal specification as the supply voltage drops below 3.3 V. It is justifiable for the analog switches to have higher on-resistance as long as the signal speeds are low. Thus, V_{DDA} can go as low as ~2.8 V before the analog switches become too resistive. It will eventually set the lowest-possible supply voltage. However, it is recommended to use V_{DDA} of 3.3 V or greater when using opamps in Deep Sleep mode. See the *PSoC 4500S datasheet* for opamp specifications during Deep Sleep mode.

To enable the opamps in Deep Sleep mode, set the DEEPSLEEP_ON bit [30] of the CTBMx_CTB_CTRL register. This bit enables both the opamps of the CTBM during deep sleep. The Deep Sleep operation of the CTBm also requires the DSAB block to be enabled.

23.4 Register Summary

Table 23-9. DSAB and CTBM Deep Sleep Configuration Register Settings

Register[Bit_Pos]	Bit_Name	Description
PASS_DSAB_DSAB_CTRL [5:0]	CURRENT_SEL	Current selection for the dsab_ibias; dsab_ibias = CURRENT_SEL × 0.075 µA (±5%)
PASS_DSAB_DSAB_CTRL [8]	SEL_OUT	CTBm bias current selection 0: Bypass DSAB and use active mode bias current 1: Use dsab_ibias as the CTBm bias current
PASS_DSAB_DSAB_CTRL [31]	ENABLED	Enable/disable DSAB bias generator 0: DSAB block is disabled and the CTBm bias current is connected to the Active mode bias current 1: DSAB block is enabled and the CTBm bias current is controlled by the SEL_OUT signal
CTBMx_CTBM_CTB_CTRL [30]	DEEPSLEEP_ON	Enable/disable the CTBMx functionality in Deep Sleep mode 0: Enabled 1: Disabled

Table 23-10. Register Summary

Name	Description
CTBM0_CTB_CTRL	global CTB and power control
CTBM0_OA_RES0_CTRL	Opamp0 and resistor0 control
CTBM0_OA_RES1_CTRL	Opamp1 and resistor1 control
CTBM0_COMP_STAT	Comparator status
CTBM0_INTR	Interrupt request register
CTBM0_INTR_SET	Interrupt request set register
CTBM0_INTR_MASK	Interrupt request mask
CTBM0_INTR_MASKED	Interrupt request masked
CTBM0_OA0_SW	Opamp0 switch control
CTBM0_OA0_SW_CLEAR	Opamp0 switch control clear
CTBM0_OA1_SW	Opamp1 switch control
CTBM0_OA1_SW_CLEAR	Opamp1 switch control clear
CTBM0_OA0_OFFSET_TRIM	Opamp0 trim control
CTBM0_OA0_SLOPE_OFFSET_TRIM	Opamp0 trim control
CTBM0_OA0_COMP_TRIM	Opamp0 trim control
CTBM0_OA1_OFFSET_TRIM	Opamp1 trim control
CTBM0_OA1_SLOPE_OFFSET_TRIM	Opamp1 trim control
CTBM0_OA1_COMP_TRIM	Opamp1 trim control
CTBM1_CTB_CTRL	global CTB and power control
CTBM1_OA_RES0_CTRL	Opamp0 and resistor0 control
CTBM1_OA_RES1_CTRL	Opamp1 and resistor1 control
CTBM1_COMP_STAT	Comparator status
CTBM1_INTR	Interrupt request register
CTBM1_INTR_SET	Interrupt request set register
CTBM1_INTR_MASK	Interrupt request mask
CTBM1_INTR_MASKED	Interrupt request masked
CTBM1_OA0_SW	Opamp0 switch control

Table 23-10. Register Summary *(continued)*

Name	Description
CTBM1_OA0_SW_CLEAR	Opamp0 switch control clear
CTBM1_OA1_SW	Opamp1 switch control
CTBM1_OA1_SW_CLEAR	Opamp1 switch control clear
CTBM1_OA0_OFFSET_TRIM	Opamp0 trim control
CTBM1_OA0_SLOPE_OFFSET_TRIM	Opamp0 trim control
CTBM1_OA0_COMP_TRIM	Opamp0 trim control
CTBM1_OA1_OFFSET_TRIM	Opamp1 trim control
CTBM1_OA1_SLOPE_OFFSET_TRIM	Opamp1 trim control
CTBM1_OA1_COMP_TRIM	Opamp1 trim control

24. Temperature Sensor

The PSoC 4 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.

