PSoC 4100/4200 Family

PSoC® 4 Architecture TRM

(Technical Reference Manual)

Document No. 001-85634 Rev. *B

July 18, 2013

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

License

Copyrights

Disclaimer

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress Data Sheets. Cypress believes that its family of PSoC products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents Overview

Contents

Section A: Overview

This section encompasses the following chapters:

Document Revision History

Revision	Issue Date	Origin of Change	Description of Change
**	23 January, 2013	XKJ	Initial release
*A	18 April, 2013	RLIU	Extensive updates throughout the document
*B	18 July, 2013	RLIU	Multiple fixes across the document

1. Introduction

PSoC® 4 is the architecture of programmable embedded system controllers with an ARM® Cortex™-M0 CPU. PSoC 4 delivers a programmable platform for embedded applications. It combines programmable analog, programmable interconnect, user-programmable digital logic, and commonly used fixed-function peripherals with a high-performance ARM Cortex-M0 subsystem.

The PSoC 4100/4200 families are the first members of PSoC 4 architecture. They are upward compatible with larger members of PSoC 4.

PSoC 4 devices have these characteristics:

- High-performance Cortex-M0 CPU core
- Fixed-function and configurable digital blocks
- Programmable digital logic
- High-performance analog system
- Flexible and programmable interconnect

This document describes each function block of PSoC 4 devices in detail. This information will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of the PSoC 4100 architecture. Figure 1-2 shows the architecture of the PSoC 4200 family.

Figure 1-1. PSoC 4100 Family Block Diagram

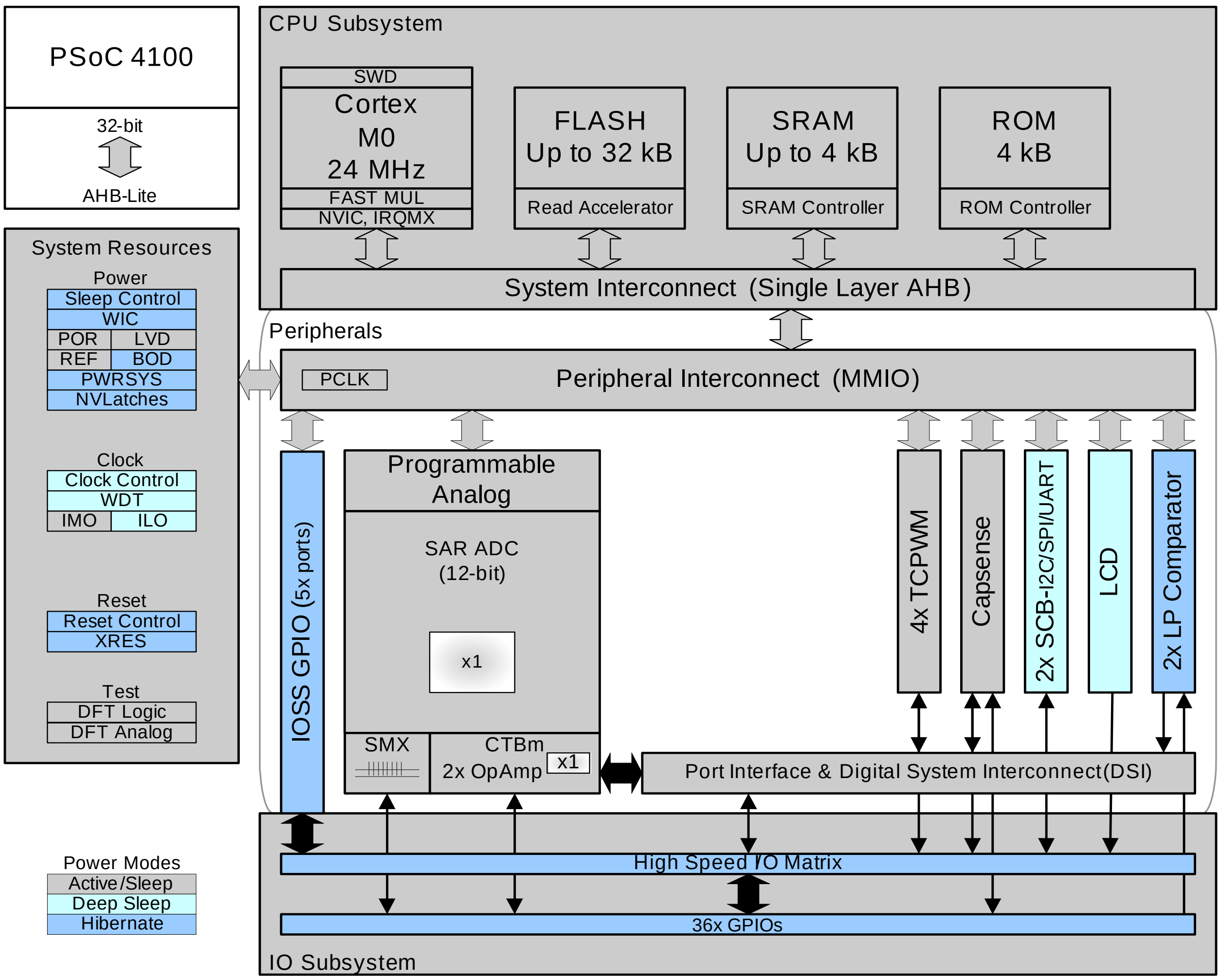
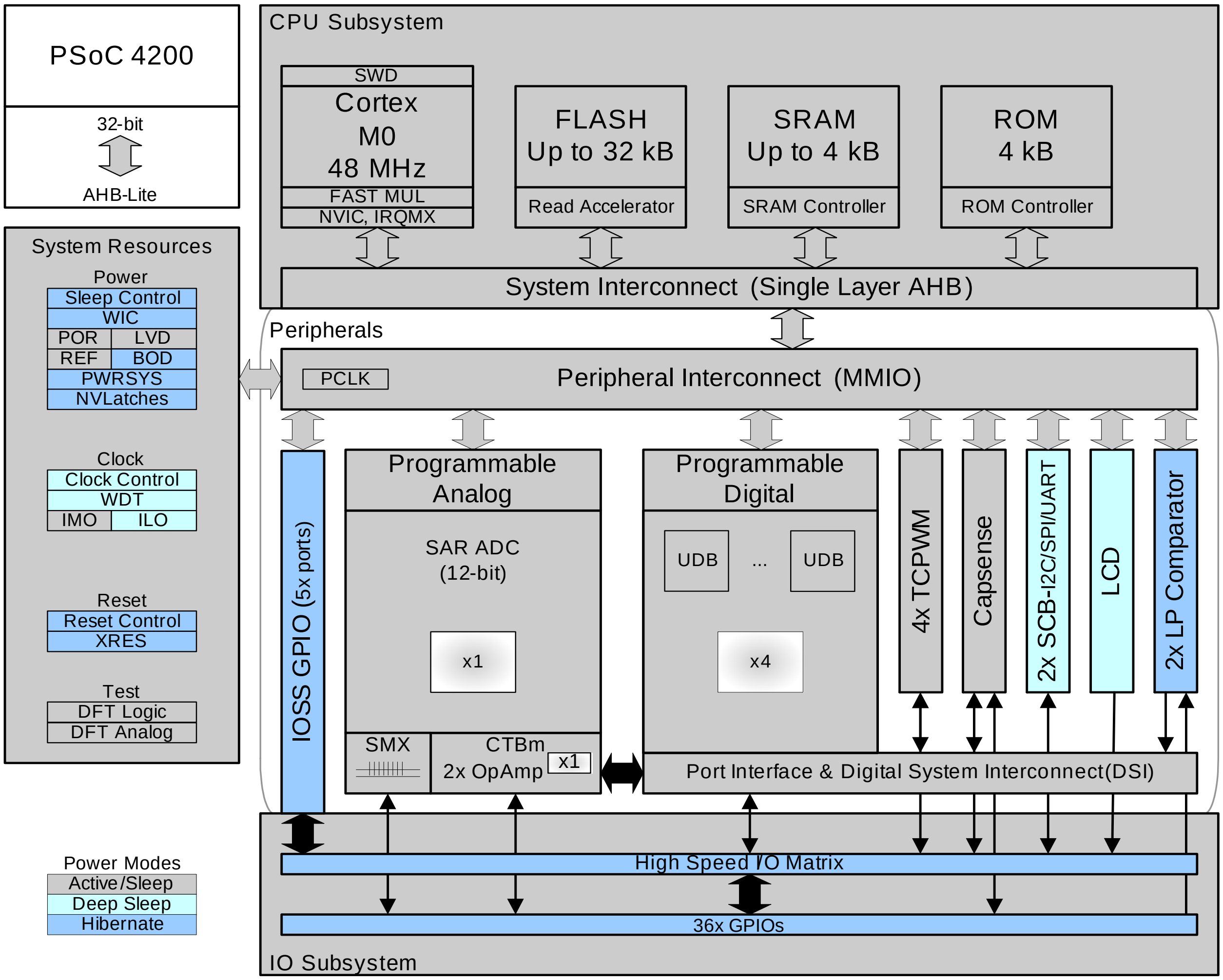


Figure 1-2. PSoC 4200 Family Block Diagram



1.2 Features

PSoC 4100/4200 families have these major components:

- 32-bit Cortex-M0 CPU with single-cycle multiply delivering up to 43 DMIPS at 48 MHz
- Up to 32-KB flash and 4-KB SRAM
- Four independent center-aligned pulse-width modulators (PWMs) with complementary dead-band programmable outputs and synchronized analog-to-digital converter (ADC) operation
- Up to 1 Msps 12-bit ADC including sample-and-hold (S/H) capability with zero-overhead sequencing
- Up to two opamps with comparator mode and successive approximation register (SAR) input buffering capability
- Two low-power comparators
- Two serial communication blocks (SCB) to work as SPI/UART/I2C serial communication channels
- Up to four programmable logic blocks, known as universal digital blocks (UDBs)
- CapSense® and segment LCD drive
- Low-power operating modes: Sleep, Deep-Sleep, Hibernate, and Stop
- Programming and debug system through serial wire debug (SWD)
- Fully supported PSoC Creator™ IDE tool

1.3 CPU System

1.3.1 Processor

The heart of the PSoC 4100/4200 is a 32-bit Cortex-M0 CPU core running up to 48 MHz for PSoC 4200 and 24 MHz for PSoC 4100. 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 enables fully compatible binary upward migration of the code to higher performance processors such as Cortex M3 and M4.

The PSoC 4100/4200 includes a hardware multiplier that provides a 32-bit result in one cycle.

1.3.2 Interrupt Controller

The CPU subsystem of PSoC 4100/4200 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. The Cortex-M0 CPU of PSoC 4100/4200 implements an non-maskable interrupt (NMI) input, which can be tied to digital routing for general-purpose use.

1.4 Memory

The PSoC 4100/4200 memory subsystem consists of flash and SRAM. A supervisory ROM, containing boot and configuration routines, is also provided.

1.4.1 Flash

The PSoC 4100/4200 has a flash module with a flash accelerator tightly coupled to the CPU to improve average access times from the flash block. The flash block is able to deliver one wait-state (WS) access time at 48 MHz and zero WS access time at 24 MHz. The flash accelerator delivers 85 percent of single-cycle SRAM access performance on an average. Part of the flash module can be used to emulate EEPROM operation optionally.

1.4.2 SRAM

The PSoC 4100/4200 provide SRAM, which is retained during hibernate mode.

1.5 System-Wide Resources

1.5.1 Clocking System

The clock system for the PSoC 4100/4200 consists of the internal main oscillator (IMO) and internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock.

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the PSoC 4100/4200. The default IMO frequency is 24 MHz and it can be adjusted between 3 MHz and 48 MHz in steps of 1 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 to generate clocks for peripheral operation in deep-sleep mode. Its clock frequency is 32 kHz with ±60 percent accuracy.

An external clock source ranging from 0 MHz to 48 MHz can be pulled in to generate the clock derivatives for the PSoC 4100/4200 functional blocks instead of the IMO.

1.5.2 Power System

The PSoC 4100/4200 operates with a single external supply over the range of 1.71 V to 5.5 V. PSoC 4100/4200 has several low-power modes – sleep, deep-sleep, hibernate, and stop modes – besides the default active mode.

In active mode, CPU runs with all the logic powered. In sleep mode, the CPU does not function with its work clock off. In deep-sleep mode, the CPU, SRAM, and high-speed logic are in retention; the main system clock is off while the low-frequency clock is on and the low-frequency peripherals are in operation. In hibernate mode, even the low-frequency clock is off and low-frequency peripherals stop operating.

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

1.5.3 GPIO

Every GPIO in PSoC 4100/4200 has the following characteristics:

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

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

1.6 Programmable Digital

The PSoC 4200 has up to four UDBs. Each UDB contains structured data-path logic and uncommitted PLD logic with flexible interconnect. The UDB array provides a switched routing fabric called the Digital System Interconnect (DSI). The DSI allows routing of signals from peripherals and ports to and within the UDBs.

The UDB arrays in PSoC 4200 enable custom logic or additional timers/PWMs and communication interfaces such as I2C, SPI, I2S, and UART.

Note PSoC 4100 does not support UDBs.

1.7 Fixed-Function Digital

1.7.1 Timer/Counter/PWM Block

The Timer/Counter/PWM block consists of four 16-bit counters with user-programmable period length. The functionality of these counters can be synchronized. Each block has a capture register, period register, and compare registers. 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 pre-scaling, pseudo random PWM, and quadrature decoding.

1.7.2 Serial Communication Blocks

The PSoC 4100/4200 has two SCBs, which can each implement a serial communication interface as I2C, universal asynchronous receiver/transmitter (UART), or serial peripheral interface (SPI).

The features for each SCB include:

- Standard I2C multi-master and slave function.
- Standard SPI master and slave function with Motorola, TI, and National (MicroWire) mode.
- Standard UART transmitter and receiver function with SmartCard reader (ISO7816), IrDA protocol, and LIN.
- EZ function mode support for SPI and I2C with 32-byte buffer.

1.8 Analog System

1.8.1 SAR ADC

PSoC 4200 has a configurable 12-bit 1-Msps SAR ADC and PSoC 4100 has a similar 12-bit SAR ADC with 806 ksps. With a gain error of ±0.1 percent, integral nonlinearity (INL) less than 1 LSB, differential nonlinearity (DNL) less than 1 LSB, and signal-to-noise ratio (SNR) better than 68 dB, this convertor addresses a wide variety of analog applications.

The ADC provides the choice of three internal voltage references (V_{DD}, $V_{DD}/2$, and V_{REF}) and an external reference through a GPIO pin. The SAR is connected to a fixed set of pins through an 8-input sequencer. The sequencer can buffer each channel to reduce CPU interrupt service requirements.

1.8.2 Continuous Time Block mini (CTBm)

The CTBm block 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 used.

1.8.3 Low-power Comparators

The PSoC 4100/4200 has a pair of low-power comparators, which operate in deep-sleep and hibernate modes. This allows the analog system blocks to be disabled while retaining the ability to monitor external voltage levels during low-power modes.

Two input voltages can both come from pins, or one from an internal signal through the AMUXBUS.

1.9 Special Function Peripherals

1.9.1 LCD Segment Drive

The PSoC 4100/4200 has an LCD controller, which can drive up to four commons and every GPIO can be configured to drive common or segment. It uses full digital methods (digital correlation and PWM) to drive the LCD segments, and does not require generation of internal LCD voltages.

1.9.2 CapSense

PSoC 4100/4200 devices has the CapSense feature, which allows you to use the capacitive properties of your fingers to toggle buttons, sliders, and wheels. CapSense functionality is supported on all GPIO pins in PSoC 4100/4200 through a CapSense Sigma-Delta (CSD) block. The CSD also provides waterproofing capability. The CapSense block has two

IDACs, which can be used for general purposes if CapSense is not used.

1.10 Program and Debug

PSoC 4100/4200 devices support programming and debug features of the device via the on-chip SWD interface. The PSoC Creator IDE software provides fully integrated programming and debug support for PSoC 4100/4200 devices. The SWD interface is also fully compatible with industry standard third-party tools.

2. Getting Started

2.1 Support

Free support for PSoC® 4 products is available online at http://www.cypress.com. Resources include Training Seminars, Discussion Forums, Application Notes, PSoC Consultants, CRM Technical Support Email, Knowledge Base, and Application Support Technicians.

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

2.2 Product Upgrades

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

2.3 Development Kits

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

3. Document Construction

The following sections in this document include these topics:

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 about any particular area that inform the reader about the construction and organization of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These are the detailed implementation and use information for some aspect of the integrated circuit.
- Glossary – Defines the specialized terminology used in this technical reference manual. Glossary terms are presented in bold, italic font throughout.
- PSoC® 4 Registers Technical Reference Manual – Supply all device register details summarized in the technical reference manual. These are additional documents.

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® 4 Registers Technical Reference Manual.

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

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

3.2.4 Acronyms

This table lists the acronyms that are used in this document

Table 3-2. Acronyms

Symbol	Unit of Measure
ABUS	analog output bus
AC	alternating current
ADC	analog-to-digital converter
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an ARM data transfer bus
API	application programming interface
APOR	analog power-on reset
BC	broadcast clock

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
BOM	bill of materials
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CAN	controller area network
CI	carry in
CMP	compare
CO	carry out
CPU	central processing unit
CRC	cyclic redundancy check
CT	continuous time
DAC	digital-to-analog converter
DC	direct current
DI	digital or data input
DNL	differential nonlinearity
DO	digital or data output
DSI	digital signal interface
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
EMIF	external memory interface
FB	feedback
FIFO	first in first out
FSR	full scale range
GPIO	general purpose I/O
I^2C	inter-integrated circuit
IDE	integrated development environment
ILO	internal low-speed oscillator
IMO	internal main oscillator
INL	integral nonlinearity
I/O	input/output
IOR	I/O read
IOW	I/O write
IRES	initial power on reset
IRA	interrupt request acknowledge
IRQ	interrupt request
ISR	interrupt service routine
IVR	interrupt vector read
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
MISO	master-in-slave-out
MMIO	memory mapped input/output
MOSI	master-out-slave-in
MSb	most significant bit
MSB	most significant byte

Table 3-2. Acronyms (continued)

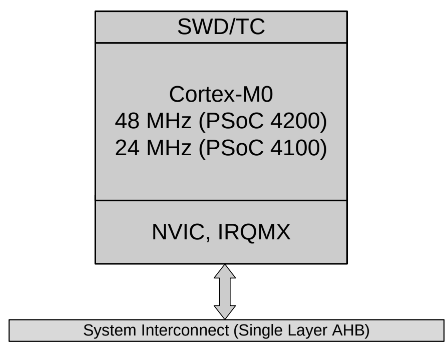
Symbol	Unit of Measure
PC	program counter
PCH	program counter high
PCL	program counter low
PD	power down
PGA	programmable gain amplifier
PICU	port interrupt control unit
PM	power management
PMA	PSoC memory arbiter
POR	power-on reset
PPOR	precision power-on reset
PRS	pseudo random sequence
PSoC®	Programmable System-on-Chip
PSRR	power supply rejection ratio
PSSDC	power system sleep duty cycle
PWM	pulse-width modulator
RAM	random-access memory
RETI	return from interrupt
ROM	read only memory
RW	read/write
SAR	successive approximation register
SC	switched capacitor
SCB	serial communication block
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SWD	single wire debug
TC	terminal count
TD	transaction descriptors
UART	universal asynchronous receiver/transmitter
UDB	universal digital block
USB	universal serial bus
USBIO	USB I/O
WDT	watchdog timer
WDR	watchdog reset
XRES_N	external reset, active low

Section B: CPU System

This section encompasses the following chapters:

Top Level Architecture

CPU System Block Diagram



4. Cortex-M0 CPU

The PSoC® 4 ARM Cortex-M0 core is a 32-bit CPU optimized for low-power operation. It has an efficient three-stage pipeline, a fixed 4-GB memory map, and supports the ARMv6-M Thumb instruction set. The Cortex-M0 also features a single-cycle multiply instruction and low-latency interrupt service routine (ISR) entry and exit.

The Cortex-M0 processor includes a number of other components that are tightly linked to the CPU core. These include a Nested Vectored Interrupt Controller (NVIC), a SYSTICK timer, and debug.

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

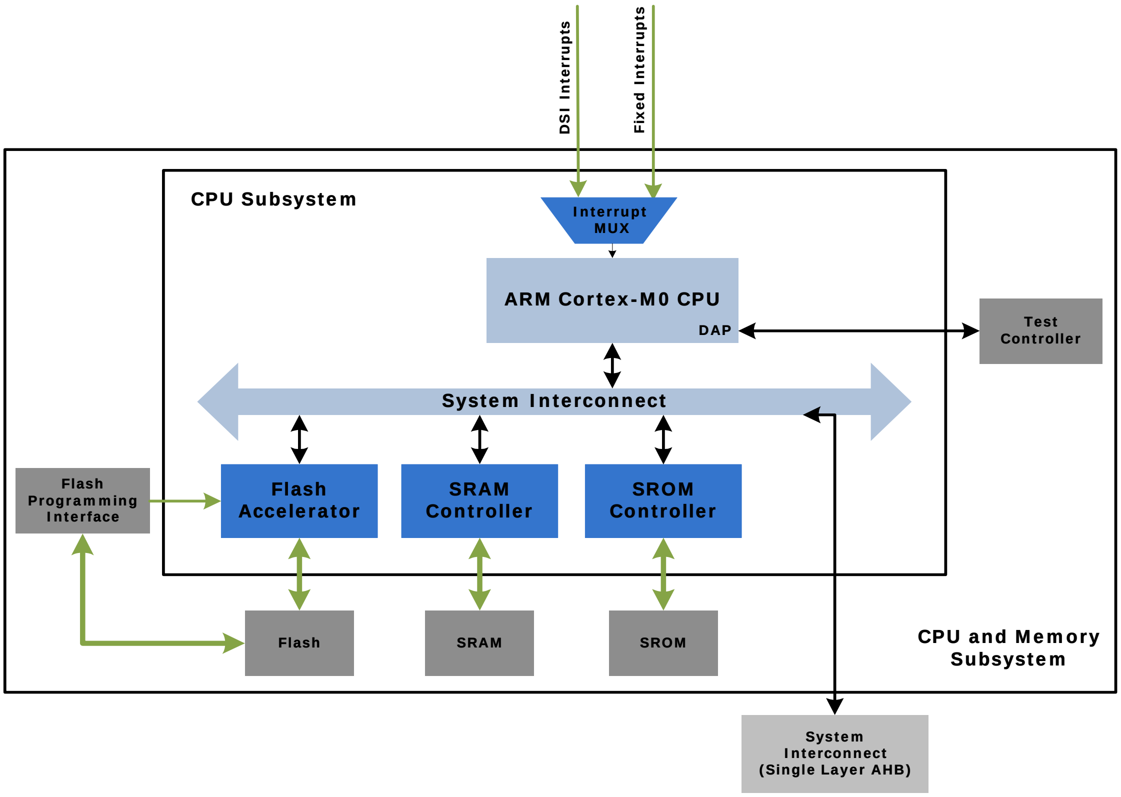
4.1 Features

The PSoC 4 Cortex-M0 has the following features:

- Easy to use, program and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Supports Thumb instruction set, for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions, for rapid and deterministic interrupt response
- Extensive debug support including:
  - Serial wire debug (SWD) port
  - Break points
  - Watch points

4.2 Block Diagram

Figure 4-1. PSoC 4 CPU Subsystem Block Diagram



4.3 How It Works

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

The processor supports two operating modes, and has a single cycle 32-bit multiplication instruction.

4.3.1 Registers

The Cortex-M0 has 16 32-bit registers, as Table 4-1 shows:

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

Table 4-1. Cortex-M0 Registers

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

a. Describes access type during program execution in thread mode and handler mode. Debug access can differ.
b. Bit[24] is the T-bit and is loaded from bit[0] of the reset vector.

Table 4-2 shows how the PSR bits are assigned.

Table 4-2. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSE	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 … 47 = IRQ31

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 37 for a list of exceptions.

4.3.2 Operating Modes

The Cortex-M0 processor supports two operating modes:

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

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

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

4.3.3 Instruction Set

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

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. Many instructions are unable to use, or have restrictions on whether you can use, the PC or SP for the operands or destination register.

Table 4-3. Thumb Instruction Set

Mnemonic	Brief Description
ADCS	Add with Carry
ADD{S}	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}	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result
MVNS	Bit wise NOT
NOP	No Operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-Reverse word
REV16	Byte-Reverse packed half-words
REVSH	Byte-Reverse signed half-word
RORS	Rotate Right
RSBS	Reverse Subtract
SBCS	Subtract with Carry
SEV	Send Event
STM	Store Multiple registers, increment after
STR	Store Register as word
STRB	Store Register as byte
STRH	Store Register as half-word
SUB{S}	Subtract
SVC	Supervisor Call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND based test

Table 4-3. Thumb Instruction Set

Mnemonic	Brief Description
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait For Event
WFI	Wait For Interrupt

4.3.3.1 Address Alignment

An aligned access is an operation where a word-aligned address is used for a word or multiple word access, or where a half word-aligned address is used for a half word access. Byte accesses are always aligned.

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

4.3.3.2 Memory Endianness

The PSoC 4 Cortex-M0 uses little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

4.3.4 Systick Timer

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

4.3.5 Debug

PSoC 4 contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

5. Interrupts

The ARM Cortex-M0 (CM0) CPU in PSoC® 4 supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, analog-to-digital converters, 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 handler (or interrupt service routine) corresponding to the interrupt/exception being executed by the CPU. PSoC 4 provides a unified exception vector table for both interrupt handlers and exception handlers.

5.1 Features

PSoC 4 supports the following interrupt features:

- Supports 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
- Two sources for each interrupt: fixed-function or a flexible on-chip digital signal
- Level-triggered and pulse-triggered interrupt signals

5.2 How It Works

Figure 5-1. PSoC 4 Interrupts Block Diagram

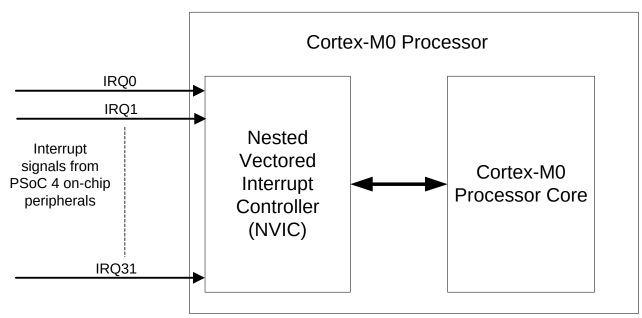


Figure 5-1 shows the interaction between interrupt signals and the Cortex-M0 CPU. PSoC 4 has 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in Figure 5-1 because they are part of CM0 core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

5.3 Interrupts and Exceptions - Operation

5.3.1 Interrupt/Exception Handling in PSoC 4

This section explains the sequence of events that happen when an interrupt or exception event is triggered. Assuming that all the interrupt signals are initially low (idle or inactive state) and the processor is executing the main code, a rising edge on any one of the interrupt lines is registered by the NVIC. The interrupt line is now in a pending state waiting to be serviced by the

CPU. 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. The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions in PSoC 4 have a unique exception number, as given in Table 5-1. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table. The CPU then branches to this address and executes the exception handler that follows. Upon completion of the exception handler, the CPU registers are restored to their original state using stack pop operations, and the CPU resumes the main code execution.

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

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

5.3.2 Level and Pulse Interrupts

PSoC 4 NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IEQ31). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 5-2. Level Interrupts

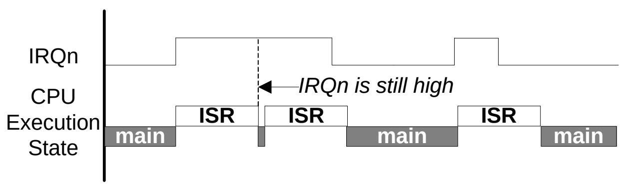


Figure 5-3. Pulse Interrupts

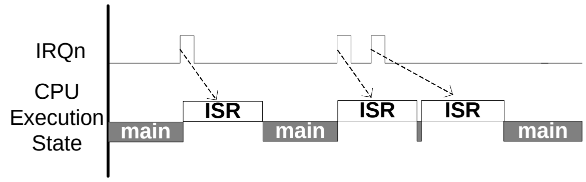


Figure 5-2 and Figure 5-3 show the working level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

- 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.
- The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
- 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 5-3 for pulse interrupts).
- If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. Figure 5-2 illustrates this for level triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

5.3.3 Exception Vector Table

The exception vector table (Table 5-1), stores the entry point addresses for all exception handlers in PSoC 4. The CPU fetches the appropriate address based on exception number.

Table 5-1. PSoC 4 Exception Vector Table

Exception Number	Exception	Exception Priority	Vector Address
	Initial Stack Pointer Value	Not Applicable (NA)	Base_Address - Can be 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 - Base_Address + 0x28
11	Supervisory Call (SVCall)	Configurable (0 - 3)	Base_Address + 0x2C
12-13	Reserved	NA	Base_Address + 0x30 - 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 5-1, the first word (4-bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. In PSoC 4, the vector table can be configured to be located either in flash memory (starting from address 0x00000000) or SRAM (address of 0x20000000). This configuration is done by writing to the VECS_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. When VECS_IN_RAM bit field is ‘1’, CPU fetches for exception handler addresses are done from the SRAM vector table location. When this bit field is ‘0’ (reset state), the vector table in flash memory is used for exception address fetches. You must set the VECS_IN_RAM bit field as part of the device boot code to configure the vector table to be in SRAM. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

The exception sources (exception numbers 1 to 15) are explained in 5.4 Exception Sources. The exceptions marked as Reserved in Table 5-1 are not used in PSoC 4, though they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 47) are explained in 5.5 Interrupt Sources.

5.4 Exception Sources

This section explains the different exception sources listed in Table 5-1 (exception numbers 1 to 15).

5.4.1 Reset Exception

Device reset is treated as an exception in PSoC 4. It is permanently enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, and watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of supervisory read-only memory (SROM). The SROM has the vector address of the reset exception, which is an address in the SROM itself. This address is the starting address of the initial boot up code executed out of SROM. The boot code and other data in SROM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in Table 5-1) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in 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 deasserted.

5.4.2 Non-Maskable Interrupt (NMI) Exception

Non-maskable interrupt (NMI) is the highest priority exception other than reset. It is permanently enabled with a fixed priority of –2. There are three ways to trigger an NMI exception in PSoC 4:

- **NMI exception due to a hardware signal (user NMI exception):** PSoC 4 provides a provision to trigger NMI exception using a digital signal. This digital signal is referred to as irq_out[0] in Table 5-2. The flexible digital signal interconnect structure in PSoC 4 ensures that the irq_out[0] line can be driven by any of the digital outputs of on-chip peripherals, or external port pin signals. The NMI exception triggered due to irq_out[0] will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **NMI exception by setting NMIPENDSET bit (user NMI exception):** NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0_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 in PSoC 4 such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREG register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM – the flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

5.4.3 HardFault Exception

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

5.4.4 Supervisor Call (SVCall) Exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system. This is called a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0 in PSoC 4 uses a proprietary privileged mode for the system call NMI exception, which is not related to the SVCall exception. There is no other privileged mode support for SVCall at the architecture level in PSoC 4. 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 bits [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.

5.4.5 PendSV Exception

PendSV is another supervisor call related exception similar to SVCall. PendSV is permanently enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, CM0_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, CM0_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two bits [23:22] of the System Handler Priority Register 3 (SHPR3).

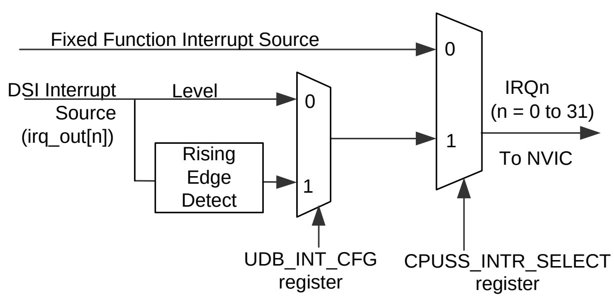
5.4.6 SysTick Exception

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 time keeping purposes such as 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 (CM0_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bits [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 PENDSTSET bit in the Interrupt Control State Register, CM0_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, CM0_ICSR.

5.5 Interrupt Sources

PSoC 4 supports 32 interrupts (IRQ0 - IRQ31 or exception numbers 16 - 47) from peripherals. The source of each interrupt is listed in Table 5-2. PSoC 4 provides flexible sourcing options for each of the 32 interrupt lines. Figure 5-4 shows the multiplexing options for interrupt source. Each interrupt has two sources: a fixed-function interrupt source and a DSI interrupt source. The CPUSS_INTR_SELECT register is used to select between these sources.

Figure 5-4. Interrupt Source Multiplexing



Note The DSI interrupt signal naming (irq_out[n]) is not readily accessible, but the PSoC Creator IDE simplifies the task by doing the routing of the digital signals through the DSI interrupt path. You do not need to manually configure the DSI path.

The fixed-function interrupts include standard interrupts from the on-chip peripherals such as PWMs, serial communication blocks, ADC, and power manager. The fixed-function 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. Fixed-function 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.

The second category of interrupt sources is the DSI interrupt signals. Any digital signal on the chip, such as digital outputs from UDBs or digital input signals on pins, can be routed as DSI interrupt sources. This provides flexibility in the choice of interrupt sources. You also have the option of routing the DSI signal through a rising edge detect circuit, as shown in Figure 5-4. This edge detect circuit converts a rising edge signal on the DSI line to a pulse signal two system clocks wide. This ensures that the interrupt is triggered once on the rising edge of the signal on the DSI line. It is useful for interrupt sources, which cannot generate proper level interrupt signals to the NVIC. The UDB_INT_CFG register is used to select between the direct DSI path and the edge detect path.

Table 5-2. List of PSoC 4 Interrupt Sources

Interrupt No.	Cortex-M0 Exception No.	Fixed Function	DSI Interrupt Source
NMI (see 5.4 Exception Sources)	2		irq_out[0]
IRQ0	16	GPIO P0 (Port Interrupt)	irq_out[1]
IRQ1	17	GPIO P1 (Port Interrupt)	irq_out[1]
IRQ2	18	GPIO P2 (Port Interrupt)	irq_out[2]
IRQ3	19	GPIO P3 (Port Interrupt)	irq_out[3]
IRQ4	20	GPIO P4 (Port Interrupt)	irq_out[4]
IRQ5	21	<DSI-only>	irq_out[5]
IRQ6	22	<DSI-only>	irq_out[6]
IRQ7	23	<DSI-only>	irq_out[7]
IRQ8	24	LPCOMP (low-power comparator)	irq_out[8]
IRQ9	25	WDT (Watchdog timer)	irq_out[9]
IRQ10	26	SCB1 (Serial Communication Block 1)	irq_out[10]
IRQ11	27	SCB2 (Serial Communication Block 2)	irq_out[11]
IRQ12	28	SPC (System Performance Controller)	irq_out[12]
IRQ13	29	PWR (Power Manager)	irq_out[13]
IRQ14	30	SAR (Successive Approximation ADC)	irq_out[14]
IRQ15	31	CSD (Capsense block counter overflow interrupt)	irq_out[15]
IRQ16	32	TCPWM0 (Timer/Counter/PWM 0)	irq_out[16]
IRQ17	33	TCPWM1 (Timer/Counter/PWM 1)	irq_out[17]
IRQ18	34	TCPWM2 (Timer/Counter/PWM 2)	irq_out[18]
IRQ19	35	TCPWM3 (Timer/Counter/PWM 3)	irq_out[19]
IRQ20	36	<DSI-only>	irq_out[20]
IRQ21	37	<DSI-only>	irq_out[21]
IRQ22	38	<DSI-only>	irq_out[22]
IRQ23	39	<DSI-only>	irq_out[23]
IRQ24	40	<DSI-only>	irq_out[24]
IRQ25	41	<DSI-only>	irq_out[25]
IRQ26	42	<DSI-only>	irq_out[26]
IRQ27	43	<DSI-only>	irq_out[27]
IRQ28	44	<DSI-only>	irq_out[28]
IRQ29	45	<DSI-only>	irq_out[29]
IRQ30	46	<DSI-only>	irq_out[30]
IRQ31	47	<DSI-only>	irq_out[31]

5.6 Enabling/Disabling Exceptions

The 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 (CM0_ISER) and the Interrupt Clear-Enable Register (CM0_ICER) are used to enable and disable the interrupts respectively. These registers are 32-bit wide 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 5-3 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 5-3. Interrupt Enable/Disable Registers

Register	Operation	Bit Value	Comment
Interrupt Set Enable Register (CM0_ISER)	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
Interrupt Clear Enable Register (CM0_ICER)	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The CM0_ISER and CM0_ICER registers are applicable only for the interrupts (IRQ0 - IRQ31). These registers cannot be used to enable or disable the exception numbers 1 - 15. The 15 exceptions have their own support for enabling and disabling, as explained in Exception Sources on page 39.

The PRIMASK register in Cortex-M0 (CM0) CPU can be used as a global exception enable register to mask all the configurable priority exceptions irrespective of whether they are enabled. Configurable priority exceptions include all the exceptions except the Reset, NMI, and HardFault exceptions listed in Table 5-1. 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 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.

5.7 Exception States

Each exception can be in one of the following states.

Table 5-4. Exception States

Exception State	Meaning
Inactive	The exception is not active and not pending. Either the exception is disabled, or the enabled exception has not been triggered.
Pending	The exception request has been 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 being serviced by the processor and there is a pending request from the same source during its exception handler execution.

The Interrupt Control State Register (CM0_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0_ICSR register store the exception number for the current executing exception. This value is zero if the CPU is not executing any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0_ICSR register store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0_ICSR register indicates if a NVIC generated interrupt (IRQ0 - IRQ-31) is in a pending state.

5.7.1 Pending Exceptions

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

The NVIC allows software pending of the 32 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending Register (CM0_ISPR) and the Interrupt Clear-Pending Register (CM0_ICPR) are used to set and clear the pending status of the interrupt lines. These registers are 32-bits wide, and each bit corresponds to the same numbered interrupt. Table 5-5 shows the register access properties for these two registers. Note that writing zero to these registers is not a valid action.

Table 5-5. Interrupt Set Pending/Clear Pending Registers

Register	Operation	Bit Value	Comment
Interrupt Set-Pending Register (CM0_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 (CM0_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 interrupt handler. The pending bit can be updated regardless of whether the corresponding interrupt is enabled or not. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0_ISER register.

Note that the CM0_ISPR and CM0_ICPR registers used only for the 32 peripheral interrupts (exception numbers 16-47). These registers cannot be used for pending the exception numbers 1 -15. These 15 exceptions have their own support for pending, as explained in Exception Sources on page 39.

5.7.2 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 except 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, meaning 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 preempt (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as active exception. The CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0 CPU in PSoC 4 allows nesting of up to four exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1-15 are explained in Exception Sources on page 39.

The priority of the 32 interrupts (IRQ0 - IRQ31) can be configured by writing to the Interrupt Priority registers (CM0_IPR). This is a group of eight 32-bit registers with each register storing the priority values of four interrupts, as given in Table 5-6. The other bit fields in the register are not used.

Table 5-6. 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.

5.8 Stack Usage for Exceptions

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

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

The Cortex-M0 uses two techniques: tail chaining and late arrival, to reduce latency in servicing exceptions. These techniques are not visible to the external user and are done as part of the internal processor architecture (http://infocenter.arm.com/help/topic/com.arm.doc.ddi0419c/index.html).

5.9 Interrupts and Low-Power Modes

PSoC 4 allows device wakeup from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the device to enter active mode when one or more wakeup sources generate an interrupt signal. After entering active mode, the interrupt handler of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0 CPU, triggers the transition into sleep, deep-sleep, and hibernate modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 75. Chip low-power modes have three categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available in the active, deep-sleep, and hibernate modes (port interrupts, low-power comparators).
- Fixed-function interrupt sources that are available only in the active and deep-sleep modes (watchdog timer interrupt, serial communication block interrupts)
- Fixed-function interrupt sources that are available only in the active mode (all other fixed-function interrupts)

DSI interrupt sources (irq_out[n] in Figure 5-4) do not have the capability to wake up the device from low-power modes. If a DSI interrupt source is selected for an interrupt line, then the fixed-function source corresponding to that line also loses the ability to wake up the device.

5.10 Exception - Initialization and Configuration

This section covers the different steps involved in initializing and configuring exceptions in 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 either a ‘1’ (SRAM vector table) or ‘0’ (flash vector table) to the VECS_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. This register write is done as part of device initialization code.

   It is recommended to locate the exception vector table in SRAM so that dynamic change of exception handler address is possible. For example, an application may require the exception handler to be changed dynamically. By placing the exception vector table in SRAM, you can modify the appropriate location in the SRAM vector table dynamically. If the flash memory vector table is used, then a flash write operation is required to modify the vector table contents.

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 and configuring the interrupt source, as shown in Figure 5-4. 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 5-1 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 42.
   d. Enable the exception, as explained in Enabling/Disabling Exceptions on page 41.

5.11 Registers

Register Acronym	Register Name
CM0_ISER	Interrupt Set-Enable Register
CM0_ICER	Interrupt Clear Enable Register
CM0_ISPR	Interrupt Set-Pending Register
CM0_ICPR	Interrupt Clear-Pending Register
CM0_IPR	Interrupt Priority Registers
CM0_ICSR	Interrupt Control State Register
CM0_AIRCR	Application Interrupt and Reset Control Register
CM0_SCR	System Control Register
CM0_CCR	Configuration and Control Register
CM0_SHPR2	System Handler Priority Register 2
CM0_SHPR3	System Handler Priority Register 3
CM0_SHCSR	System Handler Control and State Register
CM0_SYST_CSR	Systick Control and Status
CPUSS_CONFIG	CPU Subsystem Configuration
CPUSS_SYSREQ	System Request Register
CPUSS_INTR_SELECT	Interrupt Multiplexer Select Register
UDB_INT_CFG	UDB Subsystem Interrupt Configuration

5.12 Associated Documents

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

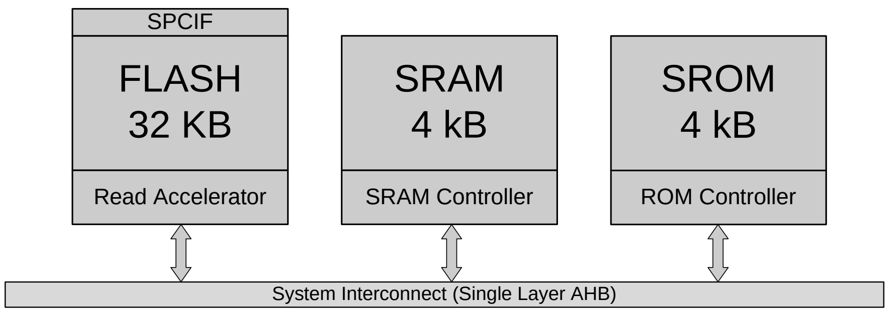
Section C: Memory System

This section presents the following chapter:

- Memory Map chapter on page 47

Top Level Architecture

Memory System Block Diagram



6. Memory Map

All PSoC® 4 memory (flash, SRAM, and supervisory ROM (SROM)) and all registers are accessible by the CPU and in most cases by the debug system. This chapter contains an overall map of the addresses of the memories and registers.

6.1 Features

The PSoC 4 memory system has the following features:

- 32K bytes flash, 4K bytes SRAM
- 4K byte SROM contains boot and configuration routines
- ARM Cortex-M0 32-bit linear address space, with regions for code, SRAM, peripherals, and CPU internal registers
- Flash is mapped to the Cortex-M0 code region
- SRAM is mapped to the Cortex-M0 SRAM region
- Peripheral registers are mapped to the Cortex-M0 peripheral region
- The Cortex-M0 Private Peripheral Bus (PPB) region includes registers implemented in the CPU core. These include registers for NVIC, SysTick timer, and serial communication block (SCB). For more information, see Cortex-M0 CPU chapter on page 31.

6.2 How It Works

The PSoC 4 memory map is detailed in the following tables. For additional information, refer to the PSoC 4 Registers Technical Reference Manual (TRM).

The ARM Cortex-M0 has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in Table 6-1. Note that code can be executed from the code and SRAM regions.

Table 6-1. Cortex-M0 Address Map

Address Range	Name	Use
0x00000000 – 0x1FFFFFFF	Code	Executable region for program code. You can also put data here. Includes the exception vector table which starts at address 0
0x20000000 – 0x3FFFFFFF	SRAM	Executable region for data. You can also put code here
0x40000000 – 0x5FFFFFFF	Peripheral	All peripheral registers. Code cannot be executed out of this region
0x60000000 – 0xDFFFFFFF		Not used
0xE0000000 – 0xE00FFFFF	PPB	Peripheral registers within the CPU core
0xE0100000 – 0xFFFFFFFF	Device	PSoC 4 implementation-specific

Table 6-2 shows the PSoC 4 address map.

Table 6-2. PSoC 4 Address Map

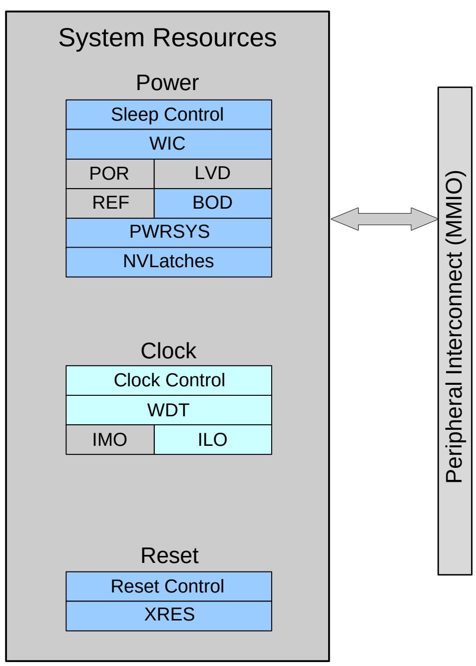
Address Range	Use
0x00000000 – 0x00007FFF	32 KB flash
0x10000000 – 0x10000FFF	4 KB supervisory flash
0x20000000 – 0x20000FFF	4 KB SRAM
0x40000000 – 0x4000FFFF	CPU subsystem registers
0x40010000 – 0x40010FFF	I/O port control (high-speed I/O matrix) registers
0x40020000 – 0x4002FFFF	Programmable clocks registers
0x40040000- -0x4004FFFF	I/O port registers
0x40050000- -0x40050FFF	Timer/counter/PWM (TCPWM) registers
0x40060000- -0x4006FFFF	Serial Communications Block (SCB) registers
0x40080000- -0x4008FFFF	CapSense registers
0x40090000- -0x4009FFFF	LCD registers
0x400A0000- -0x400AFFFF	Low-power comparator registers
0x400B0000- -0x400BFFFF	Power, clock, reset control registers
0x400F0000- -0x400FFFFF	UDB control registers (available only in PSoC 4200 family)
0xE0000000 – 0xE00FFFFF	Cortex-M0 PPB registers
0xF0000000 – 0xF0000FFF	CoreSight ROM

Section D: System-Wide Resources

This section encompasses the following chapters:

Top Level Architecture

System-Wide Resources Block Diagram



7. I/O System

This chapter discusses PSoC® 4's I/O system, its features, architecture, various operating modes, dedicated functionalities, and interrupts. Pins are grouped into ports – there are eight pins per port and the largest PSoC 4 contains up to 4.5 ports.

The input/output (I/O) system provides an interface between the PSoC and the outside world. The flexibility of PSoC devices and the capability of its I/O to route some signals to any pin simplifies circuit design and board layout. Although some critical blocks have dedicated pins and restricted routability, the I/O system offers a large number of configurations to support several types of I/O operations for mixed-signal systems. It is recommended to use dedicated pins, whenever available, for higher performance.

7.1 Features

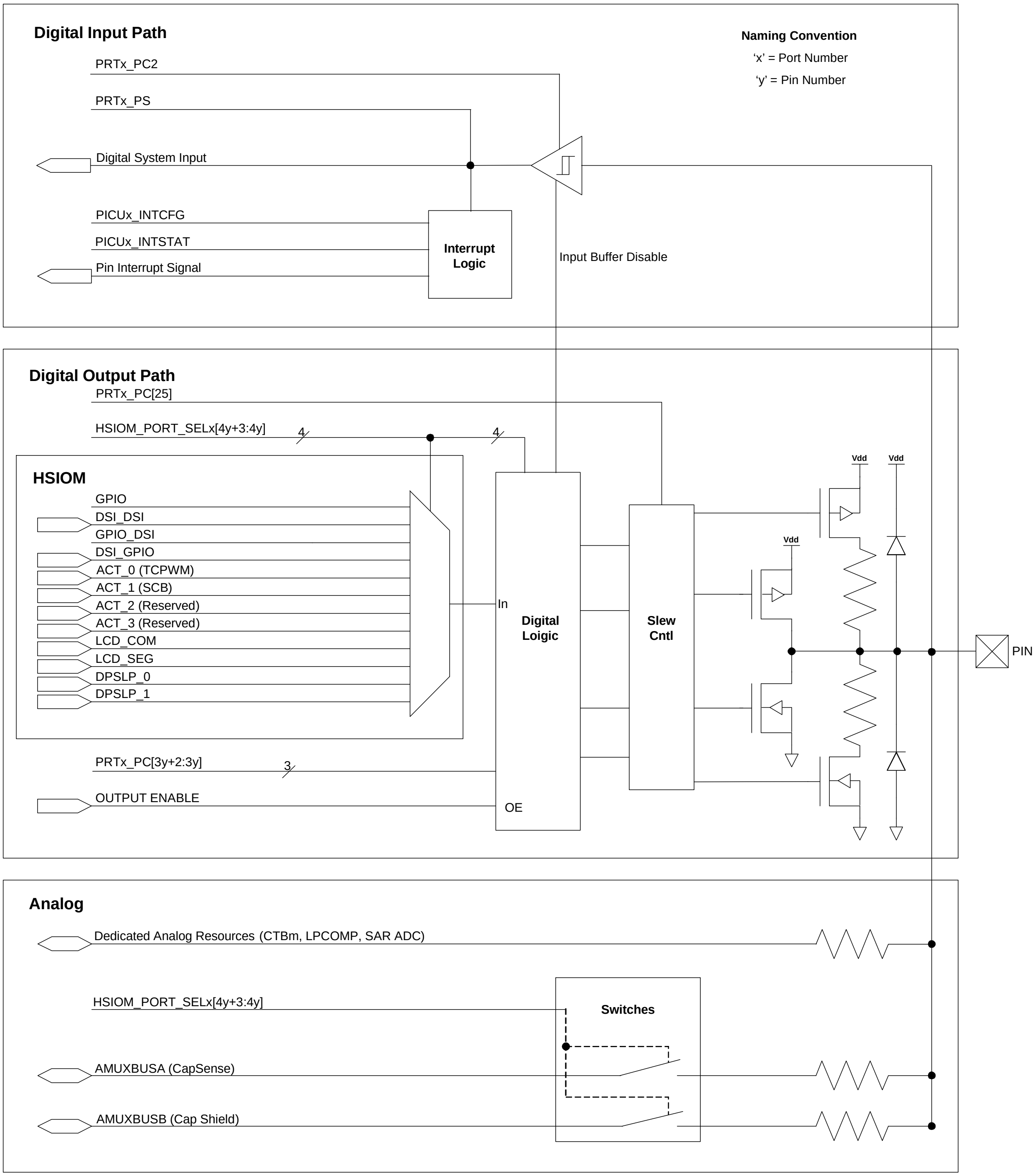
The PSoC 4 I/O system has these features:

- Analog and digital input and output capability
- LCD drive support
- CapSense support
- 8-mA sink and 4-mA source current
- Separate port read (PS) and write (DR) data registers to avoid read modify write errors
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Selectable CMOS and low-voltage LVTTL input thresholds

7.2 Block Diagram

Figure 7-1 explains the various blocks and signals that drive the I/Os.

Figure 7-1. I/O Block Diagram



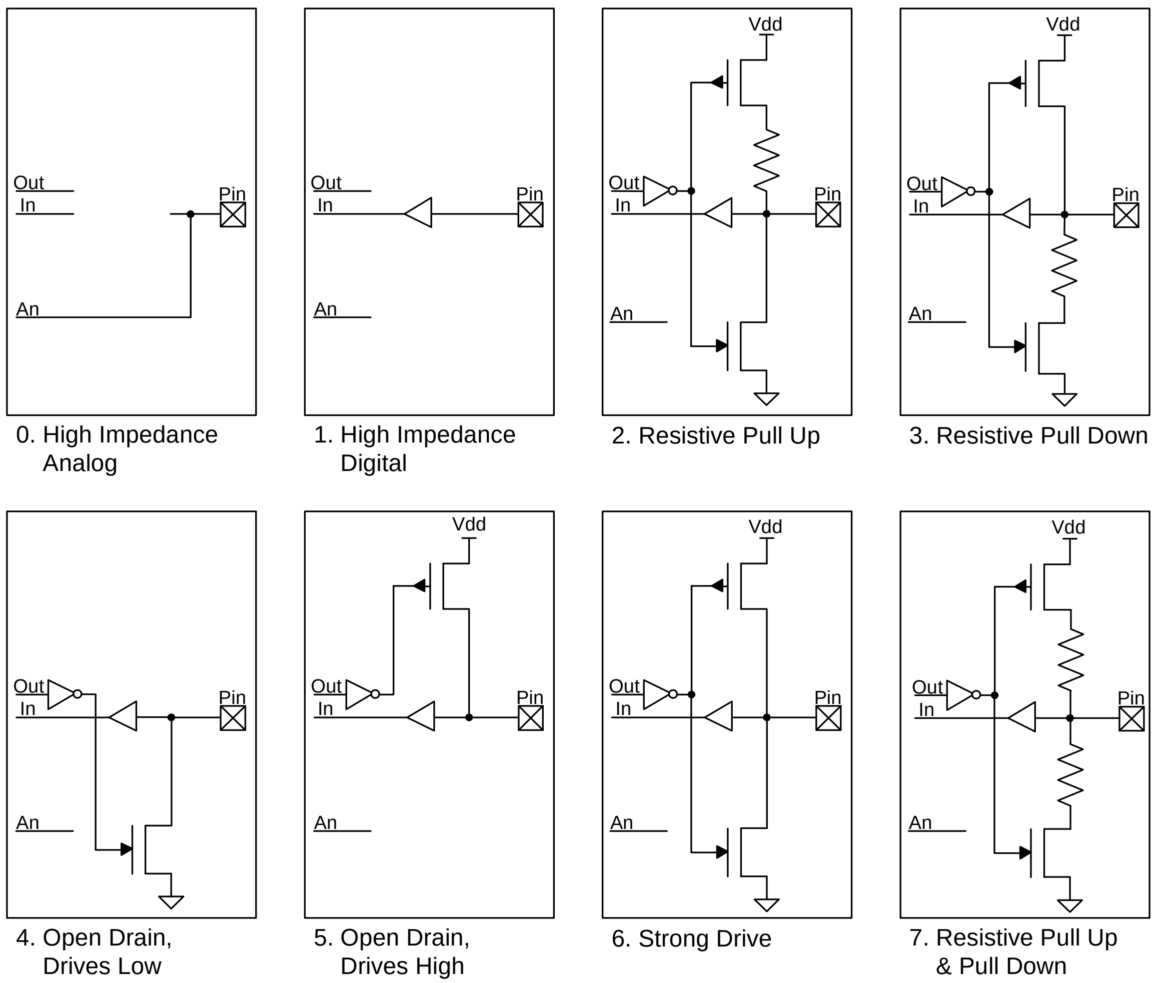
7.3 I/O Drive Modes

Each I/O is individually configurable into one of the eight drive modes listed in Table 7-1. Figure 7-2 depicts a simplified pin view based on each of the eight drive modes.

Two Port Configuration registers are used to configure I/Os in PSoC 4: Port Configuration Register (PRTx_PC) and Port Secondary Configuration Register (PRTx_PC2). PC configures the output drive and input buffer state for each pin, and the slew rate (Slew Rate Control on page 54) and input threshold selection (CMOS LVTTL Level Control on page 54) for the whole port. PC2 configures the input buffer for each pin on the port, irrespective of the port control drive mode (PRTx_PC). Each port has dedicated PC and PC2 registers.

PC2 disables the input buffer independent of the port control drive mode. This bit should be set to disable the input buffer when analog signals are present on the pin.

Figure 7-2. I/O Drive Mode Block Diagram



***Here Out = DR/DSI//TCPWM/SCB, In = DSI/PS/SCB, and An = Analog line**

Table 7-1. Drive Mode Settings

PRTx_PC ('x' denotes port no and 'y' denotes pin no)				
Bits	**Drive Mode**	**PRTx_PC [3y+2: 3y]**	**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 (5K)	Strong 0
	Resistive Pull Down	3	Strong 1	Weak 0 (5K)
	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 (5K)	Weak 0 (5K)

Table 7-2. Input Buffer Disable (Port Configuration 2)

PRTx_PC2 ('x' denotes port no and 'y' denotes pin no)		
Bits	**Name**	**Description**
7:0	INP_DIS	Disables the input buffer independent of the port control drive mode. This bit should be set when analog signals are present on the pin.

7.3.1 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 a floating voltage from causing a current to flow into the I/O 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 sleep modes, I/Os must be configured to the high-impedance analog mode.

7.3.2 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 signal input.

7.3.3 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 either for digital input or output in these modes. Interfacing mechanical switches is a common application. If a pull-up is required in the Resistive Pull-Up Drive mode, a ‘1’ must be written to that pin’s Data Register bit. If a pull-down is required with the Resistive Pull-Down Drive mode, a ‘0’ must be written to that pin’s Data Register. These drive modes are used when interfacing PSoC with open drain drive line. Resistive pull-up is used when input is open drain low and resistive pull-down is used when input it open drain high.

7.3.4 Open Drain, Drives High, and Drives Low

Open drain modes provide high-impedance in one of the data states and strong drive in the other. Pins can be used as digital input or output in these modes. These drive modes are used when a signal is externally pulled up or pulled down. Open drain drive high mode is used when signal is externally pulled down and open drain drive low is used when signal is externally pulled down.

A common application for these modes is driving I^2C bus signal lines.

7.3.5 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 to drive digital output signals or external FETs.

7.3.6 Resistive Pull-Up and Pull-Down

The resistive pull-up and pull-down mode is a single mode and is similar to the resistive pull-up and resistive pull-down modes, except that the pin is always in series with a resistor. The high data state is pulled up while the low data state is pulled down. This mode is used when the bus is driven by other signals that may cause shorts.

7.4 Slew Rate Control

I/O pins have fast and slow output slew rate options for strong drive mode; this can be configured using PRTx_PC[25] bit. Slew rate is individually configurable for each port. This bit is cleared by default and the port works in fast slew mode. For slow slew rate, set this bit. The fast slew rate is for signals between 1 MHz and 33 MHz. Slower slew rate results in reduced EMI and crosstalk; hence, the slow option is recommended for signals that are not speed critical – generally less than 1 MHz.

7.5 CMOS LVTTL Level Control

I/O pins can work at two voltage levels. These levels can be selected by writing to the PRTx_PC[24] bit.

Input level is individually configurable for each port. This bit is cleared by default and the port works in CMOS mode. Set this bit, to configure the port on LVTTL mode.

CMOS mode can be used in most cases, whereas LVTTL can be used for custom interface requirements, which works at lower voltage levels.

7.6 High-Speed I/O Matrix

High-speed I/O matrix (HSIOM) is a high-speed switch that helps to route an I/O to a specific resource inside PSoC. These resources can vary from analog sources such as the AMUXBUS and CapSense to digital sources such as TCPWMs, SCBs or the LCD controller. It also allows selecting Active and Deep-Sleep power domain sources for a pin (see Pin Specific Sources on page 58). Port 4 has a few restrictions in DSI routing; hence, HSIOM Port Settings are not valid for pins of port 4. See Restrictions on Port 4 on page 59 for details. HSIOM_PORT_SELx are 32-bit wide registers, where four dedicated bits are assigned to each pin. They can provide up to 16 different options for I/O pin routing. This selection provides different functions, as listed in Table 7-3, by connecting I/O pins to different peripherals. These roles include:

- I/O
- Fixed-function peripheral
- LCD controller
- SWD debug
- Programmable digital interface (DSI)
- External clock input
- I2C
- SPI
- SWD
- Wakeup

Some of these functions such as I2C, wakeup, and SWD have dedicated pin resource connections. Table 7-4 gives a list of these dedicated pins.

Table 7-3. HSIOM Port Settings

HSIOM PORT_SELx ('x' denotes port no and 'y' denotes pin no)			
Bits	**Name**	**Value**	**Description**
4y+3: 4y	SEL'y'		Selects pin 'y' source ($0 \leq y \leq 7$)
	I/O	0	Pin is regular firmware controlled I/O or connected to dedicated hardware block.
	I/O_DSI	1	Output is firmware controlled, but OE is controlled from DSI.
	DSI_DSI	2	Both Output and OE are controlled from DSI.
	DSI_I/O	3	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, digital I/O driver is connected to csd_charge signal (pin is still connected to AMUXBUS-B).
	ACT_0	8	Pin specific Active source #0 (TCPWM).
	ACT_1	9	Pin specific Active source #1 (SCB UART).
	ACT_2	10	Reserved
	ACT_3	11	Reserved
	LCD_COM	12	Pin is an LCD common pin. This mode remains active and usable in Deep-Sleep mode (provided the LCD block is enabled and configured correctly).
	LCD_SEG	13	Pin is an LCD segment pin. This mode remains active and usable in Deep-Sleep mode (provided the LCD block is enabled and configured correctly).
	DPSLP_0	14	Pin specific Deep-Sleep source #0 (either SCB I2C, SWD or Wakeup).
	DPSLP_1	15	Pin specific Deep-Sleep source #1 (SCB SPI).

7.7 Analog I/O

Analog resources such as LPCOMP, SARMUX, and CSD modulator capacitor, which need cleaner/low-impedance analog signals have dedicated pins. Other analog resources such as CSD sensor inputs and LCD, which do not require dedicated analog inputs use analog mux lines.

Dedicated analog pins provide direct connections to specific analog blocks, such as CTBm, external reference, LPCOMP, and SARMUX. Dedicated pins help improve performance and should be given priority over other pins when using these analog resources. See Table 7-4 for dedicated pins.