24.1 Features

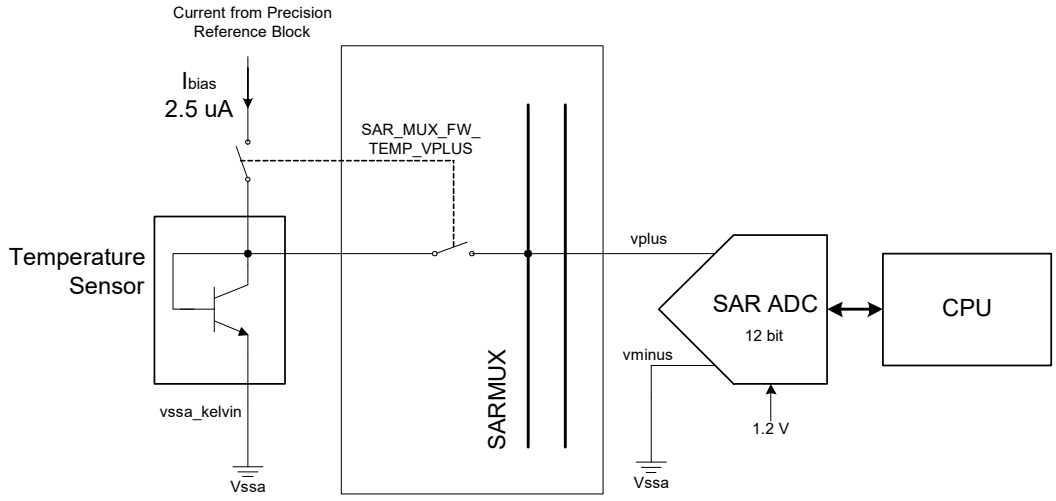
The temperature sensor has the following features:

- ±5°C accuracy over temperature range –40°C to +85°C
- 0.5°C/LSb resolution (without amplification) when using a 12-bit SAR ADC with a 1.2-V reference
- 10-µs settling time

24.2 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 24-1.

Figure 24-1. 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:

$$Temp = (A \times SAR_{out} + 2^{10}xB) + T_{adjust}$$

Equation 24-1

- 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 4 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 24-2**

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:

$$B = (\text{unsigned int})\left(2^{6}x100°C-\left(\frac{A\times SAR_{100C}}{2^{10}}\right)\right)$$ **Equation 24-3**

Where,

SAR_{100C} = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH_SAR_TEMP_OFFSET.

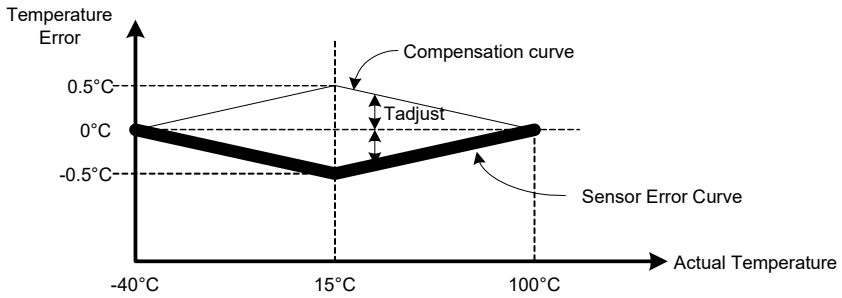
- T_{adjust} is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating $T_{initial} = (A\times SAR_{out} + 2^{10}\times B)$. If it is greater than 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 24-4**

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 24-5**

Figure 24-2. 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.

24.3 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 197 for details on how to read the temperature sensor output using the ADC.

24.4 Algorithm

1. Enable the SARMUX and SAR ADC.
2. Configure the 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 (see Equation 24-1).