To configure an I/O as a dedicated analog I/O, it should be configured in high-impedance analog mode (Table 7-1) and the respective connection should be enabled in the specific analog resource. This is usually done via registers associated with the analog resources.

To configure an I/O as an analog pin connecting to AMUX bus, it should be configured in high-impedance analog mode (Table 7-1) and then routed to AMUX bus using HSIOM_PORT_SELx register (Table 7-3).

7.8 LCD Drive

All I/O pins have the capability of driving an LCD common or segment. HSIOM_PORT_SELx registers are used to enable individual pins for LCD drive. The same register is used to select whether a pin is set as a common or segment drive pin (Table 7-3). See the LCD Direct Drive chapter on page 233 for details.

7.9 CapSense

All I/O pins can be configured as a CapSense element (Table 7-3) such as button, slider segment, and proximity sensor. CapSense uses the analog mux for connecting any pin as a capacitive sensor.

AMUXBUS-A is used as a capacitive sensor element and AMUXBUS-B can be configured as a shield signal for the capacitive sensors. See the CapSense chapter on page 245 for more information.

Table 7-4. Dedicated Pins

Function	Signal Name	Signal Type	Pad Name	Comment
External Reference	ext_vref	Analog	P1[7]	External reference for SAR ADC
Opamp 0	ctb.oa0.out	Analog	P1[2]	Output of CTBm Opamp 0
Opamp 1	ctb.oa1.out	Analog	P1[3]	Output of CTBm Opamp 1
External Clock	exe_clk	Digital	P0[6]	Provides an option to use external clock instead of IMO
I2C	SCL_0	Digital	P4[0]	
	SDA_0	Digital	P4[1]	
	SCL_1	Digital	P0[4]/P3[0]	Two options available
	SDA_1	Digital	P0[5]/P3[1]	Two options available
SPI	MOSI_0	Digital	P4[0]	
	MISO_0	Digital	P4[1]	
	CLK_0	Digital	P4[2]	
	SSEL0_0	Digital	P4[3]	
	SSEL1_0	Digital	P0[0]	
	SSEL2_0	Digital	P0[1]	
	SSEL3_0	Digital	P0[2]	
	MOSI_1	Digital	P0[4] / P3[0]	Two options available
	MISO_1	Digital	P0[5] / P3[1]	Two options available
	CLK_1	Digital	P0[6] / P3[2]	Two options available
	SSEL0_1	Digital	P0[7] / P3[3]	Two options available
	SSEL1_1	Digital	P3[4]	
	SSEL2_1	Digital	P3[5]	
	SSEL3_1	Digital	P3[6]	
SWD	IO	Digital	P3[2]	
	CLK	Digital	P3[3]	
WAKEUP	WAKEUP	Digital	P0[7]	

7.10 I/O Port Reconfiguration

Drive mode and pin connections of I/O ports can be reconfigured in runtime by changing the value of the PRTx_PC and HSIOM_PORT_SELx registers. Take care to retain the pin state during reconfiguration of pins when they are connected directly to a digital peripheral. If the ports are driven by the data registers, state maintenance is automatic. However, if the ports are bypassed and driven by the DSI, the current value must be read and written to the data register (PRTx_DR) before initiating reconfiguration. During port configuration, the current configuration should be saved as follows:

1. Read the I/O pin state, PRTx_PS in software.
2. Write the PRTx_PS value into the data registers, PRTx_DR.
3. Change the corresponding field in PORT_SELx to drive the pin by the data register, PRTx_DR.

7.11 I/O State on Power Up

By default, during power up all I/Os are in high-impedance analog state. Input buffers are disabled during power-up. When the chip is powered up, its I/O is configured by writing to the associated registers. See Registers on page 59.

7.12 Sleep Mode Behavior

The I/O pad maintains the current pin state during sleep modes. In sleep mode, all the I/Os are active and can be driven by active peripherals.

7.13 Low-Power Behavior

PSoC 4 supports three low-power states: Deep-Sleep, Hibernate, and Stop.

In Deep-Sleep mode (Table 7-3), only dedicated deep-sleep pins connected to deep-sleep peripherals are functional; the remaining pin signals are latched into I/O pins.

In the Hibernate and Stop modes, all the I/O pins are latched and remain in frozen state. See the Power Modes chapter on page 75 for details.

7.14 Port Interrupt Controller Unit

This section describes the functions of the port interrupt controller unit (PICU) for PSoC I/O.

7.14.1 Features

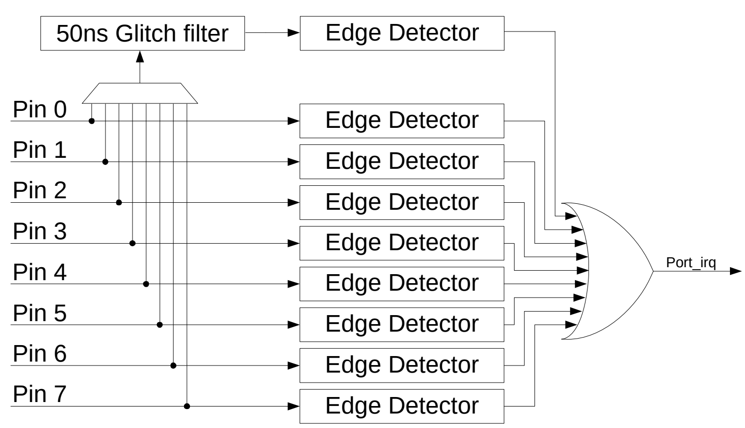
The features of the PICU are:

- All eight pins in each port interface with their own PICU and associated interrupt vector
- Pin status bits provide easy determination of interrupt source down to the pin level
- Rising/falling/either edge interrupts are handled
- Pin interrupts can be individually enabled or disabled
- AHB interfaces for read and write into its registers
- Sends out a single interrupt request (PIRQ) signal to the interrupt controller

7.14.2 Interrupt Controller Block Diagram

Each port has its own individual Interrupt Request and associated IRQ vector and ISR. Additionally, one pin can be selected on each port that is routed through a 50-ns glitch filter to form a ninth glitch-tolerant interrupt. Figure 7-3 describes how this is combined.

Figure 7-3. Interrupt Controller



7.14.3 Function and Configuration

Each pin of the port can be configured independently to generate interrupt on rising edge, falling edge, or either edge by writing to the PRTx_INTCFG register (Table 7-5). Level sensitive interrupts are not supported. Apart from this PRTx_INTCFG is also used to route a specific channel to glitch filter to generate a ninth glitch tolerant interrupt.

A single register is provided for each pin and to select the pin number and mode for the ninth interrupt.

When a port interrupt is triggered by a signal change on an enabled port pin, PRTx_INTSTAT register is updated. Firmware reads this register to determine which of the pins on the port triggered the IRQ. Firmware can then clear the IRQ bit by writing a ‘1’ to its corresponding bit.

The steps to service an interrupt are:

1. Depending on the configured mode for each pin, whenever the selected edge occurs on a pin, its corresponding status bit in the status register is set to '1', and an interrupt request is sent to the interrupt controller.
2. Status bits that have '1' are cleared upon a read of the status register. Other bits of the status register can still respond to incoming interrupt sources.
3. If an interrupt is pending, and the status register is being read, all of the incoming events on the same interrupt source (I/O) are blocked until the read is complete.

However, all of the other interrupt sources that were not pending an interrupt in status register are not blocked.

Additionally, when the Port Interrupt Control Status Register is read at the same time an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. So when using PICU interrupts, it is always recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of code.

Table 7-5. Interrupt Configuration Register

PRTx_INTCFG ('x' denotes port no and 'y' denotes pin no)			
Bits	**Interrupt Type**	**PRTx_INTCFG**	**Description**
2y+1: 2y	PRTx_INTCFG[2y+1:2y]		Selects Interrupt mode for Pin 'y' on the port ($0 \leq y \leq 7$)
	Disable	0	Interrupt Disabled
	Rising	1	Interrupt at rising edge of the signal
	Falling	2	Interrupt at falling edge of the signal
	Both	3	Interrupt at either edge of the signal
17:16	SEL INTYPE_FLT		Selects Interrupt mode for ninth glitch free input
	Disable	0	Interrupt Disabled
	Rising	1	Interrupt at rising edge of the signal
	Falling	2	Interrupt at falling edge of the signal
	Both	3	Interrupt at either edge of the signal
20:18			Selects which pin gets routed to glitch filter

Table 7-6. Port Interrupt Status Register

PRTx_INTSTAT ('x' denotes port no and 'y' denotes pin no)		
Bits	**Interrupt Type**	**Description**
y	PRTx_INTSTAT[y]	Returns Interrupt status for Pin 'y' on the port ($0 \leq y \leq 7$)
8	PRTx_INTSTAT_FLT	Returns Interrupt status for glitch tolerant interrupt

7.15 Input and Output Synchronization

For digital input and output signals I/O provides synchronization with internal clock or a digital signal as clock. By default, HFCLK is used for synchronization but any other clock can also be used.

This feature and other clock and reset features are implemented using a combination of UDB port adapter and I/O blocks. See Port Adapter Block on page 161 for details.

7.16 Pin Specific Sources

As explained in High-Speed I/O Matrix on page 54, there are two pin specific active power domain sources and two deep-sleep power domain sources in HSIOM.

Active sources such as timer, counter, and PWM block are available only when the PSoC is working in active mode. They are disabled when the device is put in any of the low-power modes.

Deep-sleep sources such as the SCB block and LCD direct drive are available in Active, Sleep, and Deep-Sleep power modes. They are disabled in Hibernate and Stop modes. See the Power Modes chapter on page 75 for more details.

These sources have restricted routability and can be routed to only specific pins using HSIOM. The list of pin specific sources is given in Table 7-7. Some of the pins do not have any dedicated active or deep-sleep power domain resources and are therefore not included in the table.

Table 7-7. Pin Specific Active and Deep-Sleep Power Domain Sources

Pin	ACT_0[a]	ACT_1	DPSLP_0	DPSLP_1
P0[0]				scb0_spi_ssel_1
P0[1]				scb0_spi_ssel_2
P0[2]				scb0_spi_ssel_3
P0[4]		scb1_uart_rx	scb1_i2c_scl	scb1_spi_mosi
P0[5]		scb1_uart_tx	scb1_i2c_sda	scb1_spi_miso
P0[6]	ext_clk			scb1_spi_clk
P0[7]			wakeup	scb1_spi_ssel_0
P1[0]	tcpwm2_p			
P1[1]	tcpwm2_n			
P1[2]	tcpwm3_p			
P1[3]	tcpwm3_n			
P2[4]	tcpwm0_p			
P2[5]	tcpwm0_n			
P2[6]	tcpwm1_p			
P2[7]	tcpwm1_n			
P3[0]	tcpwm0_p	scb1_uart_rx	scb1_i2c_scl	scb1_spi_mosi
P3[1]	tcpwm0_n	scb1_uart_tx	scb1_i2c_sda	scb1_spi_miso
P3[2]	tcpwm1_p		swd_io	scb1_spi_clk
P3[3]	tcpwm1_n		swd_clk	scb1_spi_ssel_0
P3[4]	tcpwm2_p			scb1_spi_ssel_1
P3[5]	tcpwm2_n			scb1_spi_ssel_2
P3[6]	tcpwm3_p			scb1_spi_ssel_3
P3[7]	tcpwm3_n			
P4[0]		scb0_uart_rx	scb0_i2c_scl	scb0_spi_mosi
P4[1]		scb0_uart_tx	scb0_i2c_sda	scb0_spi_miso
P4[2]				scb0_spi_clk
P4[3]				scb0_spi_ssel_0

a. tcpwm connections can be routed to any I/O except port 4, using DSI routing, though its recommended to use the pins mentioned in the table whenever possible.

7.17 Restrictions on Port 4

Port 4 does not have a dedicated port-adapter (Port Adapter Block on page 161). Therefore, none of the port 4 pins can be routed through the DSI. Apart from this, port 4 pins cannot be used for general purpose analog blocks such as SAR ADC and CTBm,LPComp. However, they can still be used as a firmware pin, LCD_COM, LCD_SEG, CapSense, or can be connected to the SCB block through the HSIOM.

Because DSI is unavailable for port 4 pins, the signals at these pins cannot be synchronized with any other clock source. Therefore, port 4 pins should always be used in Transparent mode.

7.18 Registers

Table 7-8. I/O Registers

Name	Quantity	Description
PRTx_DR	4	Port Output Data Register
PRTx_PS	4	Port Pin State Register, Used to read the logical pin state of I/O
PRTx_PC	4	Port Configuration Register, configures the output drive mode, input threshold and slew rate
PRTx_PC2	4	Port Secondary Configuration Register, Configures the input buffer of I/O
PRTx_INTCFG	4	Port Interrupt Configuration Register
PRTx_INTSTAT	4	Port Interrupt Status Register
HSIOM_PORT_SELx	4	HSIOM Port Selection Register

8. Clocking System

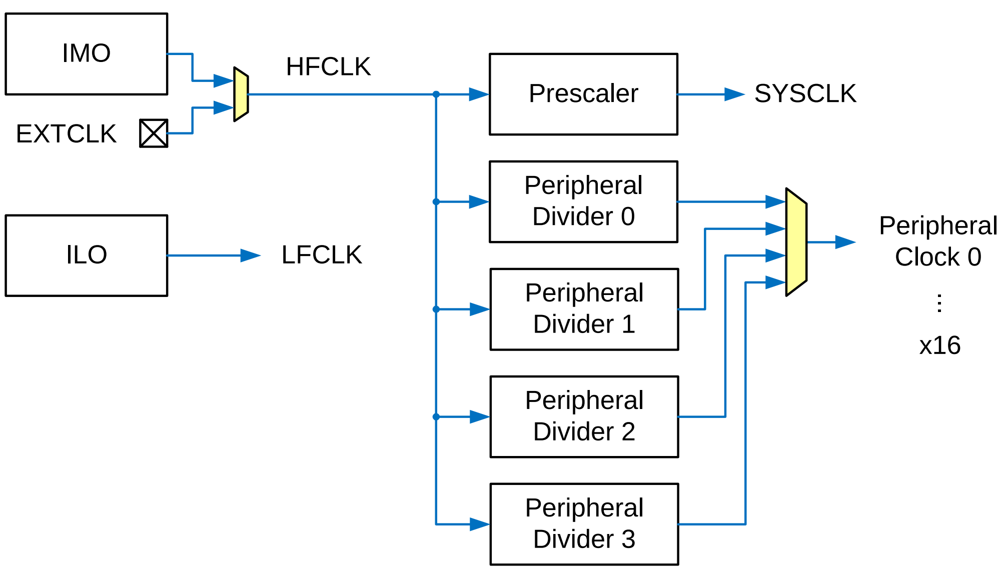
The PSoC® 4 clock system includes these clock resources:

- Two internal clock sources:
  - 3–48 MHz internal main oscillator (IMO) ±2 percent across all frequencies with trim
  - 32 kHz internal low-speed oscillator (ILO) ±60 percent with trim
- External clock (EXTCLK) generated using a signal from an I/O pin
- High-frequency clock (HFCLK) selected from IMO or external clock
- Low-frequency clock (LFCLK) sourced by ILO
- Dedicated prescaler for system clock (SYSCLK) sourced by HFCLK
- Four peripheral clock dividers, each containing three chainable 16-bit dividers
- 16 digital and analog peripheral clocks

8.1 Block Diagram

Figure 8-1 gives a generic view of the clocking system in PSoC 4 devices.

Figure 8-1. Clocking System Block Diagram



The three clock sources in the device are shown in Figure 8-1, on the left. The mux selects the HFCLK from an external clock source or the IMO. The ILO sources the LFCLK. The prescaler and dividers generate the SYSCLK and the individual peripheral clocks.

8.2 Clock Sources

8.2.1 Internal Main Oscillator

The internal main oscillator operates with no external components and outputs a stable clock at a variety of user-selectable frequencies spanning 3–48 MHz in 1-MHz increments. Frequencies are selected by setting the frequency range in register CLK_IMO_TRIM2, setting the IMO trim in register CLK_IMO_TRIM1, and finally setting the bandgap trim in registers PWR_BG_TRIM4 and PWR_BG_TRIM5. Each device has IMO trim measured during manufacturing to meet datasheet specifications; the trim is stored in manufacturing configuration data in SFLASH. These values can be retrieved and used at runtime to achieve datasheet specifications. Firmware can retrieve these trim values and reconfigure the device to change the frequency during runtime.

8.2.1.1 Startup Behavior

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

8.2.1.2 IMO Frequency Spread

The IMO is capable of operating in a spread-spectrum mode to reduce the amplitude of noise generated at the IMO’s central operating frequency. This mode causes the IMO to vary in frequency across one of four distributions selected by a register. The four distribution options are fixed frequency, triangle wave, pseudo-random, and DSI input. The DSI input mode allows you to specify the pattern with a digital signal. The distribution options are selected with register CLOCK_IMO_SPREAD bits SS_MODE, which are shown in Table 8-1. The limits of the distribution are defined with register CLOCK_IMO_SPREAD bits SS_RANGE, which are shown in Table 8-2. All spread options are downspread, meaning that instantaneous clock frequency values are always at or below the configured frequency.

Table 8-1. IMO Spread-Spectrum Distribution Mode Bits SS_MODE

Name	Description
SS_MODE[1:0]	IMO spread-spectrum mode. Defines the shape of the spread-spectrum frequency distribution. 0: Off. IMO frequency is not changed. 1: Triangle. IMO frequency forms a triangular distribution about the center frequency. Count limits are defined by bits SS_MAX[4:0]. 2: Pseudo-random. IMO frequency forms a pseudo-random distribution about the center frequency. 3: DSI. IMO frequency distribution is determined using a DSI input signal.

Table 8-2. IMO Spread Spectrum Distribution Range Bits SS_RANGE

Name	Description
SS_RANGE[1:0]	IMO spread-spectrum maximum range. Defines the frequency spread from nominal at the extreme count values of the spread-spectrum’s counter. 0: 1%. Spread-spectrum varies in frequency from 0 to –1% at the extreme count values. 1: 2%. Spread-spectrum varies in frequency from 0 to –2% at the extreme count values. 2: 4%. Spread-spectrum varies in frequency from 0 to –4% at the extreme count values. 3: Reserved. Do not use.

8.2.2 Internal Low-speed Oscillator

The internal low-speed oscillator operates with no external components and outputs a stable clock at 32 kHz nominal. The ILO is relatively low power and low accuracy. It is available in all power modes except Hibernate and Stop modes. The ILO is always used as the system low-frequency clock LFCLK in PSoC 4. The ILO is enabled and disabled with register CLK_ILO_CONFIG bit ENABLE.

8.2.3 External Clock

The external clock is a MHz range clock that can be generated from a signal on a designated PSoC 4 pin. This clock may be used in place of the IMO as the source of the system high-frequency clock HFCLK. The allowable range of external clock frequencies is 0–48 MHz. PSoC 4 always starts up using the IMO, and the external clock must be enabled in user mode, so the device cannot be started from a reset clocked by the user clock.

8.3 Clock Distribution

PSoC 4 clocks are developed and distributed throughout the part, as shown in Figure 8-1. The distribution configuration options are as follows:

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

8.3.1 HFCLK Input Selection

HFCLK in PSoC 4 has two input options: IMO and EXTCLK. The HFCLK input is selected using register CLK_SELECT bits DIRECT_SEL, as described in Table 8-3.

Table 8-3. HFCLK Input Selection Bits DIRECT_SEL

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

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the I/O System chapter on page 51 for more details.

8.3.2 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 prescaler is capable of dividing the HFCLK by powers of 2 between 2^0 = 1 and 2^7 = 128. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in Table 8-4.

Note SYSCLK cannot exceed 24 MHz for the PSoC 4100 family.

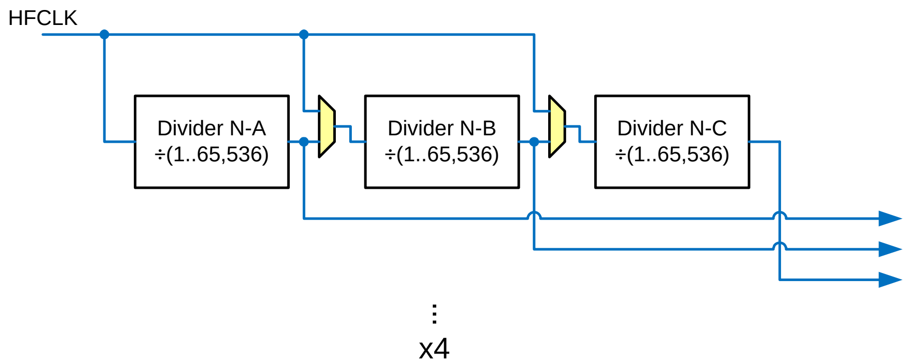
Table 8-4. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

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

8.3.3 Peripheral Clock Divider Configuration

PSoC 4 has four divider banks, each of which contains three 16-bit dividers, A, B, and C, which can be cascaded to further divide clocks. One of the four banks is capable of fractional divides, which allows the clock divisor to include a fraction of 0..31/32. These four divider banks are used to generate all of the analog and digital peripheral clocks in the device. Figure 8-2 shows a block diagram of the cascaded dividers. The peripheral clocks are generated from the intermediate and final outputs of the clock dividers.

Figure 8-2. Peripheral Clock Divider Block Diagram



The three non-fractional clock divider banks are configured with the DIVIDER_A, DIVIDER_B, and DIVIDER_C registers. The fractional clock divider bank is configured with the DIVIDER_FRAC_A, DIVIDER_FRAC_B, and DIVIDER__FRAC_C registers. Table 8-5 and Table 8-6 describe the configurations for these registers.

Table 8-5. Non Fractional Peripheral Clock Divider Configuration Register DIVIDER_x

Bits	Name	Description
15:0	DIVIDER_x	Divide value for divider x in the row. Output = input / (DIVIDER_x +1)
30	CASCADE_x–1_x	Determines the input of divider x in the row. 0: DIVIDER_x clock input driven by HFCLK 1: DIVIDER_x clock input driven by the output of DIVIDER_x–1 **Note** No effect for DIVIDER_A
31	ENABLE_x	Enables DIVIDER_x.

Table 8-6. Fractional Peripheral Clock Divider Configuration Register DIVIDER_FRAC_x

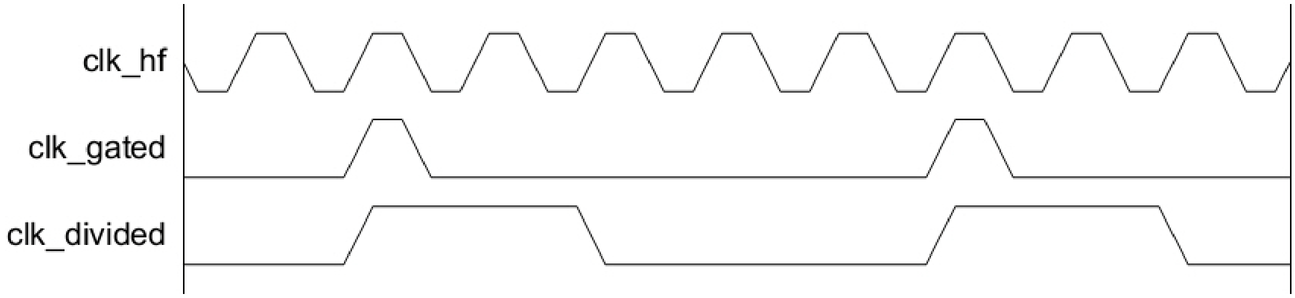
Bits	Name	Description
15:0	DIVIDER_x	Divide value for divider x in the row. Output = input / (DIVIDER_x +1 + FRAC_x/32)
20:16	FRAC_x	Fractional divider numerator value for divider x in the row. Output = input / (DIVIDER_x +1 + FRAC_x/32)
30	CASCADE_x–1_x	Determines the input of divider x in the row. 0: DIVIDER_x clock input driven by HFCLK 1: DIVIDER_x clock input driven by the output of DIVIDER_x–1 **Note** No effect for DIVIDER_A
31	ENABLE_x	Enables DIVIDER_x.

8.3.4 Peripheral Clock Configuration

The four UDB clocks and 12 additional peripheral clocks, including the analog SAR clock, are sourced by peripheral clock dividers. Each divider input can be used to generate two versions of the clock: a gated clock and a divided clock. The gated version produces one in N clocking, where the pulse width of the clock is the same as the HFCLK, but the frequency is divided. The divided version has as close as possible to 50 percent duty cycle, with the edges of the divided clock always occurring on high edges of the HFCLK. When divided by n, the divided version will be high for n/2 rounded down cycles, and low for n/2 rounded up cycles. This is shown in Figure 8-3.

Clk_gated is used by most peripherals because they are impacted only by rising edges. However, in certain peripherals that are negative edge sensitive as well, clk_divided may be preferred.

Figure 8-3. UDB and Peripheral Clock Timing Diagram



Each of the digital peripheral clocks is mapped to a specific digital peripheral; Table 8-7 shows the mapping. Each clock is configured using one of the 16 SELECT registers.

Table 8-7. Peripheral Clock Mapping

Peripheral Clock #	Peripheral
0	IMO (SS)
1	SARPUMP
2	SCB0
3	SCB1
4	LCD
5	CSD (1)
6	CSD (2)
7	SAR
8	TCPWM0
9	TCPWM1
10	TCPWM2
11	TCPWM3
12	UDB0 (available only in PSoC 4200)
13	UDB1 (available only in PSoC 4200)
14	UDB2 (available only in PSoC 4200)
15	UDB3 (available only in PSoC 4200)

Table 8-8. Peripheral Clock Configuration Register SELECT

Bits	Name	Description
3:0	DIVIDER_N	Select divider bank row to source clock from. 0 to 2: non-fractional divider 0 to 2 3: fractional divider 0
5:4	DIVIDER_ABC	Selects which divider from row N to use: 0: Clock disabled 1: Divider N-A 2: Divider N-B 3: Divider N-C

The SAR clock is derived from the clock dividers similar to other peripheral clocks. Unlike the other peripheral clocks, the SAR clock generates two outputs: a skewed and ungated 50 percent duty cycle version for analog circuits, and a version synchronized with HFCLK for digital circuits. The skew allows analog sampling to occur independently from digital clock transitions, which can improve analog performance.

8.4 Low-Power Mode Operation

PSoC 4 clock behavior is different in different power modes. The MHz frequency clocks including the IMO, EXTCLK, HFCLK, SYSCLK, and peripheral clocks only operate in Active and Sleep modes. The ILO and LFCLK operate in all power modes except Hibernate and Stop.

9. Power Supply and Monitoring

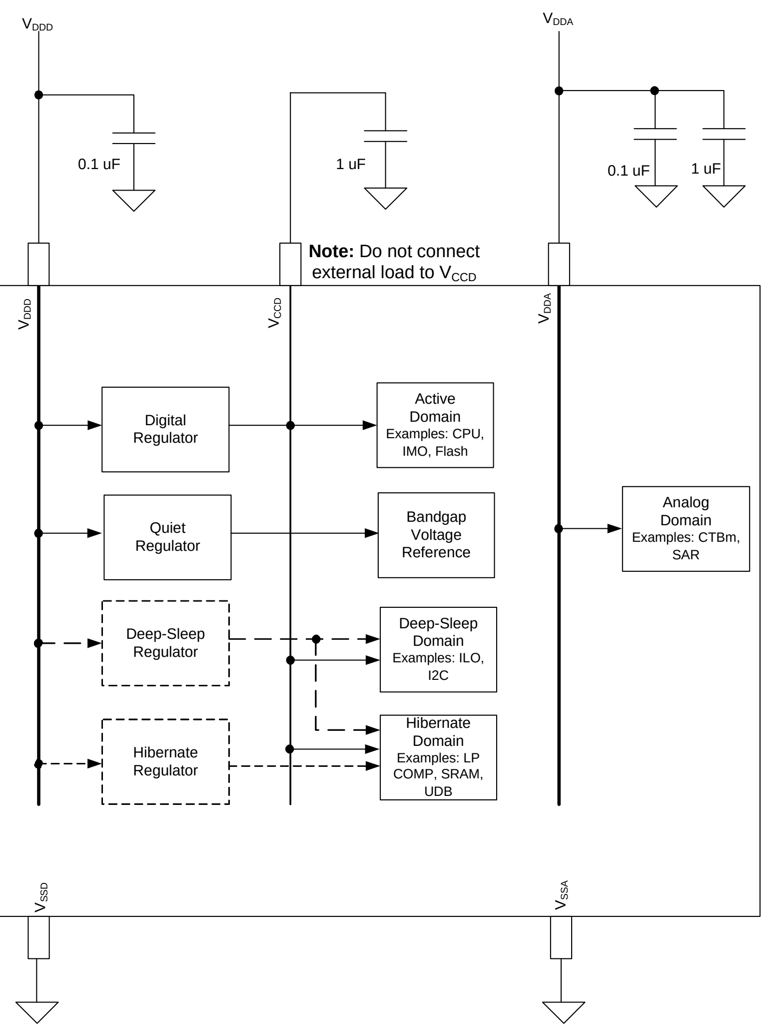
PSoC® 4 is capable of operating from a single 1.71 V to 5.5 V externally supplied voltage. This is supported through one of the following operating ranges:

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

PSoC 4 devices have different internal regulators to support various power modes. These include active digital regulator, quiet regulator, deep-sleep regulator, and hibernate regulator.

9.1 Block Diagram

Figure 9-1. Power System Block Diagram



The power system has separate digital and analog supply pins labeled V_{DDD} and V_{DDA}, as shown in Figure 9-1. Similarly, there are separate digital and analog ground pins named V_{SSD} and V_{SSA}. In some PSoC 4 device packages, V_{DDD} and V_{DDA} are shorted internally and made available as a single V_{DD} pin. **On PSoC 4 systems, the V_{DDA} supply must always be equal to the V_{DDD} supply.** Typically, this is achieved by supplying V_{DDA} and V_{DDD} from the same source. Even if both V_{DDA} and V_{DDD} are supplied by the same source, separate power supply routes are recommended in precision analog applications to isolate digital and analog currents. Similarly, separate V_{SSA} and V_{SSD} routes are also recommended.

If two different sources are used to provide V_{DDA} and V_{DDD}, V_{DDA} supply must be present before V_{DDD} supply.

One active digital regulator is provided to allow the external V_{DDD} supply to be regulated to the nominal 1.8 V required for the digital core. The output pin of this regulator has specific capacitor requirement, as shown in Figure 9-1. This active digital regulator is designed to supply the internal circuits only and **should not be loaded externally**.

The primary regulated supply, labeled V_{CCD}, can be configured for internal regulation or can be directly supplied by the pin. In internal regulation mode, V_{DDD} can vary between 1.8 V and 5.5 V and the on-chip regulators generate the other low voltage supplies.

In direct supply configuration, V_{CCD} and V_{DDD} must be shorted together and connected to a supply of 1.71 V to 1.89 V. The active digital regulator is still powered up and enabled by default. It must be disabled by the firmware to reduce power consumption; see 9.3.1.1 Active Digital Regulator.

Two additional regulators are used to provide supplemental power domains including deep-sleep and hibernate. In addition, a quiet regulator powers sensitive analog circuitry including the bandgap reference and capacitive sensing sub-system.

9.2 Power Supply Scenarios

The following diagrams illustrate the different ways in which the PSoC 4 device is powered.

9.2.1 Single 1.8 V to 5.5 V Unregulated Supply

Depending on board design, the 1.8 V to 5.5 V supply can reach the PSoC 4 device via a single route or two different routes (on boards with separate analog and digital supply networks), as shown in Figure 9-2 and Figure 9-3, respectively.

Figure 9-2. Single Power Supply

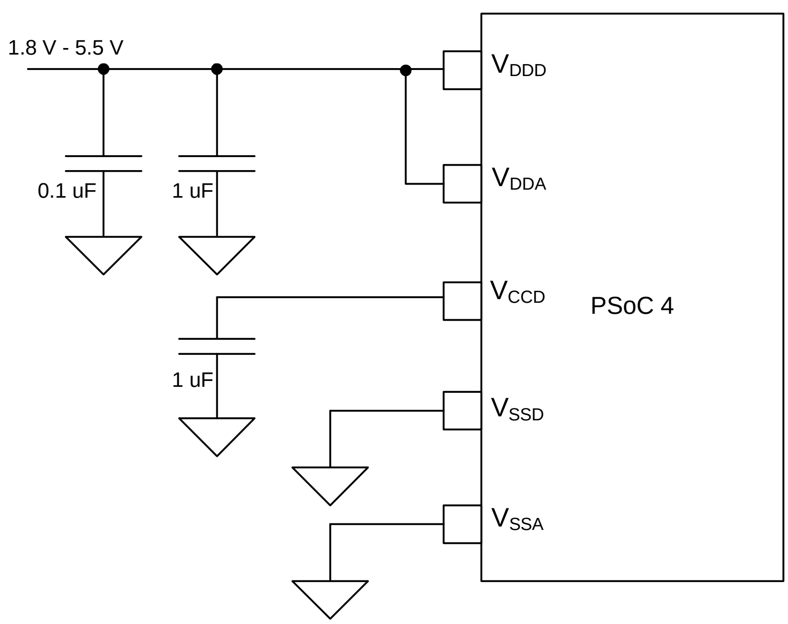
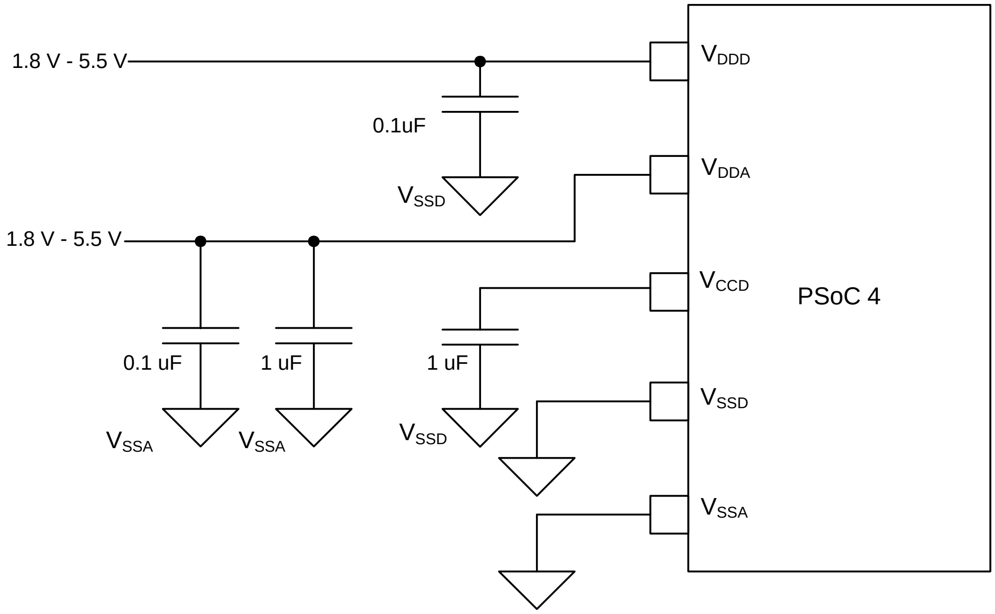
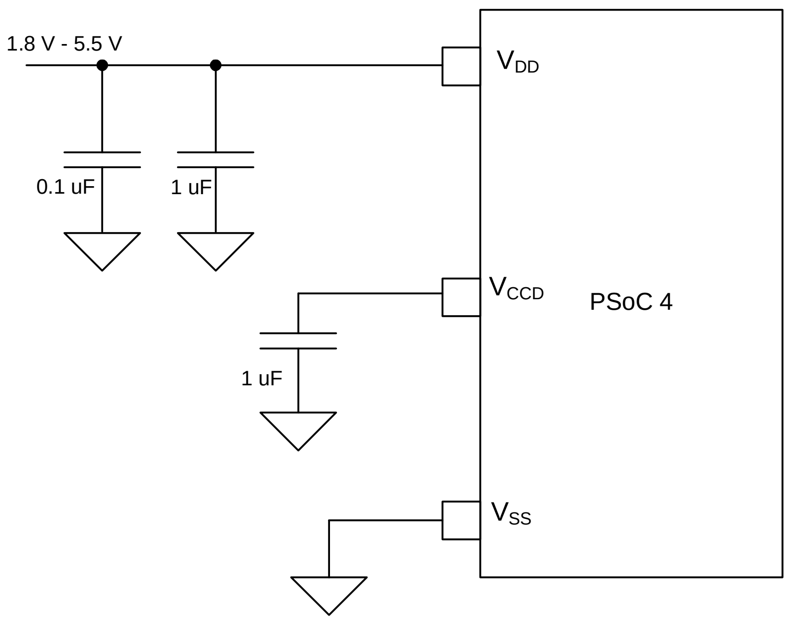


Figure 9-3. Separate V_{DDA} and V_{DDD} Supply Routes



Some PSoC 4 device packages have a single power supply and ground pins labeled V_{DD} and V_{SS}, respectively. The 1.8 V to 5.5 V supply can be connected to these packages, as shown in Figure 9-4.

Figure 9-4. Single V_{DD} Supply



9.2.2 Direct 1.71 V to 1.89 V Regulated Supply

In direct supply configuration, V_{CCD} and V_{DDD} are shorted together and connected to a 1.71 V to 1.89 V supply. This supply can reach the PSoC 4 device via a single route or two different routes, as shown in the following diagrams.

Figure 9-5. Single Power Supply Route

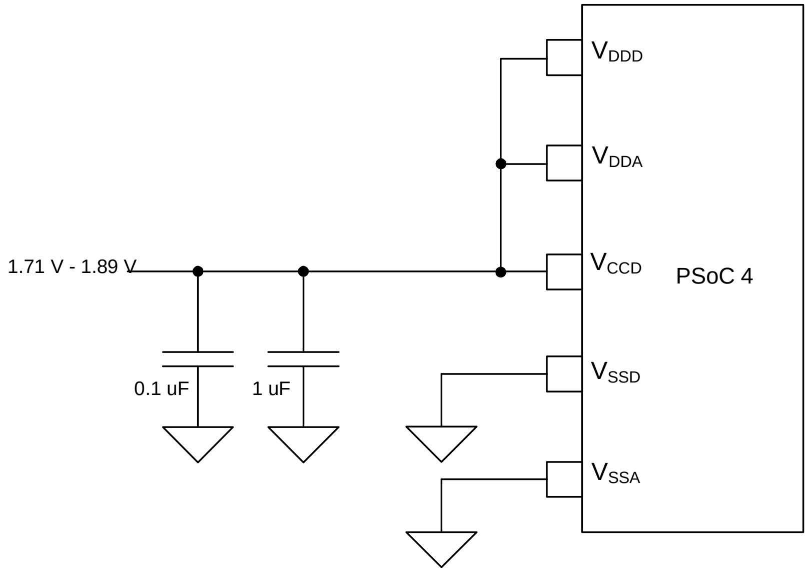
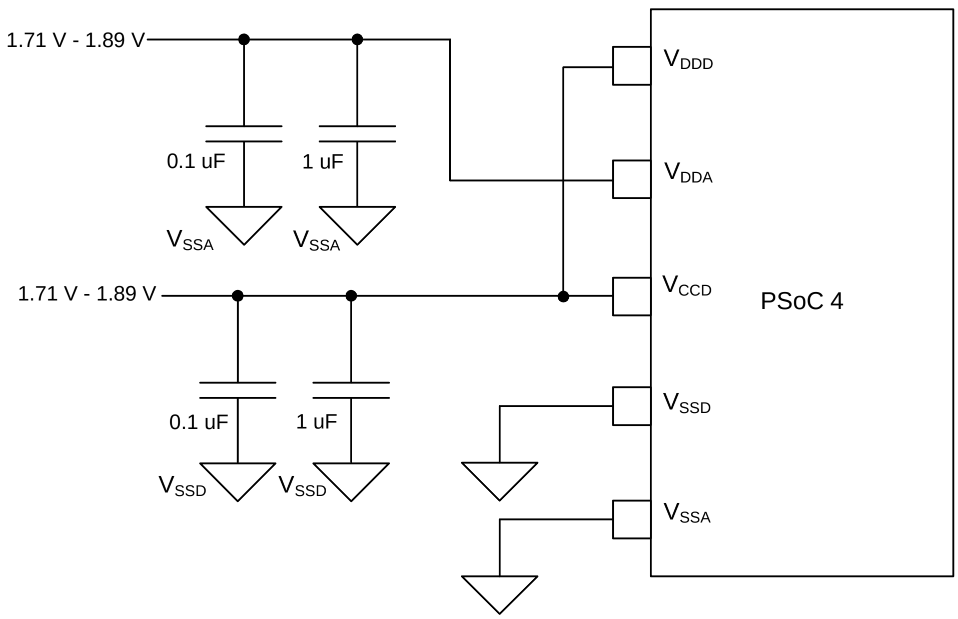
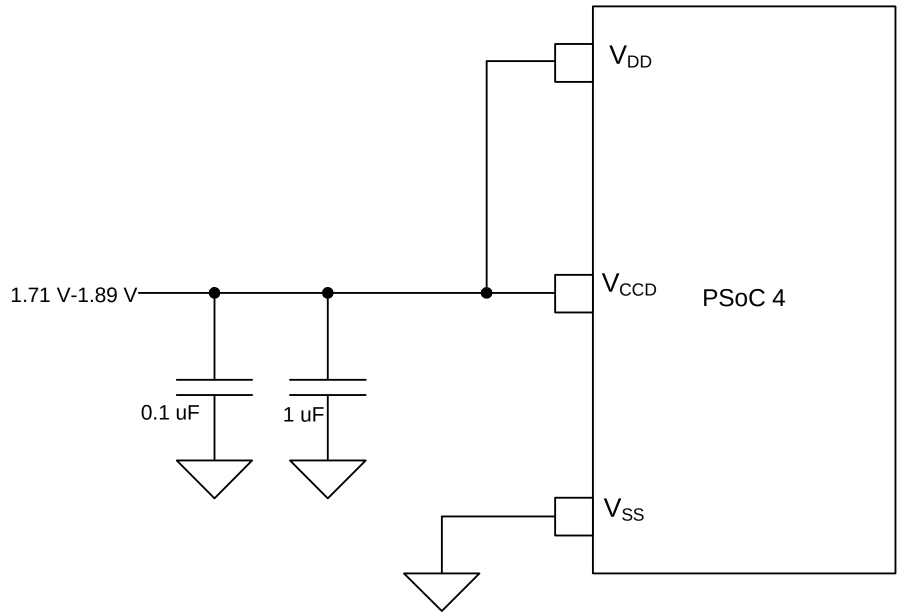


Figure 9-6. Separate Power Supply Route



Some PSoC 4 device packages have a single power supply and ground pins labeled V_{DD} and V_{SS}, respectively. The direct supply connection to these packages is illustrated in Figure 9-7.

Figure 9-7. Single V_{DD} Supply



9.3 How It Works

The regulators in Figure 9-1 power the various domains of the device. All four regulators draw their input power from the V_{DDD} pin supply.

Digital I/Os are supplied from V_{DDD}. The analog circuits run directly from the V_{DDA} input.

9.3.1 Regulator Summary

Active digital regulator and quiet regulator are enabled during the active or sleep power modes. They are turned off in deep-sleep and hibernate power modes (see Table 11-1). The deep-sleep and hibernate regulators are designed to fulfill power requirements in the low-power modes of the device.

9.3.1.1 Active Digital Regulator

For external supplies from 1.8 V and 5.5 V, this regulator supplies the main digital logic in active and sleep modes. This regulator has its output connected to a pin (V_{CCD}) and requires an external decoupling capacitor (1uFX5R).

For supplies below 1.80 V, V_{CCD} must be supplied directly. In this case, V_{CCD} and V_{DDD} must be shorted together as shown in Figure 9-4.

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

9.3.1.2 Quiet Regulator

In active and sleep modes, this regulator supplies analog circuits such as the bandgap reference and capacitive sensing subsystem, which require a quiet supply, free of digital switching noise, and power supply noise. This regulator has a high-power supply rejection ratio. The quiet regulator is available only in active and sleep power modes.

9.3.1.3 Deep-Sleep Regulator

This regulator supplies the circuits that remain powered in deep-sleep mode, such as the ILO and SCB. The deep-sleep regulator is available in all power modes except the hibernate mode. In active and sleep power modes, the main output of this regulator is connected to the output of the digital regulator (V_{CCD}). This regulator also has a separate replica output that provides a stable voltage for the ILO. This output is not connected to V_{CCD} in active and sleep modes.

9.3.1.4 Hibernate Regulator

This regulator supplies the circuits that remain powered in hibernate mode, such as the sleep controller, low-power comparator, and SRAM. The hibernate regulator is available in all power modes. In active and sleep modes, the output of this regulator is connected to the output of the digital regulator. In deep-sleep mode, the output of this regulator is connected to the output of the deep-sleep regulator.

9.3.2 Voltage Monitoring

Voltage detection includes power-on reset (POR), brownout detection (BOD), and low-voltage detection (LVD).

9.3.2.1 Power-On-Reset (POR)

Power-on-reset circuits provide a reset pulse during the initial power ramp. POR circuits monitor V_{CCD} voltage. Typically, the POR circuits are not very accurate with respect to trip-point.

POR circuits are used during initial chip power-up and then disabled.

9.3.2.2 Brownout-Detect (BOD)

These circuits protect the operating/retaining logic from possibly unsafe supply conditions by applying reset to the device. BOD circuit monitors 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 device datasheet for details). The system will not come out of RESET until the supply is detected to be valid again.

To enable firmware to distinguish a normal power cycle from a brownout event, a special register is provided (PWR_BOD_KEY) that will not be cleared after a BOD generated RESET. However, this register will be cleared if the device goes through POR or XRES. BOD is available in all power modes except the stop mode.

9.3.2.3 Low-Voltage-Detect (LVD)

The LVD circuit monitors external supply voltage and accurately detects depletion of the energy source. LVD detector generates an interrupt to cause the system to take preventive measures.

The LVD is available only in active and sleep power modes. If LVD is required in deep-sleep mode, then the chip should be configured to periodically wake up from deep sleep using WDT as the wake up source; the LVD monitoring should be done in active mode. LVD circuits generate interrupts at programmable levels within the safe operating voltage. The trip point of LVD can be configured between 1.75 V to 4.5 V using the LVD_SEL field in PWR_VMON_CONFIG register.

When enabling the LVD circuit, it is possible to get a false interrupt during the initial settling time. Firmware can mask this by waiting for 1 µs after setting the LVD_EN bit in PWR_VMON_CONFIG register. The recommended firmware procedure to enable the LVD function is:

1. Ensure that the LVD bit in the PWR_INTR_MASK register is 0 to prevent propagating a false interrupt.
2. Set the required trip-point in the LVD_SEL field of the PWR_VMON_CFG register.
3. Enable the LVD by setting the LVD_EN bit in the PWR_VMON_CFG. This may cause a false LVD event.
4. Wait at least 1 µs for the circuit to stabilize.
5. Clear the false event by writing a one to LVD bit in the PWR_INTR register. The bit will not clear if the LVD condition is truly present.
6. Unmask the interrupt using the LVD bit in PWR_INTR_MASK.

9.4 Register List

Table 9-1. Power Supply and Monitoring Register List

Register Name	Description
PWR_INTR	Power System Interrupt Register – This register indicates the power system interrupt status.
PWR_INTR_MASK	Power System Interrupt Mask Register – This register controls which interrupts are propagated to the interrupt controller of the CPU.
PWR_VMON_CONFIG	Power System Voltage Monitoring Trim and Configuration – This register contains Trim and configuration bits for Voltage Monitoring System.
PWR_CONTROLDFT_SELECT	Controls the device power mode options and allows observation of current state.

10. Chip Operational Modes

PSoC® 4 is capable of executing firmware in four different modes. These modes dictate execution from different locations in flash and ROM, and different levels of hardware privileges. Only three of the modes are used in end-applications. Debug mode is used exclusively to debug designs during firmware development. PSoC 4's operational modes are:

- Boot
- User
- Privileged
- Debug

10.1 Boot

Boot mode is an operational mode where the part is configured by instructions hard-coded in device ROM. This mode is entered after reset ends, assuming that no debug acquire sequence is received by the part. Boot mode is a privileged mode – interrupts are locked out in this mode so that the boot firmware may set up the device for operation without being interrupted. During the power-up phase, hardware trim settings are loaded from NV-latches to guarantee proper operation during power-up. When boot concludes, user mode is entered, 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 part.

For more details on device startup, see the Reset System chapter on page 85.

10.2 User

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

10.3 Privileged

Privileged mode is an operational mode, which allows execution of special subroutines that are stored in device ROM. These subroutines are written by Cypress and not user-modifiable. They are used to execute proprietary code that is not meant to be interrupted or observed. Debugging is not allowed in privileged mode. Exit from this mode returns the part to user mode.

10.4 Debug

Debug mode is an operational mode that allows observation of the operational parameters of the device, for the purpose of debugging firmware during development. This mode is entered when an SWD debugger attaches to the device during the acquire window, which occurs during device reset. Debug mode allows the use of an IDE such as PSoC Creator or ARM MDK for firmware debugging. Debug mode is only available on devices in Open mode. For more details on the debug interface, see the Program and Debug Interface chapter on page 259.

For more details on protection modes, see the Device Security chapter on page 87.

11. Power Modes

The PSoC® 4 provides a number of power modes, intended at minimizing the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep-Sleep
- Hibernate
- Stop

Active, sleep, and deep-sleep are standard ARM defined power modes, supported by the ARM CPUs and instruction set architecture (ISA). Hibernate and stop modes are lower power consuming modes that are entered from firmware similar to deep-sleep, but on wakeup, the CPU and all peripherals go through a reset.

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

- Enabling/disabling clocks to peripherals
- Powering on/off internal regulators
- Powering on/off clock generators
- Powering on/off parts of the PSoC

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

Figure 11-1. Power Mode Transitions State Diagram

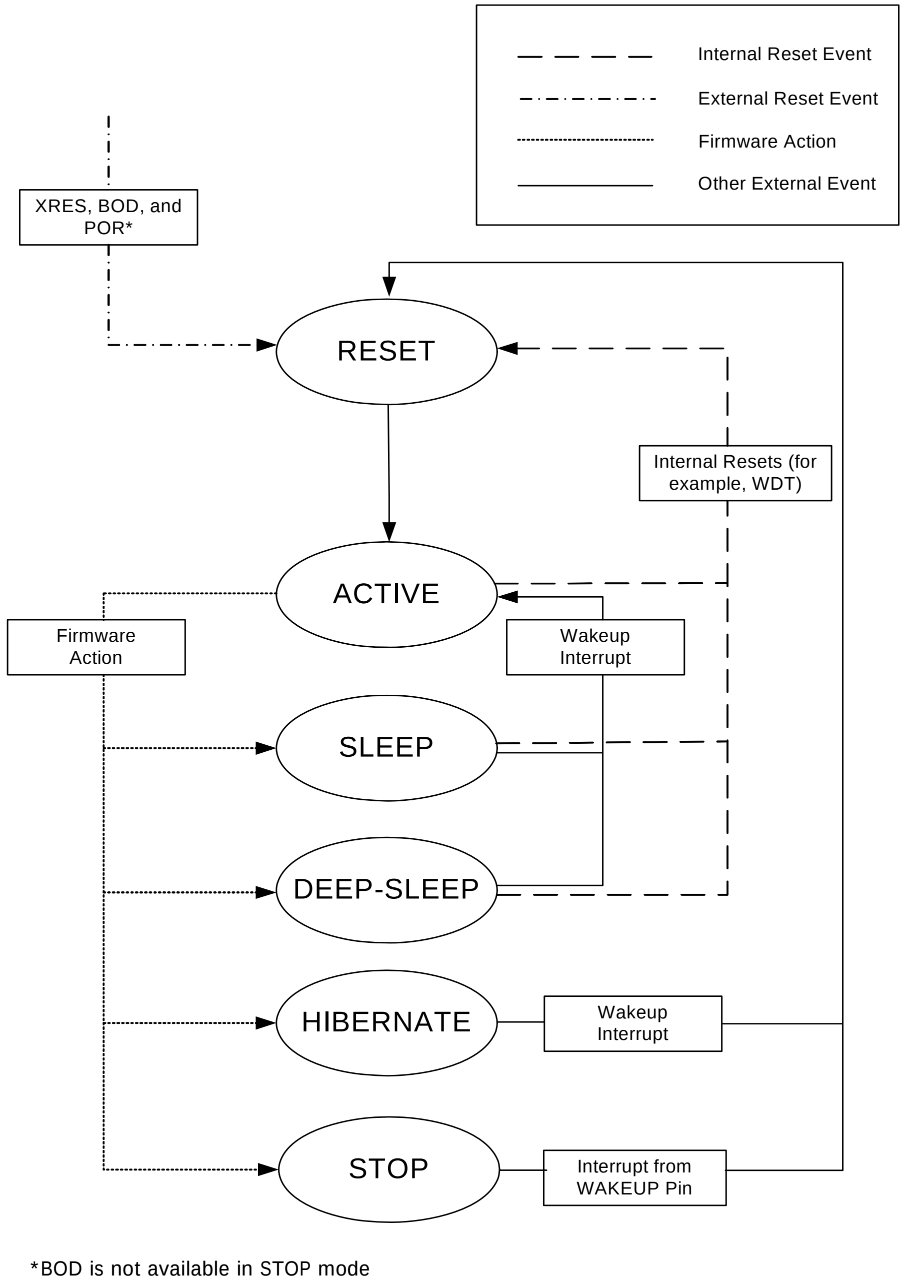


Table 11-1 illustrates the power modes offered by PSoC 4

Table 11-1. 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, power on reset	NA	Any (programmable)	Interrupt	All regulators are available. The active digital regulator can be disabled if external regulation is used.
Sleep	CPU enters sleep mode, SRAM is in retention, all peripherals are available (programmable)	Manual register entry	Any interrupt	Any (programmable)	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 Deep-Sleep regulator. IMO and high speed peripherals are off. Only the low-frequency (32 kHz) clock is available. Interrupts from low speed, asynchronous or low-power analog peripherals can cause a wakeup.	Manual register entry	PICU, low-power comparator, SCB, Watchdog timer	ILO (32 kHz)	Interrupt	Deep-sleep regulator and hibernate regulator
Hibernate	Only SRAM and UDBs are retained, most internal supplies are off. Wakeup is possible from a pin interrupt or a low-power comparator.	Manual register entry	PICU, low-power comparator	None	Reset	Hibernate regulator
Stop	All internal supplies are off. Only GPIO states are retained. Wakeup is possible from XRES or WAKEUP pins only.	Manual register entry	XRES, WAKEUP Pin	None	Reset	None

11.1 Active Mode

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

11.2 Sleep Mode

This is a CPU centric power mode. In this mode, the CPU indicates that it is in “sleep” mode and its main clock is disabled. It is a mode that the PSoC 4 must come to very often or as soon as there is nothing to do for the CPU, to accomplish low power consumption. It is identical to active mode from a peripheral point of view.

Any enabled interrupt can cause wakeup from sleep mode.

11.3 Deep-Sleep Mode

In deep-sleep mode, the CPU, SRAM, UDB, and high-speed logic are in retention.

The main system clock is off. Optionally, the internal low-frequency (32 kHz) oscillator remains on and low-frequency peripherals continue operating. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, I2C slave) continue to operate. Interrupts from low-speed, asynchronous or low-power analog peripherals can cause a wakeup from deep-sleep mode.

The available wakeup sources are listed in Table 11-3. The approximate current consumption in deep-sleep mode is 1.3 µA when the low-frequency clock is on and 0.5 µA when the low-frequency clock is off.

11.4 Hibernate Mode

This is the lowest PSoC 4 power mode that retains SRAM. It is implemented by switching off all clocks and removing power from the CPU and all peripherals, with the exception of a few (asynchronous) peripherals that can wake up the system from an external event. Note that in this mode, the CPU and all peripherals lose state.

In this mode, a hibernate regulator with limited capacity is used to achieve an extremely low power consumption. This puts a constraint on the maximum frequency of any signals present on the input pins while in hibernate mode. The combined toggle rate on all I/O pins (total frequency of signals in all inputs and outputs) must not exceed 10 kHz.

Any system that has signals toggling at high rates can use deep-sleep mode without seeing a significant difference in total power consumption.

Wakeup from hibernate mode is possible from a pin interrupt or a low-power comparator only. Wakeup from hibernate involves a reset rather than a wakeup from interrupt. When waking up from hibernate, the CPU and most peripherals are in their reset state and firmware will start at the reset vector. This reset tristates the I/Os, unless they are explicitly frozen by firmware. The interrupt status will still be available, allowing the system to identify the cause of wakeup.

External reset (XRES) triggers a full system restart. In this case, the cause is not readable after the device restarts. The current consumption in hibernate mode is approximately 150 nA.

11.5 Stop Mode

In the stop mode, the CPU, all internal regulators, and all peripherals are switched off. The GPIO output states are frozen in stop mode. (The configuration, mode, and state of all GPIOs in the system are locked. Changing the GPIO state is not possible until the device enters active mode again.) No other states are retained in this mode. Wakeup from stop mode is a system reset and it is possible from XRES or WAKEUP pins only. Note that the frozen GPIO states are lost when XRES is aserted. The current consumption in stop mode is approximately 20 nA.

Table 11-2 illustrates the available peripherals; Table 11-3 illustrates the available wakeup sources in each power mode.

Table 11-2. Available Peripherals

	Active	Sleep	Deep-Sleep	Hibernate	Stop
CPU	On	Retention[a]	Retention	Off	Off
SRAM	On	Retention	Retention	Retention	Off
High Speed Peripherals	On	On	Retention	Off	Off
Universal Digital Block (UDB)	On	On	Retention	Off	Off
Low Speed Peripherals	On	On	On (optional)	Off	Off
Internal Main Oscillator (IMO)	On	On	Off	Off	Off
Internal Low Speed Oscillator (ILO, 32kHz)	On	On	On (optional)	Off	Off
Asynchronous peripherals	On	On	On	Off	Off
Power On Reset, Brownout Detection	On	On	On	Off	Off
Regular Analog peripherals	On	On	On	Off	Off
Hibernate Analog Peripheral (LP Comparator)	On	On	On	On	Off
GPIO Output State	On	On	On/Frozen	Frozen[b]	Frozen

a. The configuration and state of the peripheral is retained. Peripheral continues its operation when the device enters active mode.
b. The configuration, mode, and state of all GPIOs in the system are locked. Changing the GPIO state is not possible until the device enters active mode.

Table 11-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any interrupt source	Interrupt
	Any reset source	Reset
Deep-Sleep	PICU (port interrupt)	Interrupt
	Low-power comparator	Interrupt
	SCB (I2C address match)	Interrupt
	Watchdog timer	Interrupt / Reset
	XRES (external reset pin)[a]	Reset
Hibernate	PICU (port Interrupt)	Reset
	Low-power comparator	Reset
	XRES (external reset pin)[a]	Reset
Stop	WAKEUP pin	Reset
	XRES (external reset pin)[a]	Reset

a. XRES triggers a full system restart. All the states including frozen GPIOs are lost. In this case, the cause of wakeup is not readable after the device restarts.

11.6 Enter and Exit Low-Power Modes

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

The transition to sleep, deep-sleep, and hibernate is controlled by the flags SLEEPDEEP in the CM0 System Control Register (SCR) and HIBERNATE in the System Resources Power subsystem (PWR_CONTROL).

- Sleep is entered when WFI instruction is executed, SLEEPDEEP = 0 and HIBERNATE = x.
- Deep-sleep is entered when WFI instruction is executed, SLEEPDEEP = 1 and HIBERNATE = 0.
- Hibernate is entered when WFI is executed, SLEEPDEEP = 1 and HIBERNATE = 1.

Use the PWR_STOP register to freeze the GPIO states in these low-power modes. This is recommended for the hibernate mode because the wakeup from hibernate mode causes a system reset.

In sleep and deep-sleep modes, a selection of peripherals are available (see Table 11-3), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from the low-power mode to the active mode. Additionally, any RESET returns the system to active mode.

Stop mode is entered directly using the PWR_STOP register in the System Resources Power subsystem. It removes power from all of the low-voltage logic in the system. Only the I/O state and PWR_STOP register contents are retained and wakeup (reset) happens on either XRES or toggling of a fixed WAKEUP pin.

The fields in PWR_STOP register are:

- TOKEN – This field contains an 8-bit token that is retained through a STOP/WAKEUP sequence that can be used by firmware to differentiate WAKEUP from a general RESET event. Note that waking up from STOP using XRES resets this register.
- UNLOCK – This register must be written to 0x3A to unlock stop mode. The hardware ignores the STOP bit if this field has any other setting.
- POLARITY – This bit sets the polarity of WAKEUP pin input. The device wakes up when the WAKEUP pin input matches the value of POLARITY bit.
- FREEZE – Setting this bit freezes the configuration, mode and state of all GPIOs in the system STOP –This bit must be set to enter the stop mode.

The recommended procedure to enter stop mode is:

- TOKEN = <any application-specified value>
- UNLOCK = 0x3A
- POLARITY = <application-specified polarity>
- FREEZE = 1
- STOP = 1

It is recommended to add two NOP cycles after the third write. Stop mode exits when either the XRES or WAKEUP pins are toggled. Both events clear the STOP bit in the PWR_STOP register and trigger a POR. A wakeup event does not clear the other bits of the PWR_STOP register.

An XRES event clears all the bits. The recommended firmware procedure on wakeup is as follows:

- Optionally read TOKEN for application-specific branching.
- Optionally write I/O drive modes and output data registers to the required settings. A typical procedure for digital output ports is to set the pad as output, read its frozen value, and set that value in the output data register.
- Unfreeze the I/O.

11.7 Register List

Table 11-4. Register List

Register Name	Description
SCR	System Control Register - Sets or returns system control data.
PWR_CONTROL	Power Mode control - Controls the device power mode options and allows observation of current state.
PWR_STOP	This register controls entry/exit from the Stop power mode.

12. Watchdog Timer

The watchdog timer (WDT) circuit automatically resets the microcontroller in the event of an unexpected firmware execution path. This timer, which is clocked by the 32-kHz ILO, must be serviced periodically in firmware to avoid a reset. Otherwise, the microcontroller resets after a specified period of time. The WDT can also be used as wakeup source in low-power modes.