   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. Multiplies A and V_{BE}: 0xBC4B × 0x595 = $(–17333)_{10} \times (1429)_{10} = (–24768857)_{10}$

   b. Multiplies B and 1024: 0x65B4 × 0x400 = $(26036)_{10} \times (1024)_{10} = (26660864)_{10}$

   c. Adds the result of steps 1 and 2 to get $T_{initial}$: $(–24768857)_{10} + (26660864)_{10} = (1892007)_{10}$ = 0x1CDEA7

   d. Calculates 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 24-4 to calculate T_{adjust}. It comes to 0x6C6C = $(27756)_{10}$

   e. Adds 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

24.5 Registers

Name	Description
SAR_MUX_SWITCH0	This register has the SAR_MUX_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 the SAR MUX.
SFLASH_SAR_TEMP_MULTIPLIER	Multiplier constant 'A' as defined in Equation 24-1.
SFLASH_SAR_TEMP_OFFSET	Constant 'B' as defined in Equation 24-1.

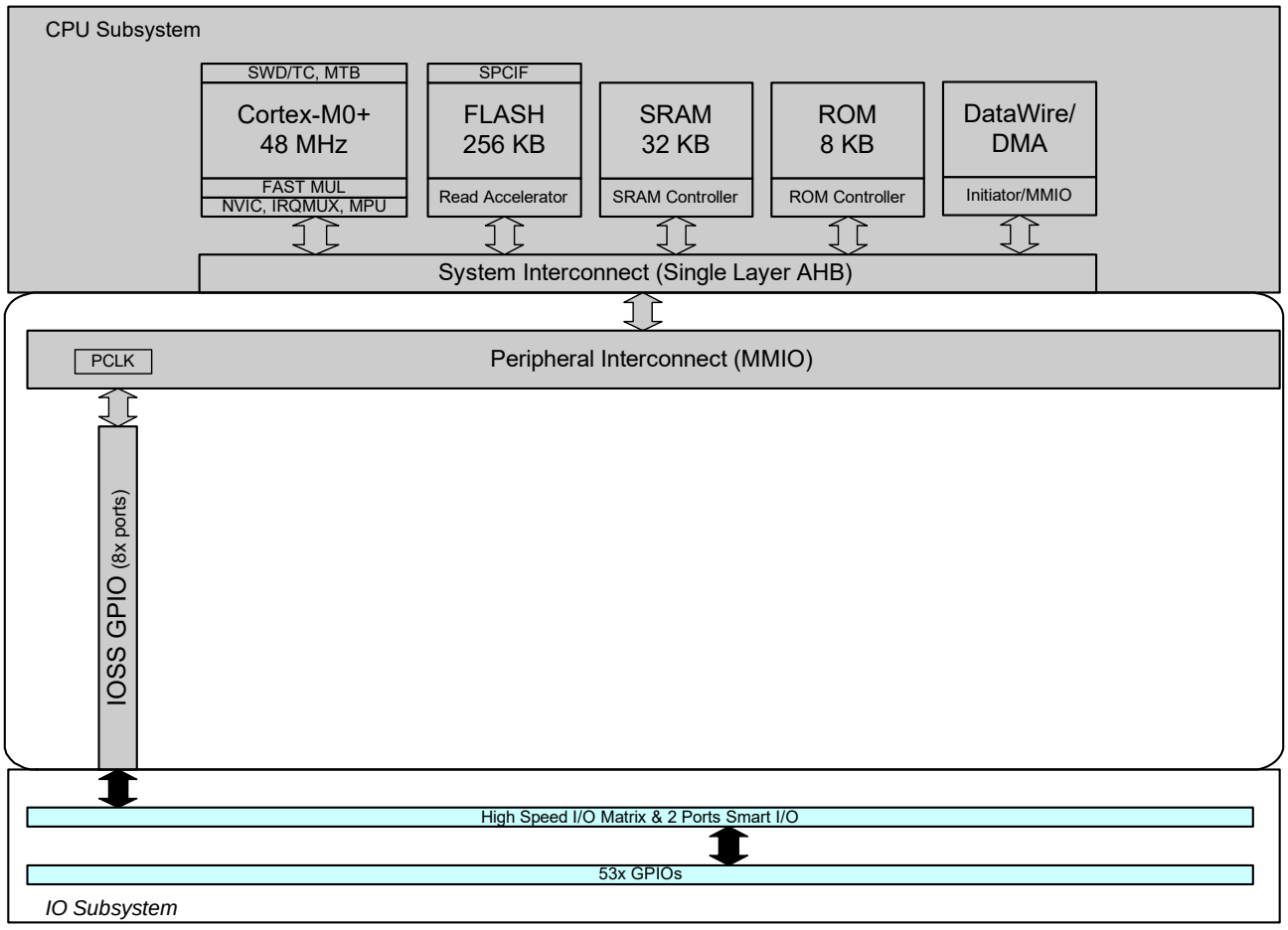
Section F: Program and Debug

This section comprises the following chapters:

- Program and Debug Interface chapter on page 249
- Nonvolatile Memory Programming chapter on page 256

Top-Level Architecture

Program and Debug Block Diagram



25. Program and Debug Interface

The PSoC 4 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. SWD interface is used as the communication protocol between the external device and PSoC 4.

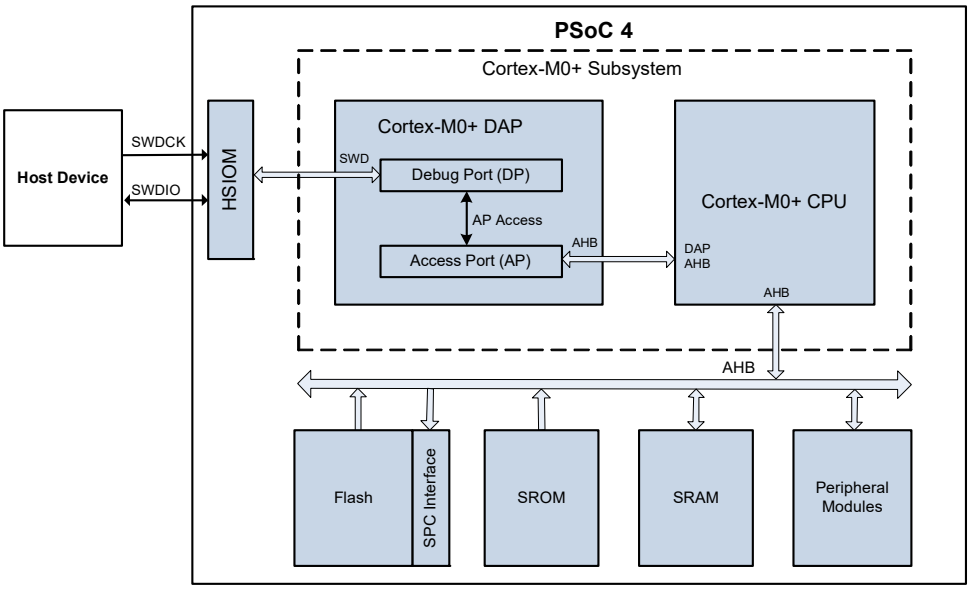
25.1 Features

- 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 CM0+ register bank when the core is running or halted

25.2 Functional Description

Figure 25-1 illustrates the block diagram of the program and debug interface in PSoC 4. CM0+ 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 4 “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 HSIOM. See the I/O System chapter on page 65 for details on HSIOM.

Figure 25-1. Program and Debug Interface



The DAP communicates with the CM0+ 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.