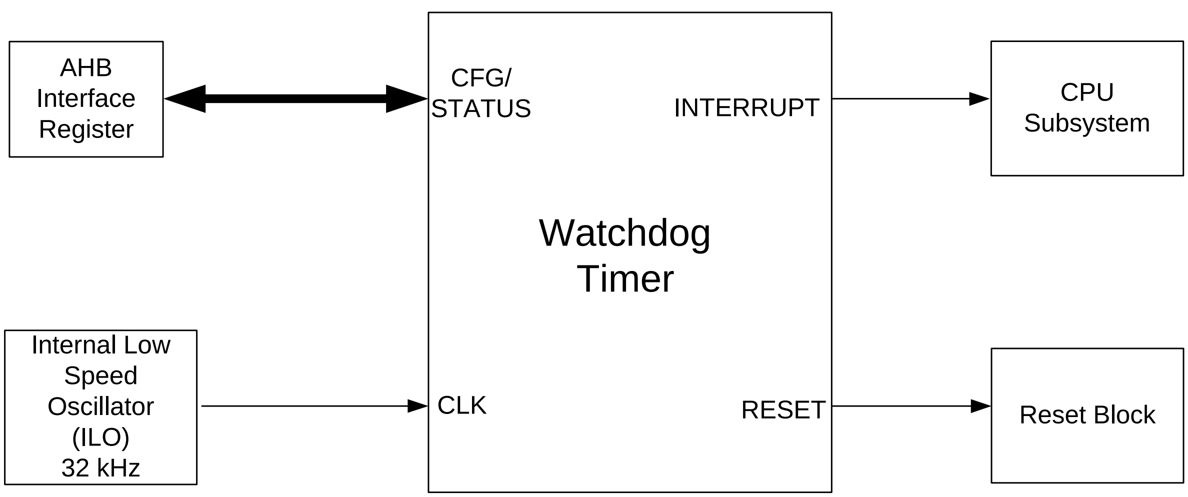
12.1 Features

The WDT has these features:

- Configurable timer period (Independent/Cascaded counters)
- Can generate an interrupt in Sleep or Deep-Sleep power mode to wake up the microcontroller
- Can generate an interrupt in Active mode after a specified interval
- Protection settings to prevent accidental corruption of the registers

12.2 Block Diagram

Figure 12-1. Watchdog Timer Block Diagram



12.3 How It Works

The WDT asserts an interrupt or a hardware reset to the device after a preprogrammed interval, unless it is periodically serviced in firmware. In PSoC 4, the WDT has two 16-bit counters (Counter-0 and Counter-1) and one 32-bit counter (Counter-2). These counters can be configured to work independently or in cascade. The cascade configuration provides an option to increase the reset or interrupt interval.

Counter-0 and Counter-1 generate an interrupt or a reset on reaching the specified terminal count for the first time, or generate a reset after three continuous unhandled interrupt, whereas Counter-2 only generates an interrupt based on the value stored in the WDT_BITS2[4:0] register bits.

12.3.1 Enabling and Disabling WDT

The WDT counters are enabled by setting the WDT_ENABLE bit and are disabled by clearing it. Enabling or disabling a WDT requires three LF clock cycles to come into effect. Therefore, the WDT_ENABLE bit value must not be changed more than once in that period.

After WDT is enabled, it is illegal to write WDT configuration (WDT_CONFIG) and control (WDT_CONTROL) registers. Accidental corruption of WDT registers can be prevented by setting the bit-field WDT_LOCK of CLK_SELECT register. If the application requires updating the terminal count value (WDT_MATCH) when the WDT is running, clear the bit-field WDT_LOCK.

12.3.2 WDT Operating Modes

The Counter-0 or Counter-1 can be used to generate a reset to avoid the system going into the unresponsive state or to generate an interrupt to wake up the system from Sleep or Deep-Sleep mode. The register bits WDT_MODEX[1:0] are configured to select the required match action when the count value stored in the register bits WDT_CTRX equals the preprogrammed terminal count value stored in the register bits WDT_MATCHX, where X is either 0 or 1.

Table 12-1. Counter-0 and Counter-1 Modes

Bit-field Name	Description
WDT_MODE0[1:0] or WDT_MODE1[1:0]	Watchdog Counter Action on Match (WDT_CTR0=WDT_MATCH0) or (WDT_CTR1=WDT_MATCH1): 00: Do nothing 01: Assert WDT_INT0 or WDT_INT1 10: Assert WDT Reset 11: Assert WDT_INT0 or WDT_INT1, assert WDT reset after the third unhandled interrupt

The Counter-2 can be used to generate the interrupt based on status of the WDT_BITS2[4:0] register bits.

Table 12-2. Counter-2 Modes

Bit-field Name	Description
WDT_MODE2	0: Free running counter with no interrupt requests 1: Free running counter with interrupt request when a specified bit in counter-2 toggles (see Table 12-4).

12.3.3 WDT Interrupts and Low-Power Modes

The WDT counter sends the interrupt requests to the CPU in Active mode and to the WakeUp Interrupt Controller (WIC) in Sleep or Deep-Sleep mode. It works as follows:

- **Active Mode:** The interrupt request from the WDT is sent to the CPU. The CPU acknowledges the interrupt request and executes the Interrupt Service Routine (ISR). The interrupt must be cleared after entering the ISR in firmware.
- **Sleep or Deep-Sleep Power Mode:** In this mode, the CPU subsystem is powered-down. So, 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.

12.3.4 WDT Reset Mode

The RESET_WDT bit is set in the RESET_CAUSE register to indicate the reset generated by the WDT. This bit remains set until cleared or until a power-on reset (POR) or brownout reset (BOD) occurs; for example in the case of device power cycle. All other resets leave this bit untouched.

For more details, see the Reset System chapter on page 85.

12.4 Register List

Table 12-3. Control and Status Register Bits for Counter-0 and Counter-1

Bit-field Name	Description
WDT_CLEARX	Clear Watchdog Counter when WDT_CTRX=WDT_MATCHX. 0: Free running counter. The counter restarts from 0 after reaching 0xFFFF. 1: Clear on match. The counter restarts from 0 after matching.
WDT_ENABLEX	Enable Counter-X 0: Counter is disabled (not counting) 1: Counter is enabled (counting up)
WDT_INTX	WDT Interrupt Request. This bit is set by hardware. This bit must be cleared by firmware. Clearing this bit also prevents Reset from happening when WDT_MODEX=3.
WDT_RESETX	Reset counter-X count to 0. Hardware resets this bit after the counter is reset.
WDT_CASCADE0_1	Cascade Watchdog Counters 0 and 1. The count value of the Counter-1 increments after the Counter-0 reaches terminal count (WDT_CTR0=WDT_MATCH0). 0: Independent counters 1: Cascaded counters
WDT_MATCH0	Match value for Watchdog Counter-0
WDT_MATCH1	Match value for Watchdog Counter-1

Note X = 0 or 1

Table 12-4. Control and Status Register Bits for Counter-2

Bit-field Name	Description
WDT_CASCADE1_2	Cascade Watchdog Counters 1 and 2. The count value of the Counter-2 increments after the Counter-1 reaches terminal count (WDT_CTR1=WDT_MATCH1). You can cascade all 3 counters using WDT_CASCADE0_1 and WDT_CASCADE1_2. 0: Independent counters 1: Cascaded counters
WDT_ENABLE2	Enable Counter-2 0: Counter is disabled (not clocked) 1: Counter is enabled (counting up)
WDT_INT2	WDT Interrupt Request. This bit is set by hardware. This bit must be cleared by the firmware.
WDT_RESET2	Resets counter-2 back to 0. Hardware resets this bit after counter is reset.
WDT_BITS2[4:0]	The value stored in WDT_BITS2[4:0] decides the WDT_INT2 interrupt occurrence interval–one interrupt per 2^(WDT_BITS[4:0])) clocks 0: Assert WDT_INT2 when bit 0 of Counter-2 toggles (one interrupt per 2^0 clock) 1: Assert WDT_INT2 when bit 1 of Counter-2 toggles (one interrupt per 2^1 clocks) ... 31: Assert WDT_INT2 when bit 31 of Counter-2 toggles (one interrupt per 2^31 clocks)

13. Reset System

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.

The reset system has these sources:

- Power-on reset (POR) to hold the part in reset while the power supply ramps up
- Brownout reset (BOD) to reset the part if the power supply falls below specifications during operation
- Watchdog reset (WRES) to reset the part if firmware execution fails to service the watchdog timer
- Software initiated reset (SRES) to reset the part on demand using firmware
- External reset (XRES) to reset the part using an electrical signal external to the PSoC 4
- Protection fault reset (PROT_FAULT) to reset the part if unauthorized operating conditions occur
- Hibernate wakeup reset to bring the part out of the hibernate low-power mode
- Stop wakeup reset to bring the part out of the stop low-power mode

13.1 Reset Sources

The following sections provide a description of the reset sources.

13.1.1 Power-on Reset

Power-on reset is provided for system reset at power-up. POR holds the device in reset until all three voltages: V_{DDA}, V_{DDD}, and V_{CCD}, are according to 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, undetectable BOD, or XRES.

13.1.2 Brownout Reset

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

BOD events do not set a reset cause status bit, but in some cases they can be detected. In some BOD events, V_{CCD} will fall below the minimum logic operating voltage, but remain above the minimum logic retention voltage. Thus, some BOD events may be distinguished from POR events by checking for logic retention. This is explained further in the Identifying Reset Sources on page 86 section.

13.1.3 Watchdog Reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. This feature is enabled by setting the WDT_CONTROL[0] register bit.

The RES_CAUSE[0] status bit is set when a watchdog reset occurs. This bit remains set until cleared or until a POR 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 81

13.1.4 Software Initiated Reset

Software initiated reset (SRES) is a mechanism that allows a software-driven reset. The Cortex-M0 application interrupt and reset control register AIRCR forces a device reset when a ‘1’ is written into bit 2.

The RES_CAUSE[4] status bit is set when a software reset occurs. This bit remains set until cleared or until a POR or BOD reset, for example in the case of a device power cycle. All other resets leave this bit untouched.

13.1.5 External Reset

External reset (XRES) is a user-supplied reset that causes immediate system reset when asserted. The XRES_N pin is ***active low*** – a high voltage on the pin causes no behavior and a low voltage causes a reset. XRES_N is available as a dedicated pin on all devices.

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

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

13.1.6 Protection Fault Reset

Protection fault reset (PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code.

The RES_CAUSE[3] bit is set when a protection fault occurs. This bit remains set until cleared or until a POR or BOD reset – for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.1.7 Hibernate Wakeup Reset

Hibernate wakeup reset detects hibernate wakeup sources and performs a device reset to return to the active power mode. Hibernate wakeup resets are caused by interrupts. Both pin and comparator interrupts are available in the hibernate low-power mode. After a hibernate wakeup reset, both SRAM and UDB register contents are retained, but code execution begins after reset as it does after any other reset source occurs.

Hibernate resets can be detected by checking the interrupt registers for comparators and pins. These interrupt register states will be retained across resets.

For more details, see Hibernate Mode on page 78.

13.1.8 Stop Wakeup Reset

Stop wakeup reset detects stop wakeup sources and performs a device reset to return to the active power mode. Stop wakeup resets are caused by the XRES pin or the WAKEUP pin. After a stop wakeup reset, no memory contents are retained; code execution begins after reset as it does after any other reset source occurs.

Some stop wakeup resets can be detected by examining the TOKEN bit-field (bits 0:7) in the PWR_STOP register. This bit-field will be filled with a key when stop mode is entered. Its contents will be retained if the part is woken up using the WAKEUP pin. If the part is woken up with the XRES pin, the wakeup source cannot be detected. For more details, see Stop Mode on page 78.

13.2 Identifying Reset Sources

When the device comes out of reset, it is often useful to know the cause of the most recent or even older resets. This is achieved in the device primarily through the RES_CAUSE register. This register has specific status bits allocated for some of the reset sources. The RES_CAUSE register supports detection of watchdog reset, software reset, and protection fault reset. It does not record the occurrences of POR, BOD, XRES, or the Hibernate and Stop wakeup resets. The bits are set on the occurrence of the corresponding reset, and remain set after the reset, until cleared or a loss of retention, such as a POR reset or brownout below the logic retention voltage. Hibernate wakeup resets can be detected by examining the comparator and pin interrupt registers that were configured to wake the part up from hibernate mode. Stop wakeup resets that occur as a result of a WAKEUP pin event can be detected by examining the PWR_STOP register, as described previously. Stop wakeup resets that occur as a result of an XRES cannot be detected.

The other reset sources can be inferred to some extent by the status of RES_CAUSE. Brownout events can be subdivided into two categories: retention resets and non-retention resets. If V_{CCD} dips below the minimum logic operating voltage, but not below the minimum logic retention voltage, then a BOD reset occurs; but retention of registers is maintained. If V_{CCD} dips below both minimum operating and minimum retention voltage, then a BOD reset occurs without retention of registers. This register retention can be detected using a special register, PWR_BOD_KEY. The PWR_BOD_KEY register only changes value when written by firmware or when a non-retention reset such as a non-retention BOD, XRES, or POR event. This register may be initialized by firmware, and then checked in subsequent executions of startup code to determine if a retention BOD occurred.

If these methods cannot detect the cause of the reset, then it can be one of the non-recorded and non-retention resets: non-retention BOD, POR, XRES, or XRES Stop Wakeup reset. These four cannot be distinguished using on-chip resources.

14. Device Security

PSoC® 4 offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and robust flash protection provide a high level of security. Additional security can be gained by implementing custom functionality in the universal digital blocks (UDBs) instead of in firmware.

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 then cannot access the device.

Note Because all programming, debug, and test interfaces are disabled when maximum device security is enabled, PSoC 4 devices with full device security enabled may not be returned for failure analysis.

14.1 Features

The PSoC 4 device security system has the following features:

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

14.2 How It Works

The CPU operates in normal user mode or in privileged mode, and the device operates in one of four protection modes: BOOT, OPEN, PROTECTED, and KILL. Each mode provides specific capabilities for the CPU software and debug (through the DAP):

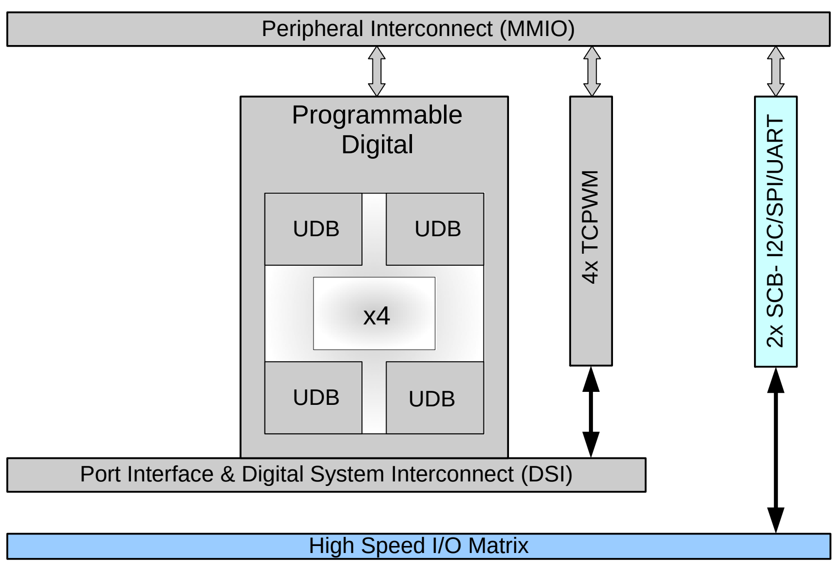
- **BOOT mode**: The device comes out of reset in BOOT mode. It stays there until its protection state is copied from supervisor flash to the protection control register (CPUSS_PROTECTION). The debug-access port is stalled until this has happened. BOOT is a transitory mode required to set the part to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

Section E: Digital System

This section encompasses the following chapters:

Top Level Architecture

Digital System Block Diagram



15. Serial Communications (SCB)

The Serial Communications Block (SCB) of PSoC® 4 supports three serial interface protocols: SPI, UART, and I2C. Only one of the protocols is supported by an SCB at any given time. PSoC 4 devices have two SCBs. Additional instances of the SPI and UART protocols can be implemented using Universal Digital Blocks (UDBs) and PSoC Creator.

15.1 Features

This block supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, Local Interconnect Network, and IrDA protocols
- Standard I2C master and slave functionality
- SPI and I2C EZ mode, which allows for operation without CPU intervention
- Low-power (Deep-Sleep) mode of operation for SPI and I2C protocols (using external clocking)

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

15.2 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) protocol is a synchronous serial interface protocol. Devices operate in either master or slave mode. The master initiates the data transfer. The SCB supports Single Master-Multiple Slaves topology for SPI. Multiple slaves are supported with individual slave select lines.

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

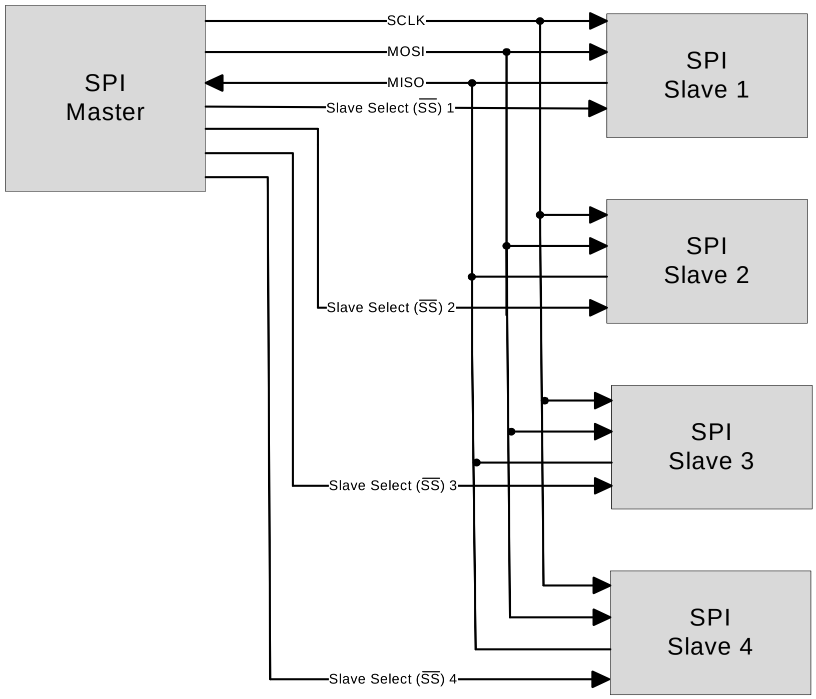
15.2.1 Features

- Supports master and slave functionality
- Supports 3 types of SPI protocols:
  - Motorola SPI – modes 0, 1, 2, and 3
  - TI SPI, with coinciding and preceding data frame indicator for mode 1
  - National (MicroWire) SPI for mode 0
- 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 and Easy I2C (EZI2C) Protocol)
- Supports externally clocked slave operation:
  - In this mode, the slave operates in Active, Sleep, and Deep-Sleep system power modes
  - EZSPI mode allows for operation without CPU intervention

15.2.2 General Description

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

Figure 15-1. SPI Example



A standard SPI interface consists of four signals as follows.

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

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

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

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

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

15.2.3 SPI Modes of Operation

15.2.3.1 Motorola SPI

The original SPI protocol was defined by Motorola. It is a full duplex protocol. Multiple data transfers may happen with the 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 SS line is held at '1' and SCLK is typically off.

Modes of Motorola SPI

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

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

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

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

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

Figure 15-2. SPI Motorola, 4 Modes

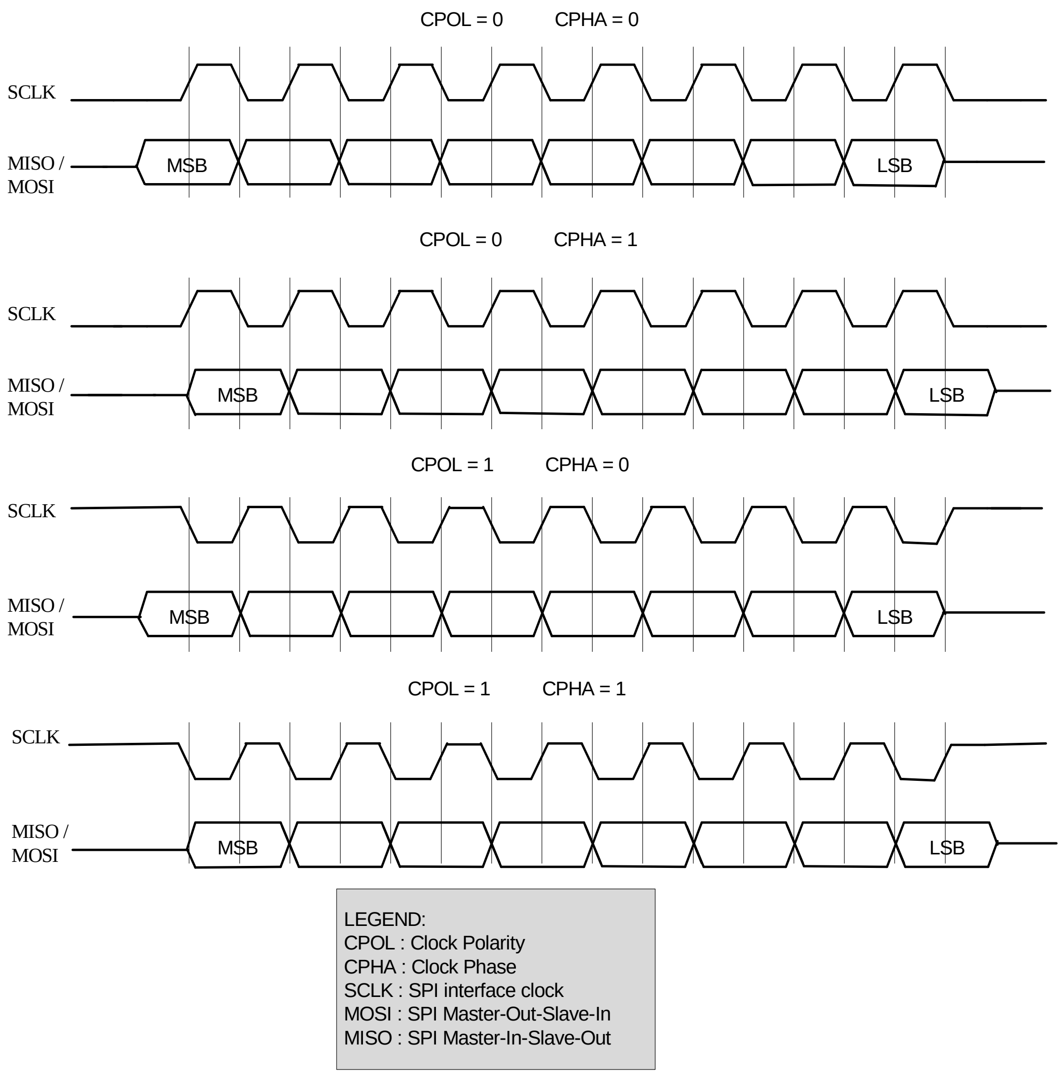
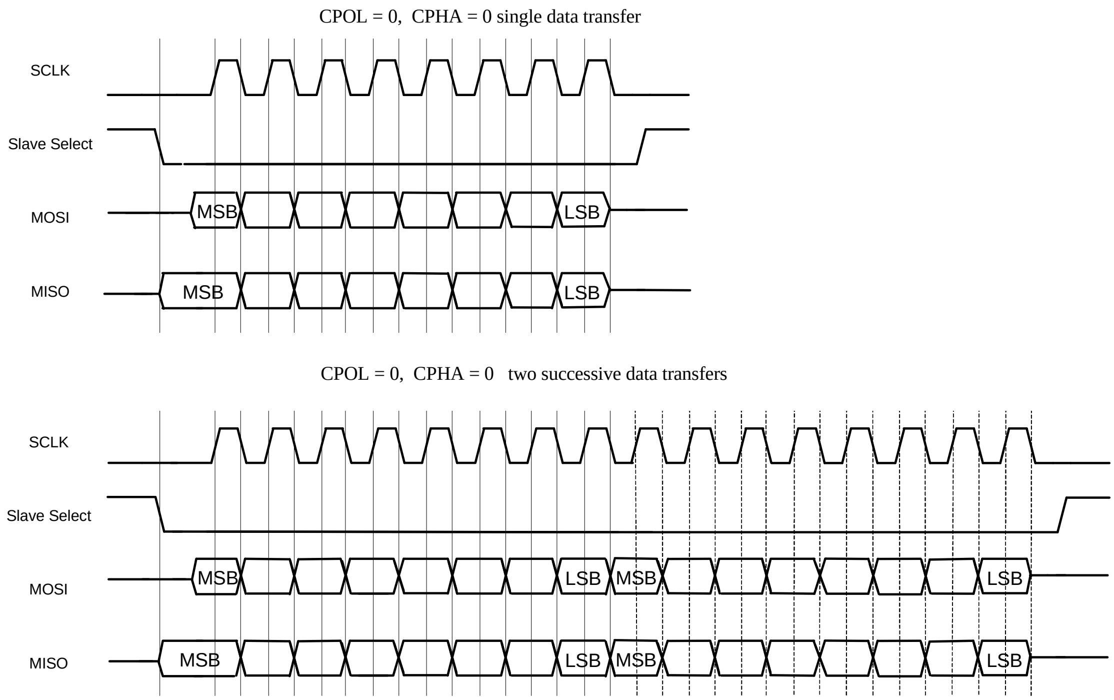


Figure 15-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 15-3. SPI Motorola Data Transfer Example



LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out

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 SCB_MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI Motorola mode by writing '00' to the SCB_MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the SCB_CPHA and SCB_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 101.

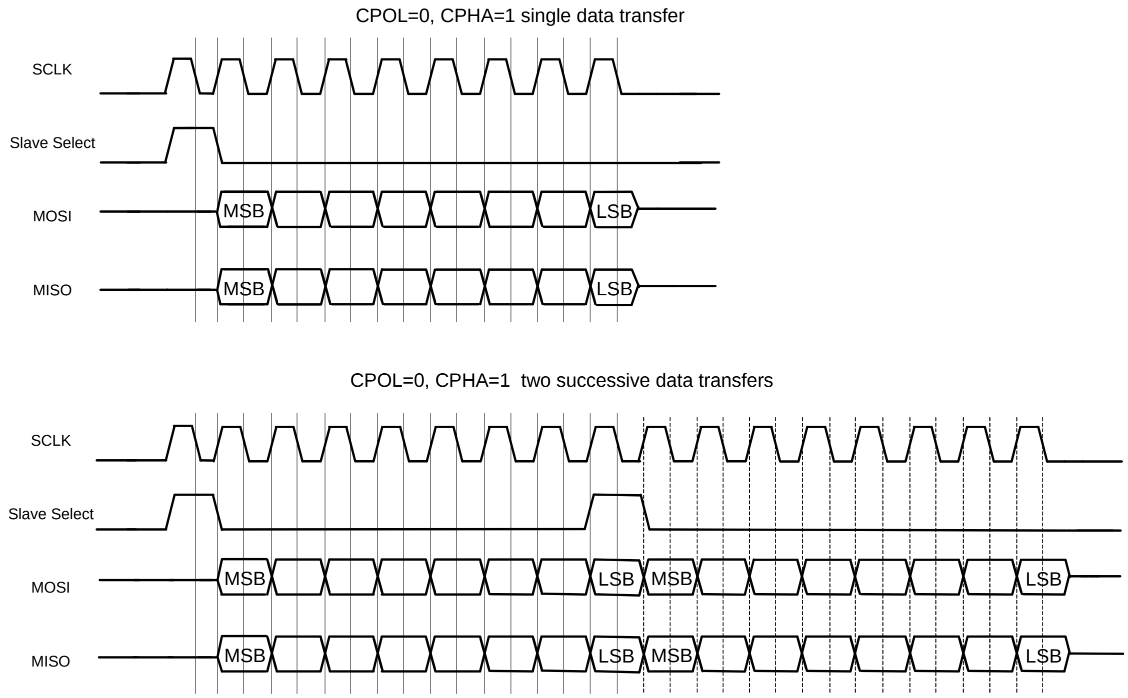
Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4 Registers TRM.

15.2.3.2 *Texas Instruments SPI*

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

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

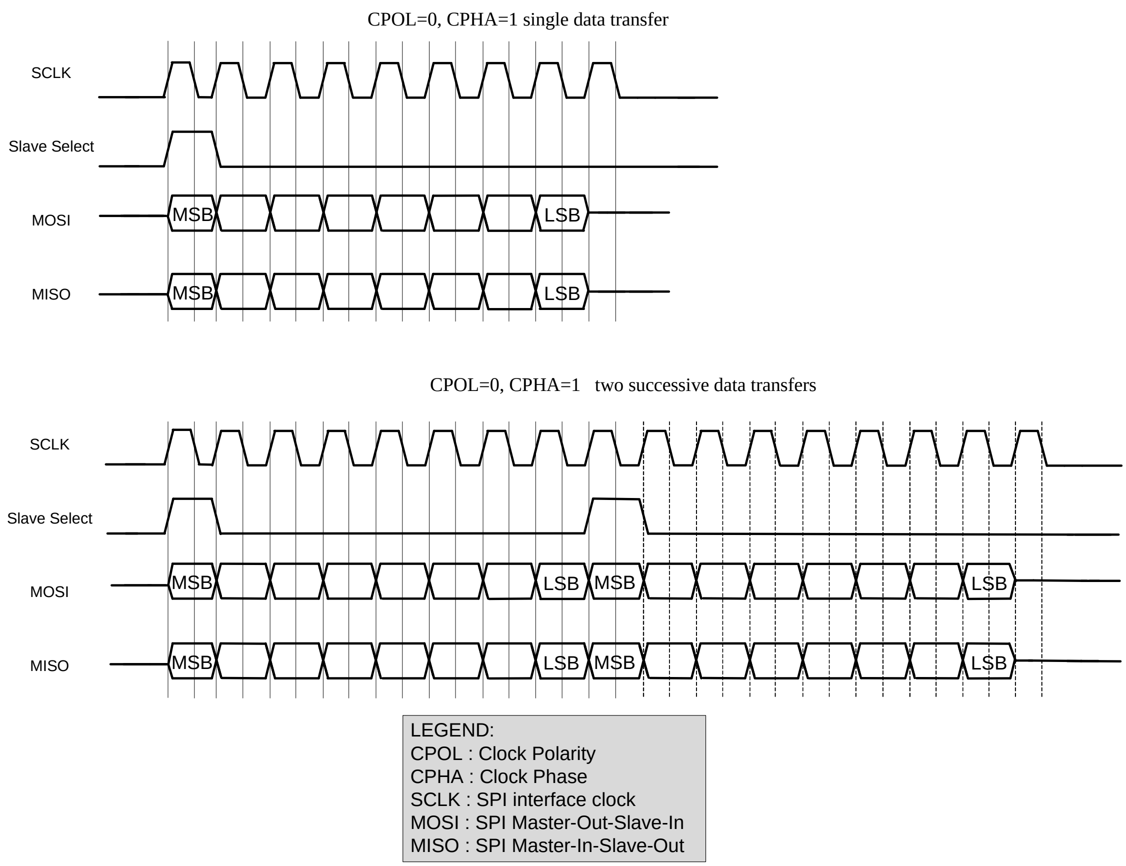
Figure 15-4. SPI TI Data Transfer Example



LEGEND:
CPOL : Clock Polarity
CPHA : Clock Phase
SCLK : SPI interface clock
MOSI : SPI Master-Out-Slave-In
MISO : SPI Master-In-Slave-Out

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

Figure 15-5. SPI TI Data Transfer Example



Configuring the 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 SCB_MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI TI mode by writing '01' to the SCB_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 4 mentioned in Enabling and Initializing SPI on page 101.

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

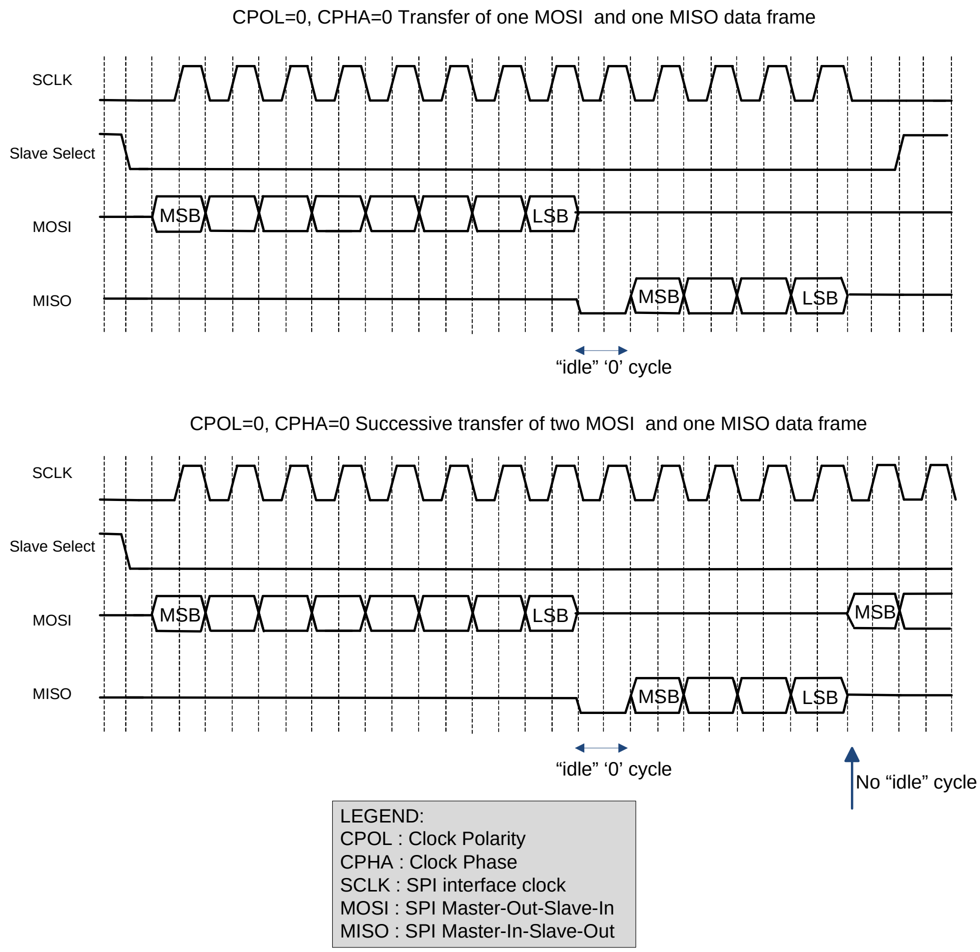
15.2.3.3 National Semiconductors SPI

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

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

Figure 15-6 illustrates a single data transfer and two successive data transfers. In both cases the transmission data transfer size is 8 bits and the reception data transfer size is 4 bits.

Figure 15-6. SPI NS Data Transfer Example



Configuring the 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 SCB_MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI NS mode by writing '10' to the SCB_MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Follow steps 2 to 4 mentioned in Enabling and Initializing SPI on page 101.

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

15.2.4 Easy SPI (EZSPI) Protocol

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

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

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

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

15.2.4.1 EZ Address Write

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

15.2.4.2 Memory Array Write

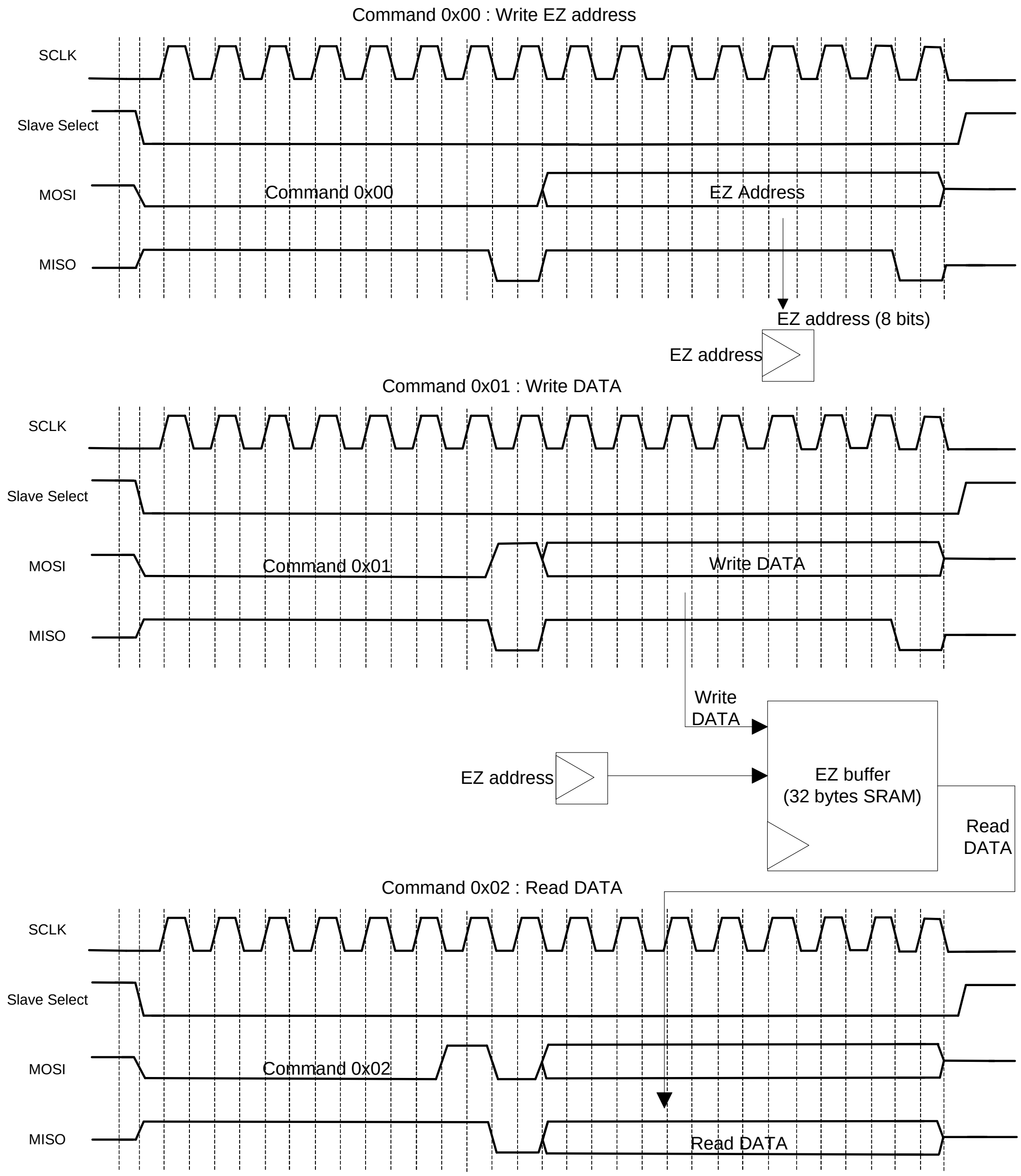
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was observed (0xFE) or not (0xFF). Any additional write data bytes on the MOSI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

15.2.4.3 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 observed (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 wraps around to 0.

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

Figure 15-7. EZSPI Example



LEGEND :

CPOL : Clock Polarity	0x00 : Write EZ address
CPHA : Clock Phase	0x01 : Write DATA
SCLK : SPI Interface Clock	0x02 : Read DATA
MISO : SPI Master-In-Slave-Out	0xFE : “slave ready”
MOSI : SPI Master-Out-Slave-In	0xFF : “slave busy”

15.2.4.4 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 various 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. Follow steps 2 to 4 mentioned in Enabling and Initializing SPI on page 101.
3. Use continuous transmission mode for transmitter by writing '1' to the SCB_CONTINUOUS bit of SCB_SPI_CTRL register.
4. EZSPI mode is applicable only for slave functionality (write '0' to the SCB_MASTER_MODE field, bit 31 of SCB_SPI_CTRL register).
5. Set the data frame width eight bits long (write ‘0111’ to the SCB_DATA_WIDTH field, bits [3:0] of SCB_TX_CTRL and SCB_RX_CTRL registers).
6. Set the shift direction as MSB first (write '1' to the SCB_MSB_FIRST field, bit 8 of SCB_TX_CTRL and SCB_RX_CTRL registers).

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

15.2.5 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 15-1. For more information on these registers, see the PSoC 4 Registers TRM.

Table 15-1. SPI Registers

Register Name	Operation
SCB_CTRL	Used to enable the SCB, select the type of serial interface (SPI, UART, I2C), and for selecting 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	Used to configure the SPI as either a master or a slave, selecting SPI protocols (Motorola, TI, National), clock-based submodes in Motorola SPI (modes 0,1,2,3), selecting the type of SELECT signal in TI SPI.
SCB_SPI_STATUS	Indicates whether the SPI bus is busy. It is also used to set the SPI slave EZ address in the internally clocked mode.
SCB_TX_CTRL	Used to enable the transmitter, also 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.
SCB_TX_FIFO_CTRL	Used to specify the trigger level, clear the transmitter FIFO and shift registers, and for 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 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_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_EZ_DATA	Holds the data in EZ memory location

15.2.6 SPI Interrupts

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

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

The SPI supports interrupts on the following events:

- SPI transfer done
- SPI is Idle
- TX FIFO is not full
- TX FIFO is empty
- SPI Byte / Word transfer complete
- RX FIFO is empty
- RX FIFO is not empty
- Attempt to write to a full RX FIFO.
- RX FIFO is Full

15.2.7 Enabling and Initializing SPI

The SPI must be programmed in the following order:

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

Table 15-2. SCB_SPI_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	SPI Motorola submode
		01	SPI Texas Instruments submode
		10	SPI National Semiconductors submode
		11	Reserved
31	MASTER_MODE	0	Master mode
		1	Slave mode

Table 15-3. 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
31	ENABLED	Transmitter enable bit for SCB_TX_CTRL and Receiver enable bit for SCB_RX_CTRL registers. They must be enabled for all the protocols. Otherwise, the block may not function or the data may get lost.

Table 15-4. SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[2:0]	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.

Table 15-5. SCB_CTRL Register

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

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 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).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register).

15.2.8 Internally and Externally Clocked SPI Operations

The SCB supports both internally and externally clocked operations for SPI and I2C 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, in which a no-chip internal clock is provided to the block.

Internally clocked operation uses the high-frequency clock of the system. For more information on system clocking, see the Clocking System chapter on page 61. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The SCB_OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

In SPI Master mode, the valid range for oversampling is 4 to 16. Hence, the maximum bit rate is 12 Mbps. However, if you consider the I/O cell and routing delays, the effective oversampling range becomes 6 to 16. So, the maximum bit rate is 8 Mbps. **Note** LATE_MISO_SAMPLE must be set to '1' in SPIM mode.

In SPI Slave mode, the oversampling field (bits [3:0]) of SCB_CTRL register is not used. However, there is a frequency requirement for the SCB clock with respect to the interface clock (SCLK). This requirement is expressed in terms of the ratio (SCB Clock / SCLK). This ratio is dependent on two fields: MEDIAN of SCB_RX_CTRL register and LATE_MISO_SAMPLE of SCB_CTRL register. With the MEDIAN bit set to ‘0’ and LATE_MISO_SAMPLE bit set to ‘1’, the SCB can achieve a maximum bit rate of 16 Mbps. However, if you consider the I/O cell and routing delays, the maximum data rate that can be achieved becomes 8 Mbps. Based on these bits, the maximum bit rates are given in Table 15-6.

Table 15-6. SPI Slave Maximum Data Rates

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

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 (in the case of SPI slave functionality).

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

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

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

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

15.2.8.1 Non EZ Mode of Operation

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

Table 15-7. SPI Non-EZ Mode

SPI, standard (non-EZ) mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
System Power Mode	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**	**EC_AM_MODE = 1**	**EC_AM_MODE=0**
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in the Sleep mode, the MASK bit can be configured by the user. After that, selection using internal clock. Operation using internal clock.	Not supported	Invalid configuration
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause in enabled (MASK = 1). Generate 0xff bytes	Not supported	
Hibernate	The SCB is not available in these modes (see Power Modes chapter on page 75)			
Stop				

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

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power mode and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active system power mode both 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 slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in the Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic 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' bits or "0xFF" bytes are send out on the MISO line) and the internally clocked logic takes care of the next SPI transfer when it is woken up.

15.2.8.2 EZ Mode of Operation

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

Table 15-8. SPI EZ Mode

SPI, EZ mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
System Power Mode	**EC_AM_MODE = 0**	**EC_AM_MODE = 1**	**EC_AM_MODE = 1**	**EC_AM_MODE=0**
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in Sleep mode, the mask bit can be configured by the user. After that, selection using internal clock. Operation using internal clock.	Selection using external clock. Operation using external clock.	Invalid configuration
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Generate 0xff bytes.	Selection using external clock Operation using external clock	
Hibernate	The SCB is not available in these modes (refer the chapter on Power modes)			
Stop				

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active system power mode. 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 system power mode and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active system power mode both 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 slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic 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' bits or "0xFF" bytes are send out on the MISO line) and the internally clocked logic 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 system power mode and Deep-Sleep system power mode. The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK of the SCB_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

15.3 UART

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

- TX: Transmitter output
- RX: Receiver input

15.3.1 Features

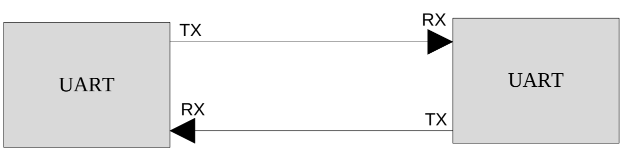
- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 1 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports Local Interconnect Network (LIN)
  - Break detection

- ❐ Baud rate detection
- ❐ Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus).
- Multi-processor mode
- Data frame size programmable from 4 bits to 16 bits.
- Programmable number of STOP bits, which can be sret to 1, 1.5, or 2 data bits
- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

15.3.2 General Description

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

Figure 15-8. UART Example



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

Three different serial interface protocols are supported:

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

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop and parity bits. The number of stop bits can be in the range of 1 to 3. The parity bit can be either enabled or disabled. If enabled, the type of parity can be set to either even parity or odd parity. The option of using the parity bit is available only in the Standard UART and SmartCard UART modes. For IrDA UART mode, the parity bit is automatically disabled. Figure 15-9 depicts the default configuration of the UART interface of the SCB.

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

15.3.3 UART Modes of Operation

15.3.3.1 Standard Protocol

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

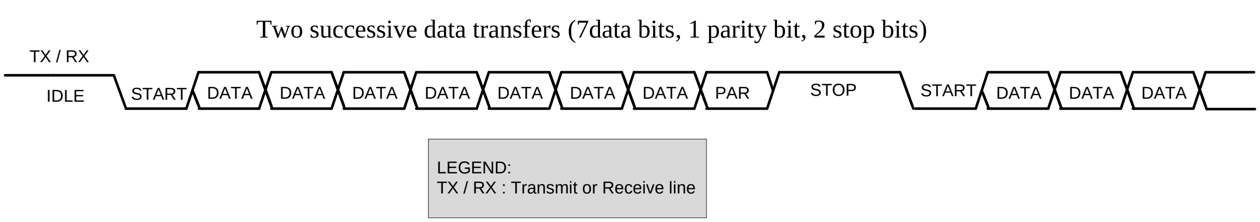
Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample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

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

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

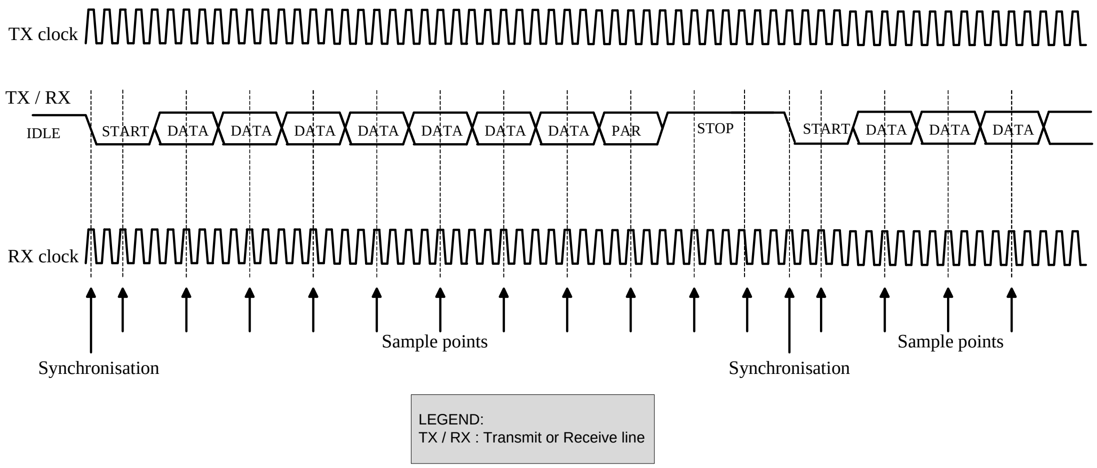
Figure 15-9 illustrates the UART protocol.

Figure 15-9. UART, Standard Protocol Example



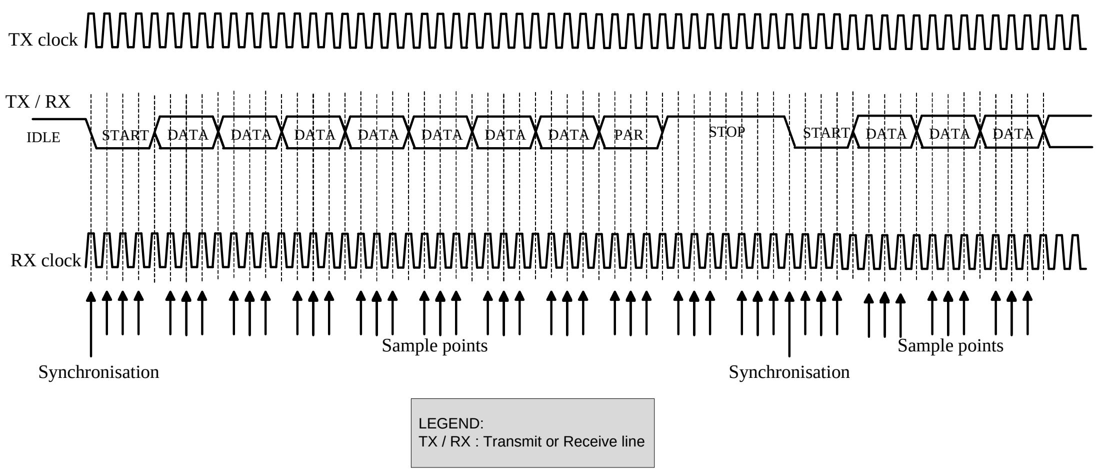
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. Figure 15-10 illustrates this.

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



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

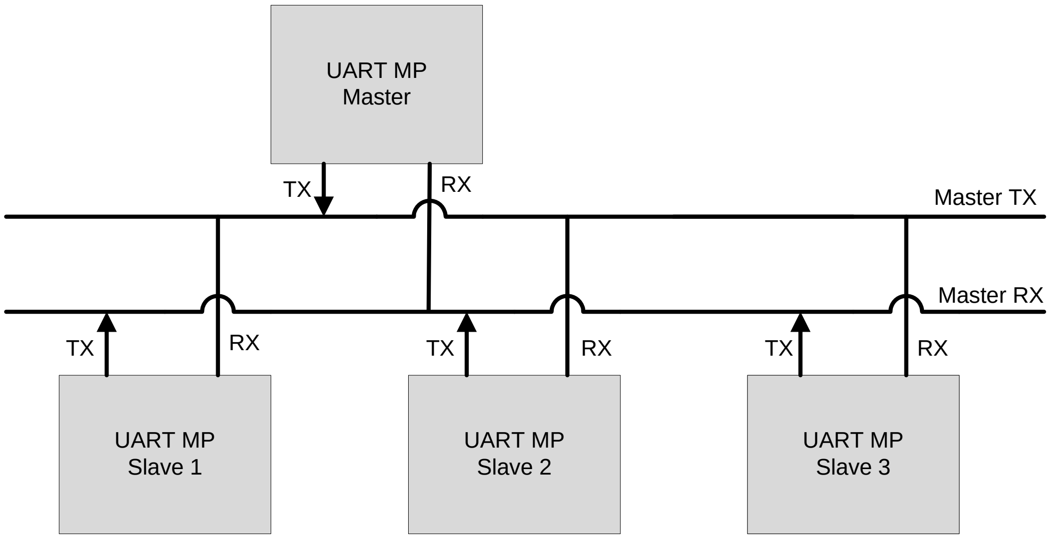
Figure 15-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor Mode

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

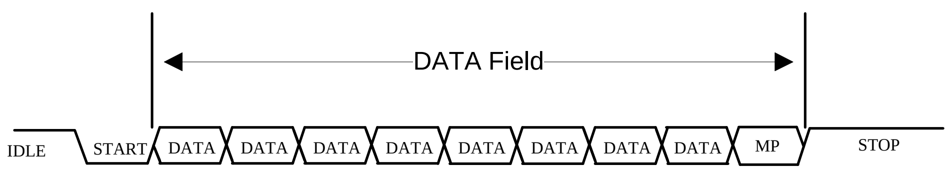
Figure 15-12. UART MP Mode Bus Connections



The main properties of UART_MP mode are:

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

Figure 15-13. UART MP Data Frame

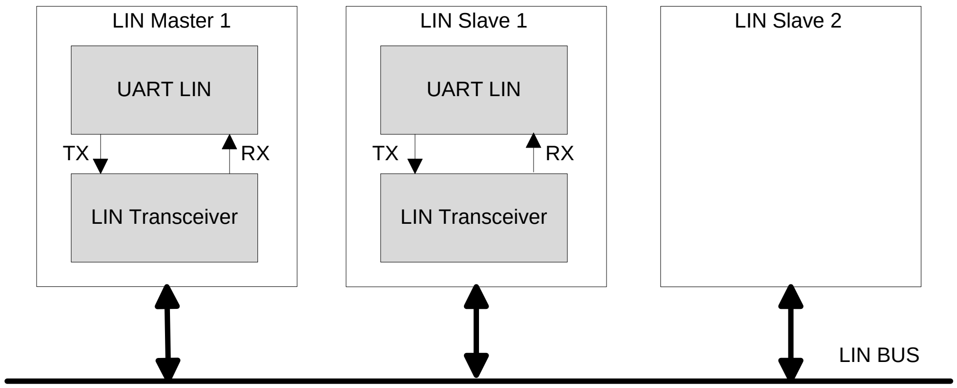


The SCB can be used as either master or slave device in UART_MP mode. Both SCB_TX_CTRL and SCB_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device, the MP_MODE field of the SCB_UART_RX_CTRL register should be set to '1'. The SCB_RX_MATCH register should be set for the slave address and address mask. The matched address is written in the RX_FIFO when SCB_ADDRESS_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 Local Interconnect Network (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. Figure 15-14 illustrates the UART_LIN and LIN Transceiver.

Figure 15-14. UART_LIN and LIN Transceiver

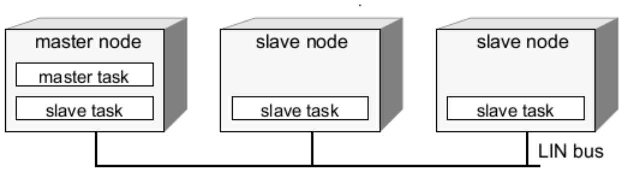


LIN protocol defines two tasks:

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

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

Figure 15-15. LIN Bus Nodes and Tasks



LIN is based on the transmission of frames at pre-determined moments of time. A frame is divided into header and response fields.

- 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.

The UART LIN of SCB supports slave task, receiving the header and transmitting the response. It provides baud rate detection (using sync field - 0x55) operation. Apart from the break field, a frame transmission (both header and response) consist of one or multiple byte frame transmissions, with each byte transmission consisting of a start bit, 8 data bits and 1 or more stop bits (on both the UART TX and RX lines).

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 SCB_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 SCB_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 SCB_MP_MODE (bit 11) or SCB_LIN_MODE (bit 12) respectively of the SCB_UART_RX_CTRL register.
4. Follow steps 2 to 4 described in Enabling and Initializing UART on page 110.

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

15.3.3.2 SmartCard (ISO7816)

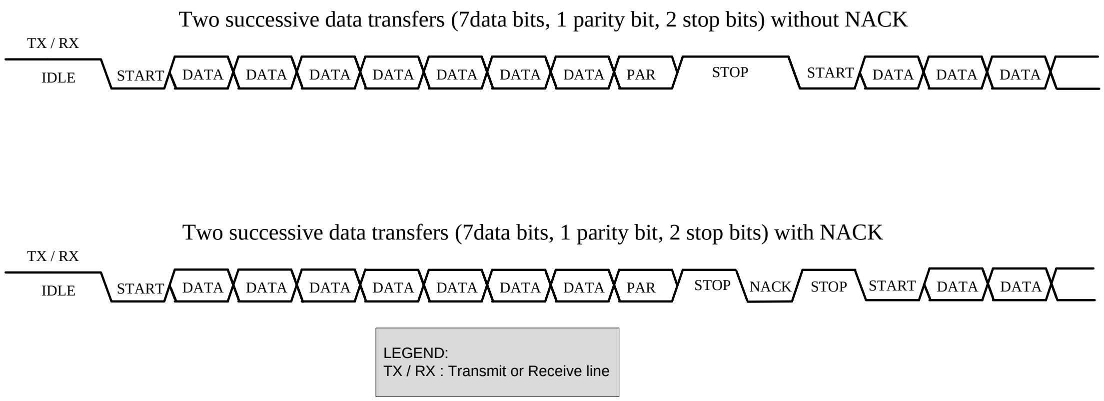
ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic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

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

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

Figure 15-16 illustrates the SmartCard protocol.

Figure 15-16. SmartCard Example



The communication Baud rate for ISO7816 is given as:

Baud rate= f7816 × (D / F)

Where f7816 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 4 Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the SCB_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 SCB_MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Then follow steps 2 to 4 described in Enabling and Initializing UART on page 110.

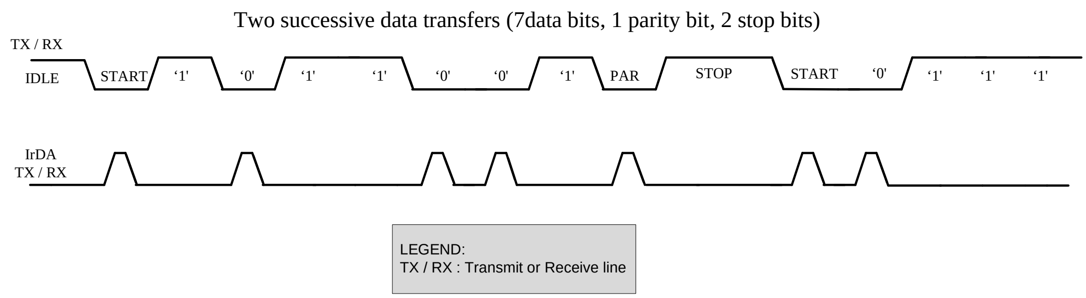
15.3.3.3 IrDA

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

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

Different communication speeds under 115.2 Kb/s can be achieved by configuring corresponding block clock frequency. Additional allowable rates are 2.4 Kbps, 9.6 Kbps, 19.2 Kbps, 38.4 Kbps, and 57.6 Kbps. An IrDA serial infrared interface operates at 9.6 Kbps. Figure 15-17 shows how a UART transfer is IrDA modulated.

Figure 15-17. 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 4 Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the SCB_MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the SCB_MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Configure the SCB as described in Enabling and Initializing UART on page 110.

15.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 15-9. For more information on these registers, see the PSoC 4 Registers TRM.

Table 15-9. UART Registers

Register Name	Operation
SCB_UART_CTRL	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCB_UART_STATUS	Used to specify the BR_COUNTER value that determines the bit period.
SCB_UART_TX_CTRL	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.
SCB_UART_RX_CTRL	Performs same function as SCB_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error.
SCB_TX_CTRL	Used to enable the transmitter, also 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.

15.3.5 UART Interrupts

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

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

- UART transmission done.
- UART TX received a NACK in SmartCard mode.
- UART arbitration lost (in LIN or SmartCard modes).
- Frame error in received data frame.
- Parity error in received data frame.
- LIN baud rate detection is completed.
- LIN break detection is successful.

15.3.6 Enabling and Initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCB_UART_CTRL register, according to Table 15-10. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-11.
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted or received.
   c. Enable the transmitter and receiver.

3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in Table 15-12.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFOs.
4. Program SCB_CTRL register to enable the SCB block. Also select the mode of operation, as shown in Table 15-13.

Table 15-10. SCB_UART_CTRL Register

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

Table 15-11. SCB_TX_CTRL / SCB_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the no. of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits.
8	MSB_FIRST	1= MSB first 0= LSB first
31	ENABLED	Transmitter enable bit for SCB_TX_CTRL and Receiver enable bit for SCB_RX_CTRL registers. They must be enabled for all the protocols. Otherwise, the block may not function or the data may get lost.

Table 15-12. SCB_TX_FIFO_CTRL / SCB_RX_FIFO_CTRL Registers

Bits	Name	Description
[2:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared / invalidated.
17	FREEZE	When '1', hardware reads / writes to the transmitter or receiver FIFO have no effect. Freeze will not advance the TX or RX FIFO read / write pointer.

Table 15-13. SCB_CTRL Register

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

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).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register).

15.4 Inter Integrated Circuit (I2C)

The effective chip-to-chip communication is the important requirement in embedded systems. The serial data transfer standards have become popular in low data-rate applications because it reduces the number of pins, and thus effective area of the chip. The 2-wire Inter Integrated Circuit (I2C) standard is widely used synchronous serial interface for communicating with peripheral devices such as microcontroller, ADC, DAC, and EEPROM.

15.4.1 Features

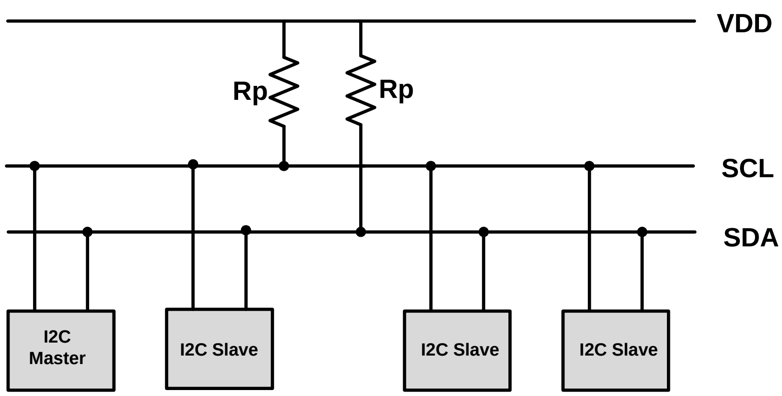
The following are the I2C features supported in PSoC 4.