25.3 Serial Wire Debug (SWD) Interface

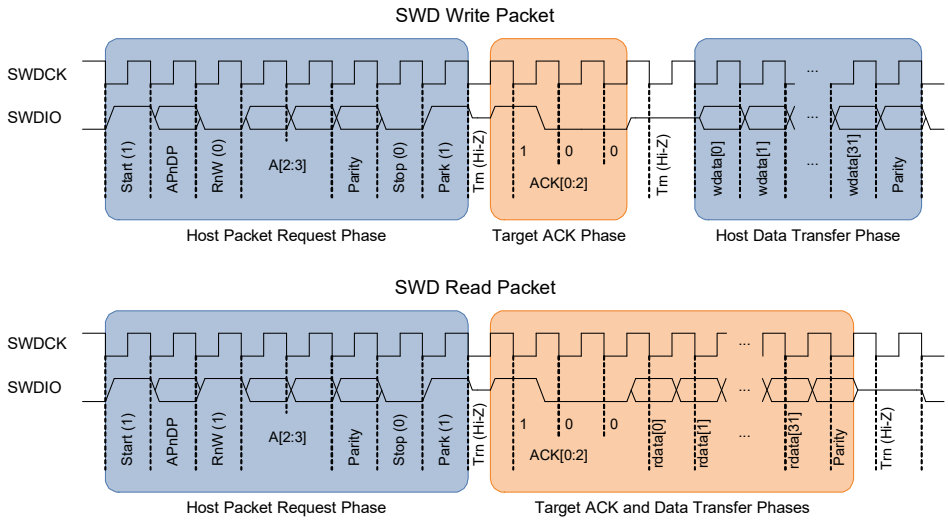
PSoC 4's CM0+ supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses SWDIO and SWDCK. The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete 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 4 target.
- **Target Acknowledge Response Phase** – The PSoC 4 target sends an acknowledgment 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 HIGH-Z state. This period is either one-half or one and a half clock cycles, depending on the transition.

Figure 25-2 shows the timing diagrams of read and write SWD packets.

Figure 25-2. 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 access port or debug port, depending on the APnDP bit value. See Table 25-3 and Table 25-4 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 PSoC 4; 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 25-1 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.

25.3.1 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 25-1 and Figure 25-2 illustrate the timing of SWDIO bit writes and reads.

Table 25-1. SWDIO Bit Write and Read Timing

SWD Packet Phase	SWDIO Edge	
	Falling	Rising
Host Packet Request	Host Write	Target Read
Host Data Transfer		
Target Ack Response	Host Read	Target Write
Target Data Transfer		

25.3.2 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 25-2 shows the ACK bit-field decoding details.

Table 25-2. 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 PSoC 4. However, 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 PSoC 4 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.

25.3.3 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 25-2. 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 PSoC 4 should drive the SWDIO line during the Trn period.

25.4 CM0+ Debug Access Port (DAP)

The CM0+ 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 CM0+ DAP-AHB interface. Typically, the debug port registers are used for a one-time configuration or for error detection purposes, and the access port registers are used to perform programming and debugging operations. Complete architecture details of the DAP is available in the *Arm® Debug Interface v5 Architecture Specification*.

25.4.1 Debug Port Registers

Table 25-3 shows the CM0+ 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 debug port registers. Note that for the same address bits, different debug port 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 debug port registers.

Table 25-3. Main Debug Port Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	Used to force a DAP abort and to clear the error and sticky flag conditions
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	Holds the SWD ID of the CM0+ CPU, which is 0x0BC11477
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	Allows control of the DP and contains status information about the DP
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	Used to select the current AP. In PSoC 4 there is only one AP, which interfaces with the DAP AHB
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	Holds the result of the last AP read operation

25.4.2 Access Port Registers

Table 25-4 lists the main CM0+ access port registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for access port register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 25-4. Main Access Port 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)	Configures and controls accesses through the memory access port to a connected memory system (which is the PSoC 4 Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	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	Holds the 32-bit data read from or to be written to the address specified in the TAR register

25.5 Programming the PSoC 4 Device

The PSoC 4 is programmed using the following sequence. See the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4500S, PSoC 4000S, PSoC 4100S, 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 4.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

25.5.1 SWD Port Acquisition

25.5.1.1 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 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 DAP. 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 DP. 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 CM0+ DAP.

25.5.2 SWD Programming Mode Entry

After the SWD port is acquired, the host must enter 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 DP 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 4500S, PSoC 4000S, PSoC 4100S, 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.

25.5.3 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 256. The exact sequence of calling the programming routines is given in the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4500S, PSoC 4000S, PSoC 4100S, PSoC 4700S Device Programming Specifications.

25.6 PSoC 4 SWD Debug Interface

CM0+ 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 CM0+ Debug.

See the *Armv6-M Architecture Reference Manual* for complete details on the debug architecture.

25.6.1 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 CM0+ 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 DWT block enable, reset vector catch, and hard fault exception catch.

25.6.2 Breakpoint Unit (BPU)

The breakpoint unit provides breakpoint functionality on instruction fetches. The CM0+ DAP in PSoC 4 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 CM0+. 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 4).
- 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.

25.6.3 Debug Watchpoint

The debug watchpoint provides watchpoint support on a data address access or a 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 PC 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.

25.6.4 Debugging the PSoC 4 Device

The host debugs the target PSoC 4 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 CM0+ DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PSoC 4 device 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 65 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 62. 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.

25.7 Registers

Table 25-5. 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

26. Nonvolatile Memory Programming

Nonvolatile memory programming refers to the programming of flash memory in the PSoC 4 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 4 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.

26.1 Features

- Supports programming through the DAP and CM0+ CPU
- Supports both blocking and non-blocking flash program and erase operations from the CM0+ CPU

26.2 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 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 4 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 4 using the SWD protocol by sending the commands to the DAP. The programming sequence for the PSoC 4 device with an external programmer is given in the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4500S, PSoC 4000S, PSoC 4100S, PSoC 4700S Device 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 114) 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 4 implements a User SFlash, which can be used to store application-specific information. These rows are not part of the hex file; their programming is optional.

26.3 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 incorrect, 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.

26.4 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. All other system calls are blocking.

The CPU cannot execute code from flash while doing an erase or program operation on the flash, so 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 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 ISR to ensure that the subsequent steps in the system call are completed. The CPU can execute code only from the SRAM when a non-blocking write or program operation is being done, so the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in 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 provided later in this chapter.

26.4.1 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 initiate 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 NMI 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 the 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 26-1 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.

26.5 System Calls

Table 26-1 lists all the system calls supported in PSoC 4 along with the function description and availability in device protection modes. See the Device Security chapter on page 62 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as listed in Table 26-1. Detailed information on each of the system calls follows the table.

Table 26-1. 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 28 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 SFlash area	✔	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔	–	✔
Write Protection	Programs both flash row-level protection settings and chip-level protection settings into the SFlash (row 0).	✔	✔	–	
Non-Blocking Write Row	Erases and then programs a row of flash with the data in the page latch buffer. During program/erase pulses, you may execute code from SRAM. This function is meant only for CPU access.	–	–	–	✔
Non-Blocking Program Row	Programs a row of flash with the data in the page latch buffer. During program/erase pulses, you 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.	–	–	–	✔

26.5.1 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
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	2800-28FF
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision Id	See the PSoC 4100M, PSoC 4200M, PSoC 4200D, PSoC 4400, PSoC 4500S, PSoC 4000S, PSoC 4100S, PSoC 4700S Device 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 0xB8 for PSOC 4500S
Bits [15:12]	Chip Protection	See the Device Security chapter on page 62
Bits [31:16]	0xXXXX	Not used

26.5.2 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.

26.5.3 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 (see the Cortex-M0+ CPU chapter on page 30 for the number of flash macros in the device)
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

Address	Value to be Written	Description
.	.	.
.	.	.
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)

26.5.4 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.

For more information on the CLK_IMO_CONFIG register, see the *PSoC 4500S: PSoC 4 Registers TRM*.

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		
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)

26.5.5 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)

26.5.6 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in SFlash 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		

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]	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)

26.5.7 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 SFlash 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

26.5.8 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the SFlash 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 SFlash. The size of the SFlash 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	

26.5.9 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 the 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 257.

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 the SRAM. When the non-blocking write row system call is initiated, you 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)

26.5.10 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 257. 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, you 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 the 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)

26.5.11 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. For more details on the procedure of using the non-blocking functions, see Blocking and Non-Blocking System Calls on page 257.

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		
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)

26.6 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 26-2. 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.

Table 26-2. System Call Status Codes *(continued)*

Status Code (32-bit value in CPUSS_SYSARG register)	Description
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.

26.7 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 the 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 52 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.