- 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-rate
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Routes data signal (SDA) and clock signal (SCL) connections directly to a set of dedicated pins
- Clock stretching and collision detection
- Programmable oversampling of I2C clock signal (SCL)
- Error reduction by means of digital median filter on the input path of I2C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter, which can filter out glitches less than 10 ns or 50 ns
- EZI2C mode support
- Externally clocked slave functionality
- Interrupt or polling CPU interface

15.4.2 General Description

Figure 15-18 illustrates an example of I2C master with three slaves.

Figure 15-18. I2C Interface Block Diagram



The standard I2C bus is a two wire interface:

- 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). Simple master/slave relationships exist 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 4 also supports 10-bit address matching with firmware support.

Table 15-14 illustrates the various features supported by I2C interface.

Table 15-14. 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

Bus Stalling (Clock Stretching)

When a slave device is not capable of processing data, it may hold down the SCL line by driving a '0'. Due to the implementation of the I/O signal interface, the SCL line value will be '0', independent of the values 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 reacts by postponing the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock.

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 a SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value '1' on the SDA line and master 2 is driving the value '0' on the SDA line, the actual line value will be '0' (due to the implementation of the I/O signal interface). Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

15.4.3 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 serial interface clock on the serial interface clock lane.

Table 15-15 illustrates the I2C modes of operation.

Table 15-15. 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

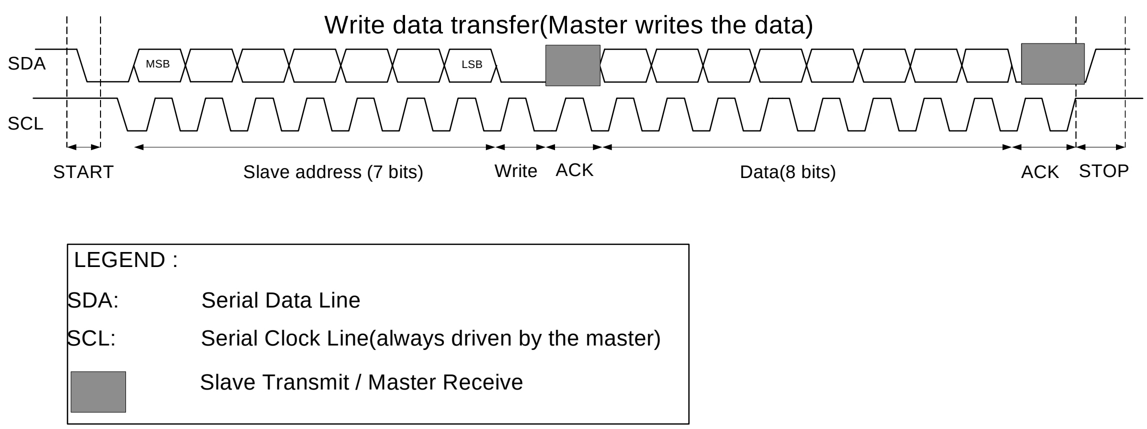
When operating in Multi-Master mode, the bus should always be checked to see if it is busy. Another master may already be communicating with another slave. In this case, the master must wait until the current operation is complete before issuing a START signal. The master looks for STOP signal to start the data transmission.

When operating in Multi-Master-Slave mode, if the master loses arbitration during an address byte, the hardware reverts to Slave mode and the received byte generates a slave address interrupt.

The data transfer in all these modes happens through write and read transfer. 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 120, Slave Mode Transfer Examples on page 122, and Multi-Master Mode Transfer Example on page 126.

15.4.3.1 Write Transfer

Figure 15-19. Master Write Data Transfer



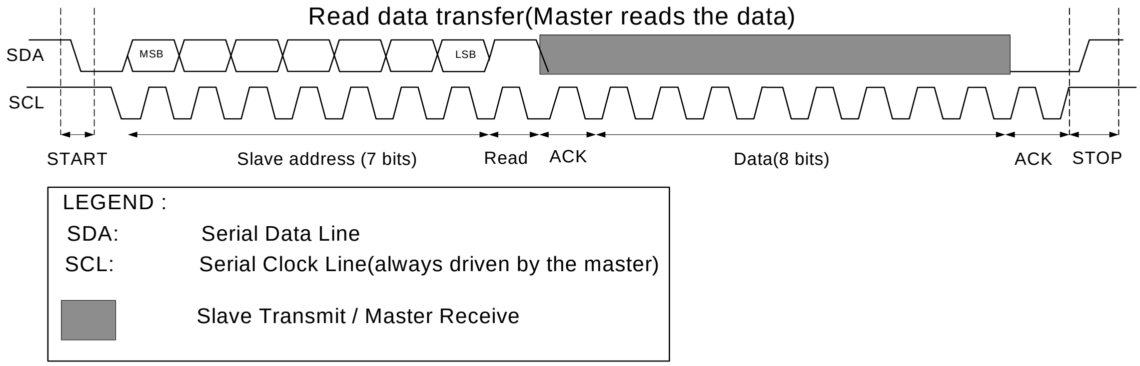
- A typical write transfer starts with the master transmitting a START event. Next, it transmits a 7-bit I2C slave address and a write indicator ('0'). The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, no acknowledgement is transmitted. The absence of an acknowledgement,

results in a SDA line value of '1' (due to the pull-up resistor implementation).

- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. A Repeated Start condition may also be generated for a retry attempt.
- If an acknowledgement is transmitted, the master may transmit write data. The addressed slave transmits an acknowledgement to confirm the receipt of the write data. Upon receipt of this acknowledgement, the master may transmit another write data.
- When the transfer is complete, the master generates a STOP condition.

15.4.3.2 Read Transfer

Figure 15-20. Master Read Data Transfer



- A typical read transfer starts with the master transmitting a START event. Next, it transmits a 7-bit I2C slave address and a read indicator ('1'). The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, no acknowledgement is transmitted. The absence of an acknowledgement, results in a SDA line value of '1' (due to the pull-up resistor implementation).
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event.
- Next, the addressed slave transmits data. The master transmits an acknowledgement to confirm the receipt of the data. Upon receipt of this acknowledgement, the addressed slave may transmit more data.
- When the transfer is complete, the master generates a STOP condition.

15.4.4 Easy I2C (EZI2C) Protocol

The Easy I2C (EZI2C) protocol allows data frame communication between the master and slave without the need for CPU intervention at the level of individual frames. The EZI2C protocol defines an 8-bit EZ address that indexes a memory array (8-bit wide 32 locations) located on the slave device. To address these 32 locations, lower five bits of the EZ address are used. By comparing the EZ address at START detection event and the EZ address at STOP detection event, you can find how many bytes are written into the memory.

Note The SCB has a FIFO memory, which has 16-bit wide 16 locations (16x16) 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, FIFO is used as a single memory unit, which has 8-bit wide 32 locations (32x8).

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

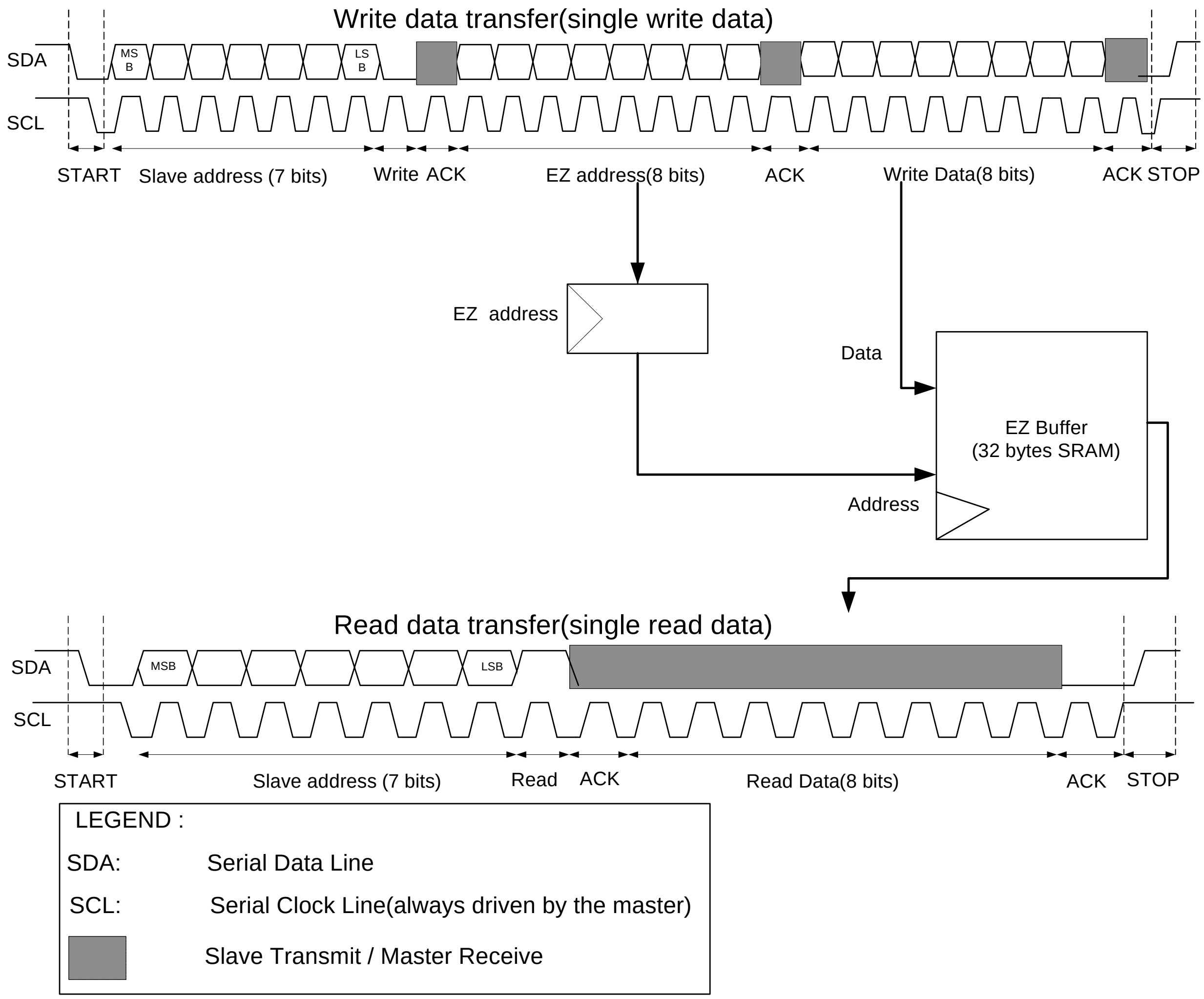
15.4.4.1 Memory Array Write

An EZ write to a memory array index is by means of an I2C 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. When the EZ address exceeds the amount of memory entries of 32, it wraps around to 0.

15.4.4.2 Memory Array Read

An EZ read from a memory array index is by means of an I2C 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.

Figure 15-21. EZI2C Write and Read Data Transfer



See EZ Slave Mode Transfer Example on page 124 for examples.

15.4.4.3 *Configuring SCB for EZI2C Mode*

By default, the SCB is configured for non-EZ mode of operation. To configure the SCB for EZI2C mode, set various register bits in the following order:

1. Select EZI2C mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Follow the steps 2 to 4 mentioned in Enabling and Initializing I2C on page 116.

15.4.5 I2C Registers

The I2C interface is controlled by reading and writing a set of configuration, control, and status registers listed in Table 15-16.

Table 15-16. I2C Registers

Register	Function
SCB_CTRL	Used to enable the SCB block and select the type of serial interface (SPI, UART, I2C). Also used to select internally and externally clocked operation, EZ and non-EZ modes of operation.
SCB_I2C_CTRL	Used for mode selection (Master, Slave) and send ACK or NACK signal based receiver FIFO status.
SCB_I2C_STATUS	Indicates bus busy status detection, Read/Write transfer status of slave/master, and store EZ slave address.
SCB_I2C_M_CMD	Enables master to generate START, STOP, and ACK/NACK signal.
SCB_I2C_S_CMD	Enables slave to generate ACK/NACK signal.

Table 15-16. I2C Registers

Register	Function
SCB_STATUS	In the EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory.
SCB_TX_CTRL	Used to enable the transmitter and specify the data frame width; also used to specify whether MSB or LSB is the first bit in transmission.
SCB_TX_FIFO_CTRL	Used to specify the trigger level, clearing of the transmitter FIFO and shift registers, and for FREEZE operation of the transmitter FIFO.
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_CTRL	Performs the same function as that of the TX_CTRL register, but for the Receiver.
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 EZ memory location.

15.4.6 I2C Interrupts

The I2C interface generates interrupts for the following conditions. These interrupts are internal only; therefore, they cannot be routed to external pins to write custom ISRs. PSoC Creator generates the necessary interrupt service routines (ISRs) for internal interrupts to handle buffer management.

- Arbitration lost
- After slave address match
- I2C bus Stop/Start condition is detected
- I2C bus error is detected
- I2C Byte / Word transfer complete
- I2C TX FIFO is not full
- I2C TX FIFO is empty
- I2C RX FIFO is empty
- I2C RX FIFO is not empty
- I2C RX FIFO is overrun
- I2C RX FIFO is full

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. For more information, see the PSoC 4 Registers TRM.

15.4.7 Enabling and Initializing I2C

The I2C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to Table 15-17. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-18.
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted / received.
   c. Enable the transmitter and receiver
3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in Table 15-19.
   a. Set the trigger level
   b. Clear the transmitter and receiver FIFO and Shift registers
   c. Freeze the TX and RX FIFO
4. Program SCB_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in Table 15-20.

Table 15-17. SCB_I2C_CTRL Register

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

Table 15-18. 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 re-ceived 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
31	ENABLED	Transmitter enable bit for SCB_TX_CTRL and Receiver ena-ble bit for SCB_RX_CTRL regis-ters. They must be enabled for all the protocols. Other-wise, the block may not function or the data may get lost.

Table 15-19. SCB_TX_FIFO_CTRL/ SCB_RX_FIFO_CTRL

Bits	Name	Description
[2:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the val-ue of this field, a transmitter or re-ceiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads / writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read / write pointer.

Table 15-20. 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 enabled
		1	SCB block disabled

The last step of initialization should always be enabling the SCB. When the block is enabled, no control information should be changed. Changes should be made after disabling the block. The change takes effect after the block is re-enabled. Note that disabling the block causes re-initialization of the design and associated state is lost (for example, FIFO content).

15.4.8 Internal and External Clock Operation in I2C

The SCB supports both internally and externally clocked operation for data-rate generation. An internally clocked operation uses a SCBCLK clock, which is derived from the system bus clock. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which a no-chip internal clock is provided to the SCB.

Internally clocked operation uses the high-frequency clock of the system. For more information on system clocking, see the Clocking System chapter on page 61. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The SCB_OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

Externally clocked operation is limited to:

- 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**: Indicates whether I2C slave selection is internally ('0') or externally ('1') clocked. I2C slave selection comprises the first part of the protocol.
- **EC_OP_MODE**: Indicates whether the rest of the protocol operation (besides I2C 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 I2C. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. Table 15-21 and Table 15-22 describe the settings for I2C (in EZ and non EZ mode).

15.4.8.1 Non-EZ Operation Mode

In non-EZ mode, there are two possible settings. As externally clocked operation is not supported for non-EZ functionality (no FIFO support), EC_OP_MODE should always be set to '0'. However, EC_AM_MODE can be set to '0' or '1'. Table 15-21 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.

Table 15-21. I2C Non-EZ Mode

<table>
<tr><th colspan="5">I2C Standard (Non-EZ) Mode</th></tr>
<tr><th></th><th colspan="2">EC_OP_MODE = 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>System Power Mode</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE=1</th><th>EC_AM_MODE=0</th></tr>
<tr><td>Active and Sleep</td><td>Address match using internal clock
Operation using internal clock</td><td>Address match using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in Sleep mode, the MASK bit can be configured by the user.
Operation using internal clock (generates ACK)</td><td>Not supported</td><td rowspan="2">Invalid configuration</td></tr>
<tr><td>Deep-Sleep</td><td>Not Supported</td><td>Address match using external clock: Wakeup interrupt cause is enabled (MASK = 1), generate NACK, or Stretch
Operation using internal clock (generates ACK)</td><td>Not Supported</td></tr>
<tr><td>Hibernate</td><td colspan="4" rowspan="2">The SCB is not available in these modes (see Power Modes chapter on page 75)</td></tr>
<tr><td>Stop</td></tr>
</table>

EC_OP_MODE is '0' and EC_AM_MODE is '0'. This setting only works in Active/Sleep system power mode. The SCB functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1'. This setting works in Active system power mode and Deep-Sleep system power mode. I2C address matching is performed by the externally clocked logic in both Active 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 wake up the CPU.

- 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 (MASK bit can be either '1' or '0') based on the application. The remaining operations are same as that of the Active mode.
- 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 and the ongoing I2C transfer is either negatively acknowledged or the clock is stretched. In the case of a negative acknowledge, the internally clocked logic takes care of the first I2C transfer after it is woken up. In the case of clock stretching, the internally clocked logic takes care of the ongoing/stretched transfer when it is woken up. The register bit S_NOT_READY_ADDR_NACK of SCB_I2C_CTRL register determines whether the externally clocked logic performs a negative acknowledge ('1') or clock stretch ('0').

15.4.8.2 EZ Operation Mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-22 gives an overview of the possibilities (the grey colored cells indicate a possible, yet non preferred setting as it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation)). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

Table 15-22. I2C EZ Mode

<table>
<tr><th colspan="5">I2C, EZ mode</th></tr>
<tr><th></th><th colspan="2">EC_OP_MODE= 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>System Power Mode</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE=0</th></tr>
<tr><td>Active and Sleep</td><td>Address match using internal clock
Operation using internal clock</td><td>Address match using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in Sleep mode, the MASK bit can be configured by the user.
Operation using internal clock (generates ACK)</td><td>Address match using external clock
Operation using external clock</td><td rowspan="2">Invalid configuration</td></tr>
<tr><td>Deep-Sleep</td><td>Not Supported</td><td>Address match using external clock: Wakeup interrupt cause is enabled (MASK = 1), generate NACK or stretch.
Operation using internal clock (generates ACK)</td><td>Address match using external clock
Operation using external clock</td></tr>
<tr><td>Hibernate</td><td colspan="4" rowspan="2">The SCB is not available in these modes (see Power Modes chapter on page 75)</td></tr>
<tr><td>Stop</td></tr>
</table>

- EC_AM_MODE is '0' and EC_OP_MODE is '0'. This setting only works in Active/Sleep system power mode.
- 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 system power mode and Deep-Sleep system power mode.

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

15.4.9 Wake up from Sleep

The system wakes up from sleep or deep-sleep when an address match occurs. The I2C interface performs one of two actions after address match: Address ACK or Address NACK.

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

Address NACK - the I2C slave NACKs the address immediately. The master must poll the slave again after device wakeup time passed. This option is only valid if the mode is Slave or Muti-Master-Slave.

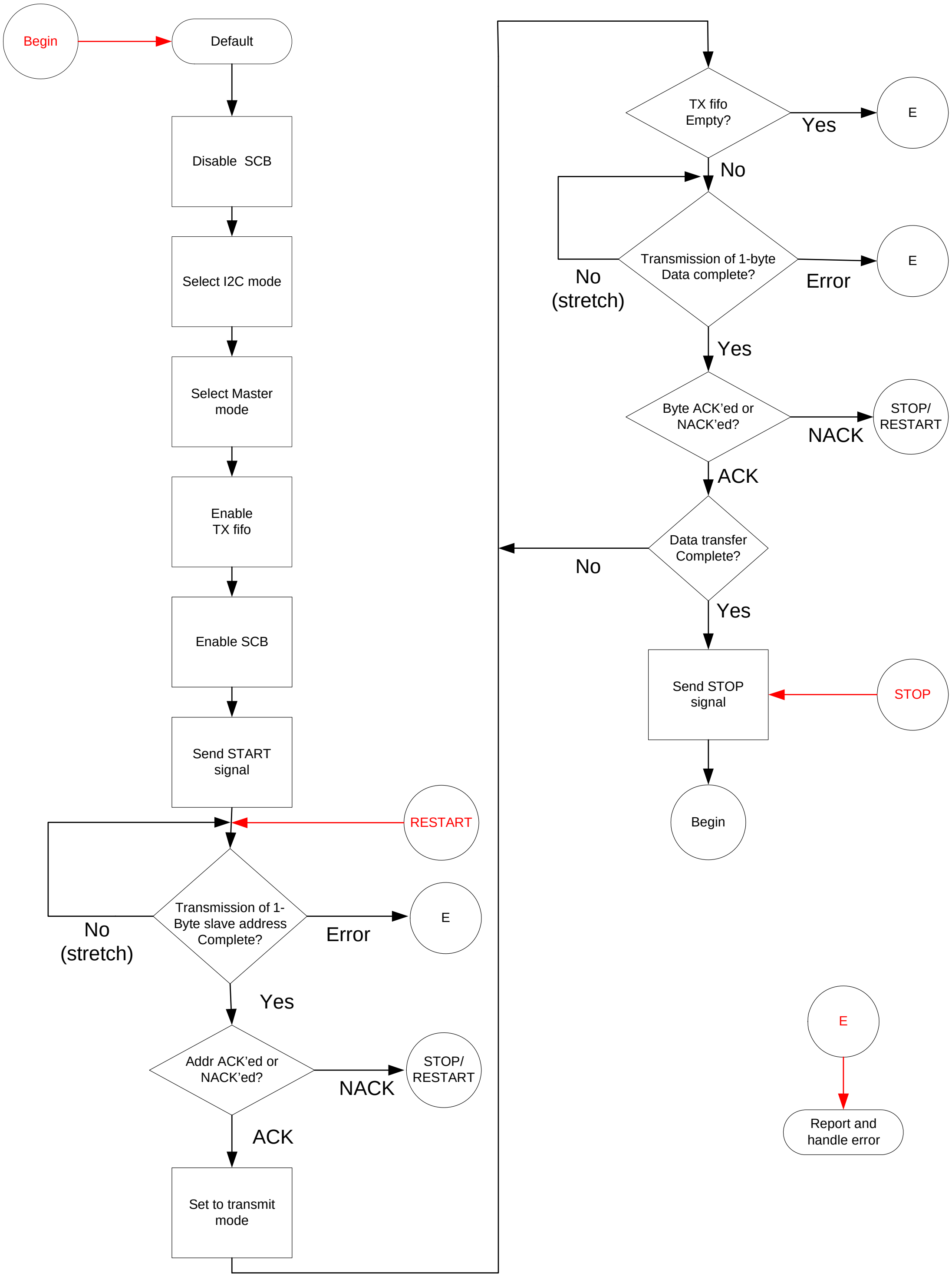
Note You must enable the interrupt bit SCB_INTR_I2C_EC.SCB_WAKE_UP to wake up the device on slave address match while switching to the sleep mode.

15.4.10 Master Mode Transfer Examples

Master mode transmits or receives data.

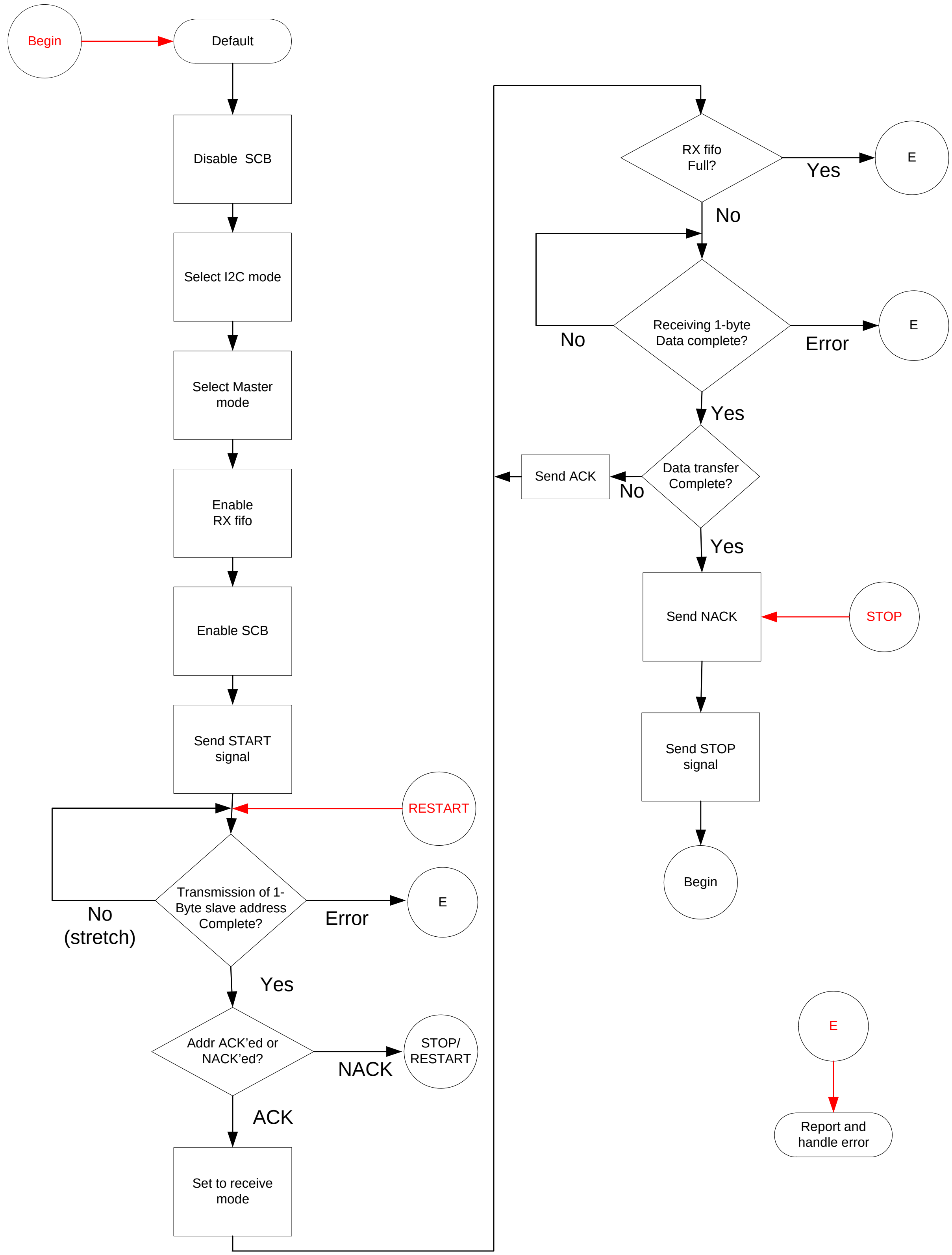
15.4.10.1 Master Transmit

Figure 15-22. Single Master Mode Write Operation Flow Chart



15.4.10.2 Master Receive

Figure 15-23. Single Master Mode Read Operation Flow Chart

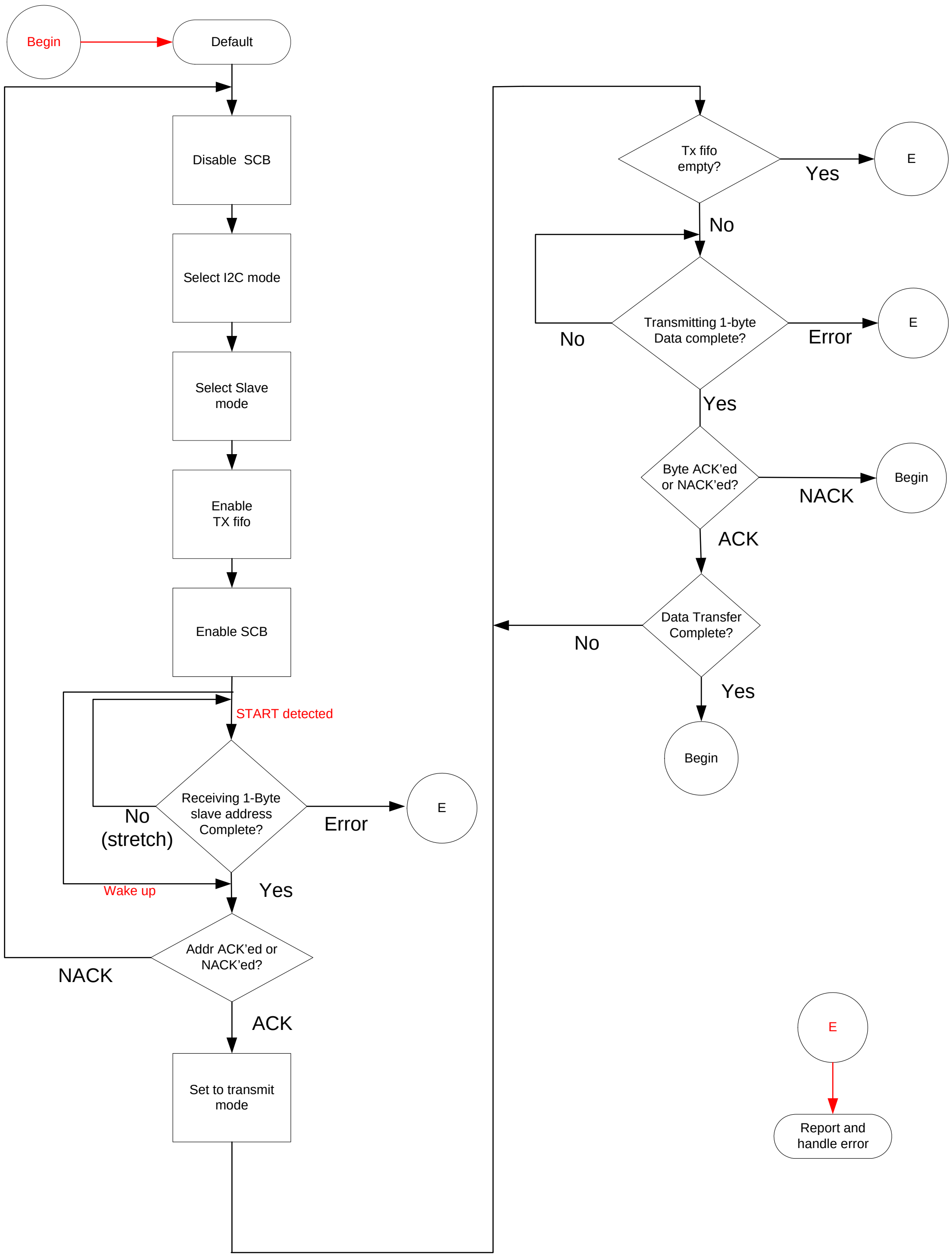


15.4.11 Slave Mode Transfer Examples

Slave mode transmits or receives data.

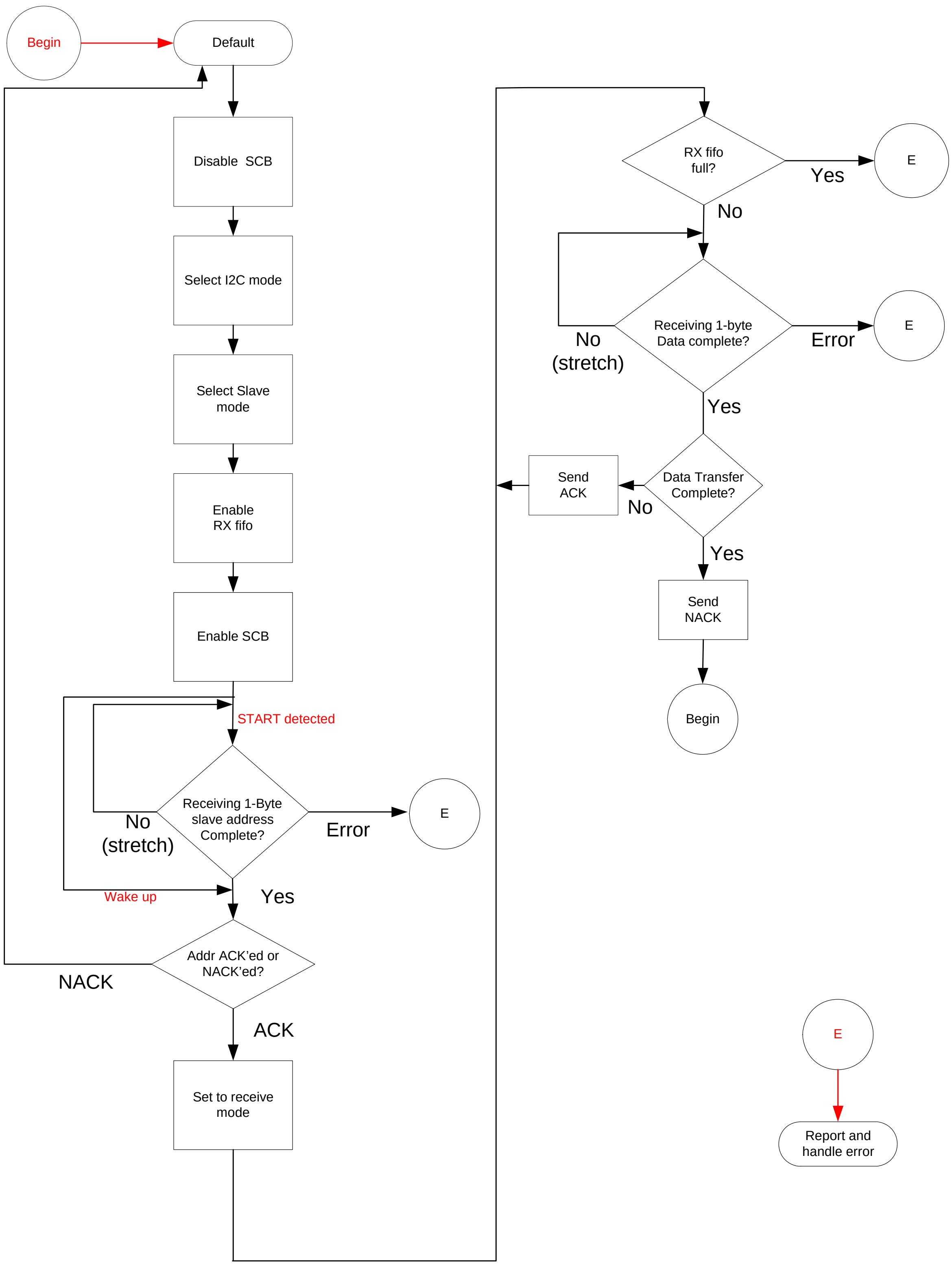
15.4.11.1 Slave Transmit

Figure 15-24. Slave Mode Write Operation Flow Chart



15.4.11.2 Slave Receive

Figure 15-25. Slave Mode Read Operation Flow Chart

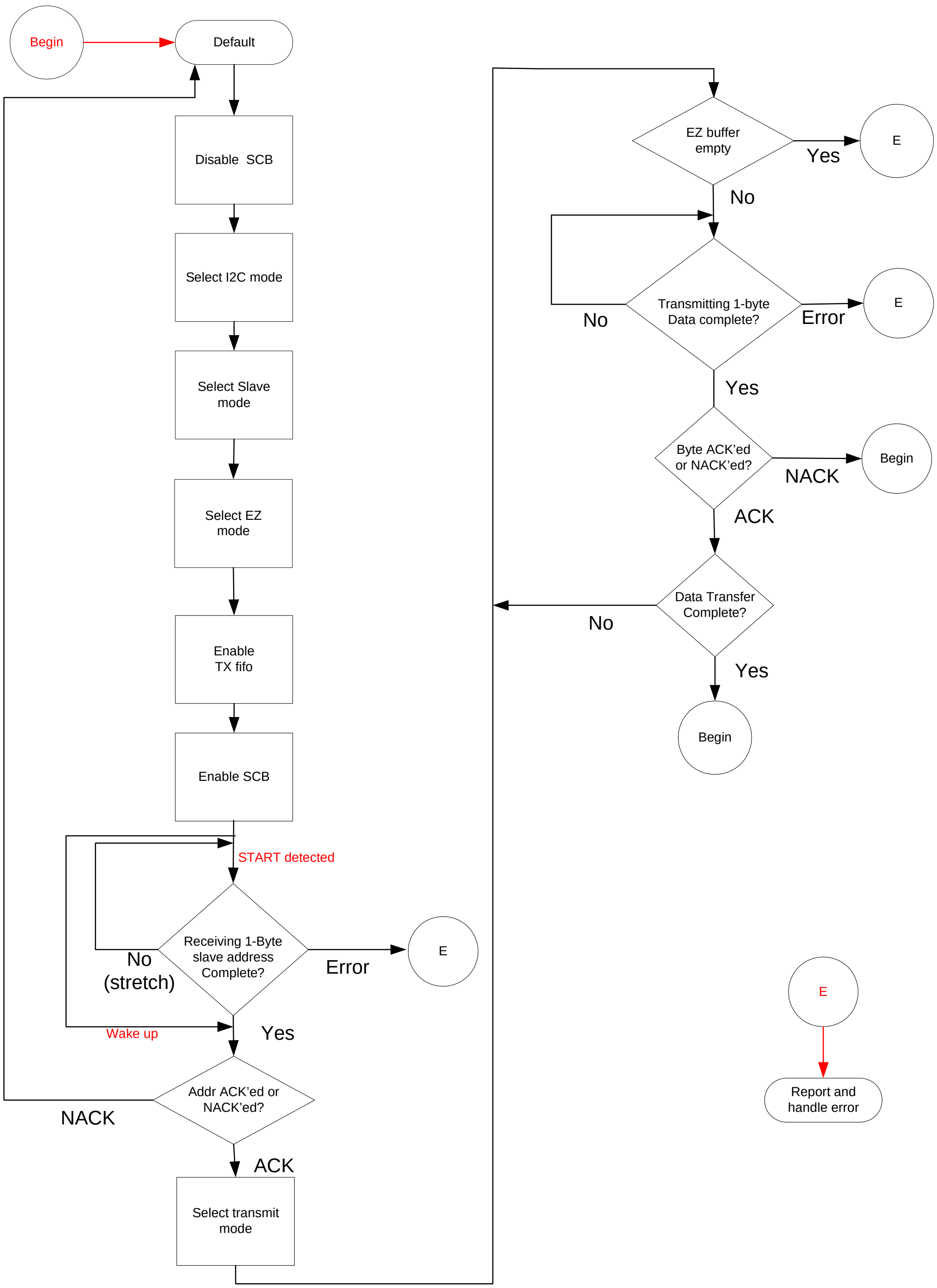


15.4.12 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

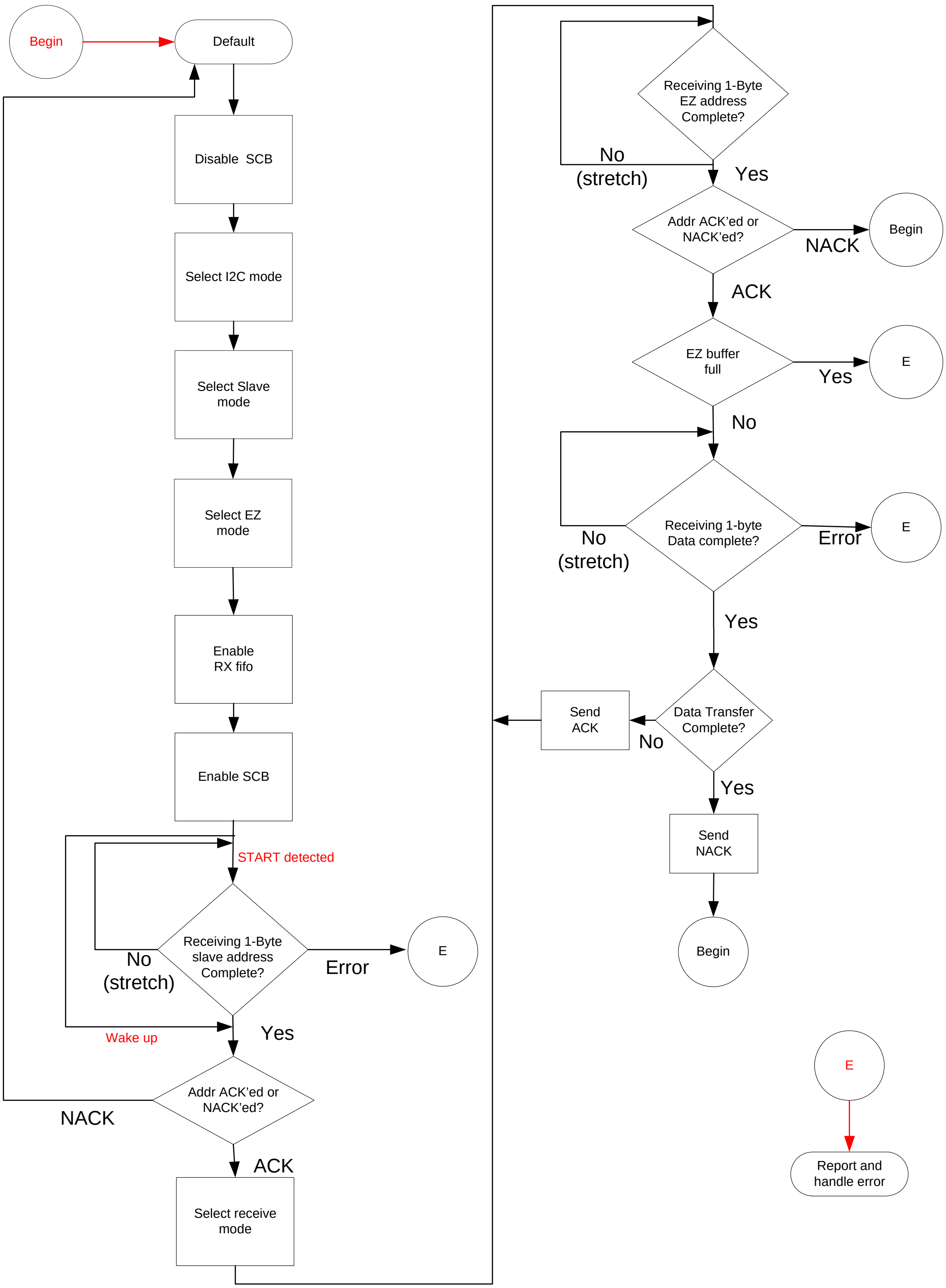
15.4.12.1 EZ Slave Transmit

Figure 15-26. EZI2C Slave Mode Write Operation Flow Chart



15.4.12.2 EZ Slave Receive

Figure 15-27. EZI2C Slave Mode Read Operation Flow Chart

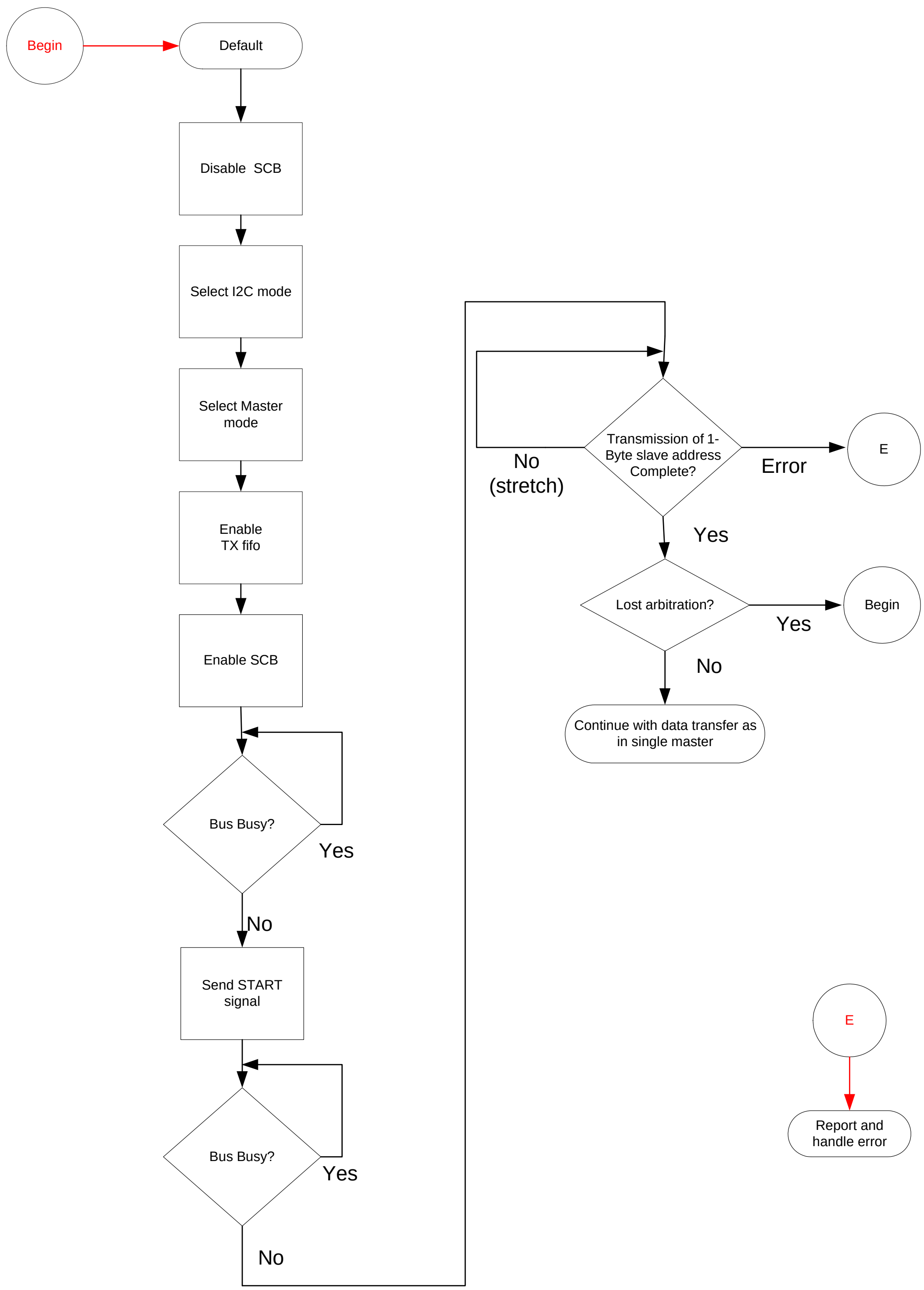


15.4.13 Multi-Master Mode Transfer Example

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

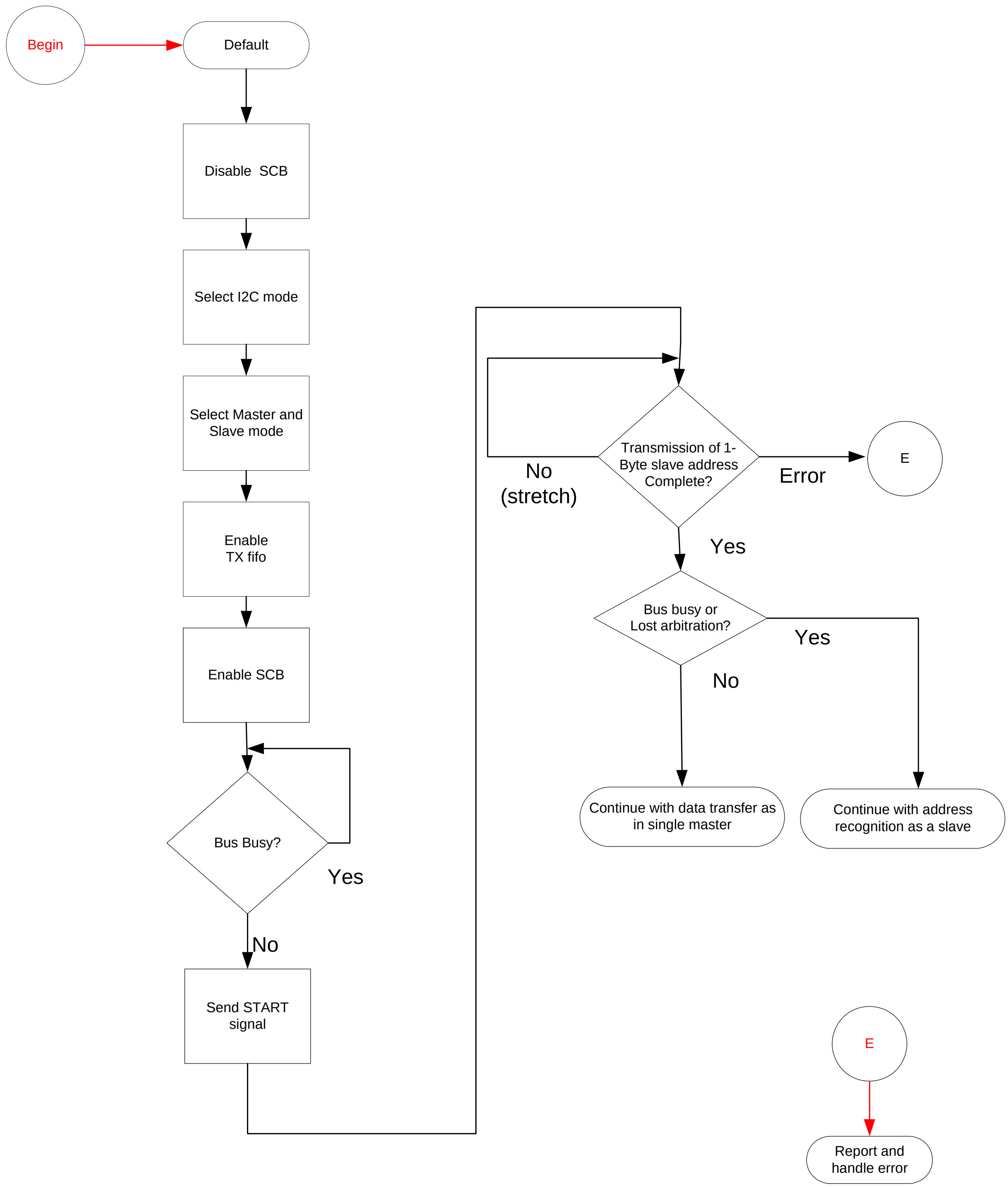
15.4.13.1 Multi-Master - Slave Not Enabled

Figure 15-28. Multi-Master, Slave Not Enabled Flow Chart



15.4.13.2 Multi-Master - Slave Enabled

Figure 15-29. Multi-Master, Slave Enabled Flow Chart



16. Universal Digital Blocks (UDB)

This chapter shows the design details of the PSoC® 4 universal digital blocks (UDBs). The UDB architecture implements a balanced approach between configuration granularity and efficiency; UDBs have a combination of programmable logic devices (PLDs), structured logic (datapaths), and a flexible routing scheme. **Note** UDBs are not supported in the PSoC 4100 family of devices.

16.1 Features

- PSoC 4 contains an array of four UDBs
- For optimal flexibility, each UDB contains several components:
  - An ALU-based 8-bit datapath (DP) with multiple registers, FIFOs, and an 8-word instruction store
  - Two PLDs, each with 12 inputs, eight product terms, and four macrocell outputs
  - Control and status modules
  - Clock and reset modules
- Flexible routing through the UDB array
- Portions of UDBs can be shared or chained to enable larger functions
- Flexible implementations of multiple digital functions, including timers, counters, PWM (with dead band generator), UART, SPI, and CRC generation/checking
- Register-based interface to CPU

Figure 16-1 shows the components of a single UDB: two PLDs, a datapath, and control, status, clock and reset functions. Figure 16-2 shows how the array of four UDBs interfaces with the rest of the PSoC 4.

Figure 16-1. Single UDB Block Diagram

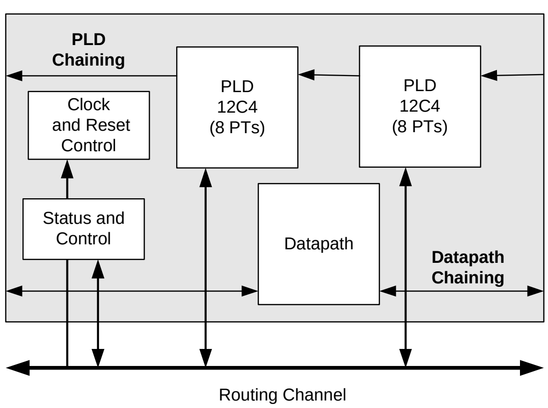
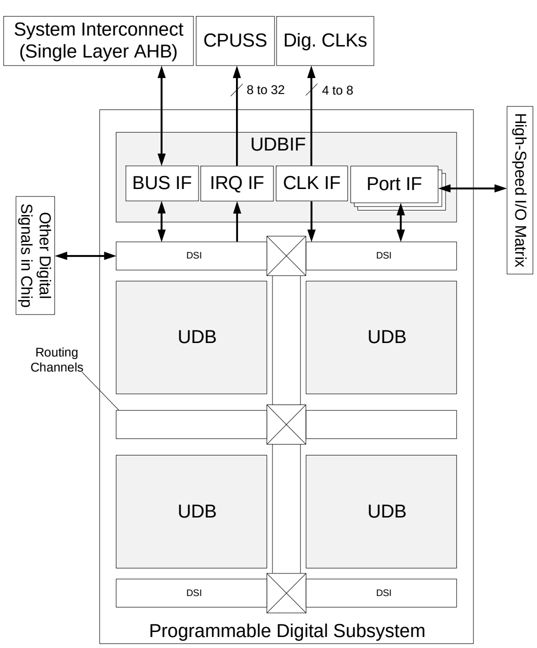


Figure 16-2. UDBs Array in PSoC 4



16.2 How It Works

The major components of a UDB are:

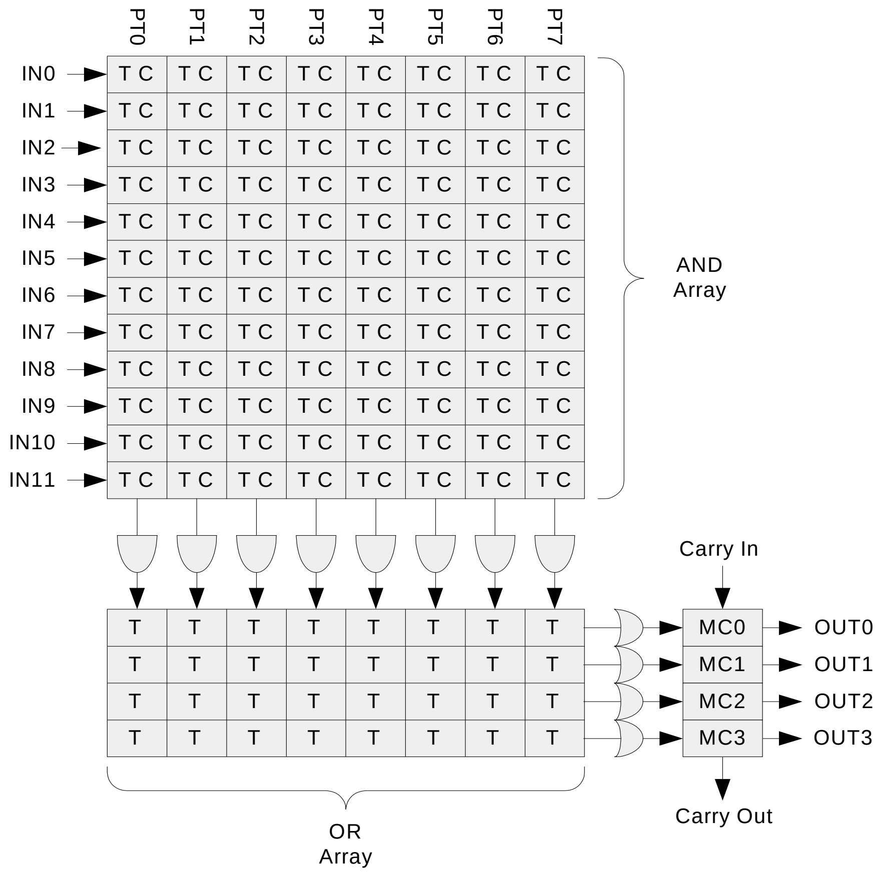
- **PLDs (2)** – These blocks take inputs from the routing channel and form registered or combinational sum-of-products logic to implement state machines, control for datapath operations, conditioning inputs, and driving outputs.
- **Datapath** – This block contains a dynamically programmable ALU, four registers, two FIFOs, comparators, and condition generation.
- **Control and Status** – These modules provide a way for CPU firmware to interact and synchronize with UDB operation.
- **Reset and Clock Control** – These modules provide clock selection and enabling, and reset selection, for the other blocks in the UDB.
- **Chaining Signals** – The PLDs and datapath have chaining signals that enable neighboring UDBs to be linked, to create higher precision functions.
- **Routing Channel** – UDBs are connected to the routing channel through a programmable switch matrix for connections between blocks in one UDB, and to all other UDBs in the array.
- **System Bus Interface** – All registers and RAM in each UDB are mapped into the system address space and are accessible by the CPU as 8, 16 and 32-bit accesses.

16.2.1 PLDs

Each UDB has two “12C4” PLDs. The PLD blocks, shown in Figure 16-3, can be used to implement state machines, perform input or output data conditioning, and to create lookup tables (LUTs). PLDs may also be configured to perform arithmetic functions, sequence the datapath, and generate status. General-purpose RTL can be synthesized and mapped to the PLD blocks. This section presents an overview of the PLD design.

A PLD has 12 inputs, which feed across eight product terms (PT) in the ***AND*** array. In a given product term, the true (T) or complement (C) of the input can be selected. The outputs of the PTs are inputs into the OR array. The 'C' in 12C4 indicates that the OR terms are constant across all inputs, and each OR input can programmatically access any or all of the PTs. This structure gives maximum flexibility and ensures that all inputs and outputs are permutable.

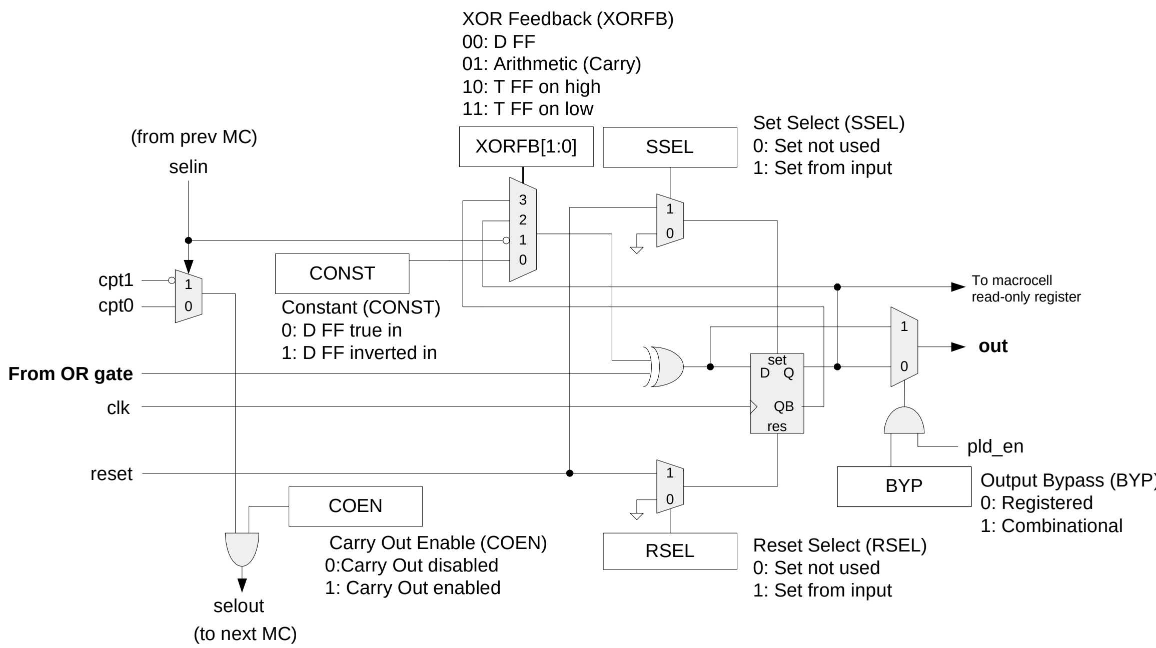
Figure 16-3. PLD 12C4 Structure



16.2.1.1 PLD Macrocells

Figure 16-4 shows the macrocell architecture. The output drives the routing array and can be registered or combinational. The registered modes are D Flip-Flop (DFF) with true or inverted input and Toggle Flip-Flop (TFF) on input high or low. The output register can be set or reset for purposes of initialization, or asynchronously during operation under control of a routed signal.

Figure 16-4. PLD Macrocell Architecture



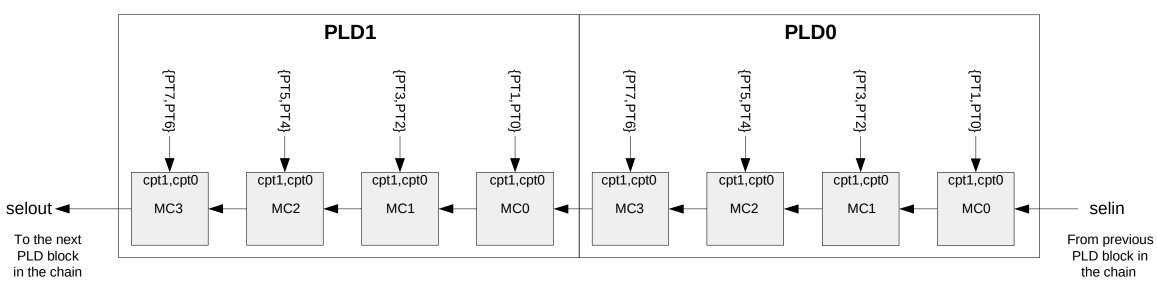
PLD Macrocell Read-Only Registers

The outputs of the eight macrocells in the two PLDs can be accessed by the CPU as an 8-bit read-only register. Macrocells across multiple UDBs can be accessed as 16 or 32-bit read-only registers. See UDB Addressing on page 160.

16.2.1.2 PLD Carry Chain

PLDs are chained together in UDB address order. As shown in Figure 16-5, the carry chain input "selin" is routed from the previous UDB in the chain through each macrocell in both PLDs, and then to the next UDB as the carry chain out "selout". To support the efficient mapping of arithmetic functions, special product terms are generated and used in the macrocell in conjunction with the carry chain.

Figure 16-5. PLD Carry Chain and Special Product Term Inputs



16.2.1.3 PLD Configuration

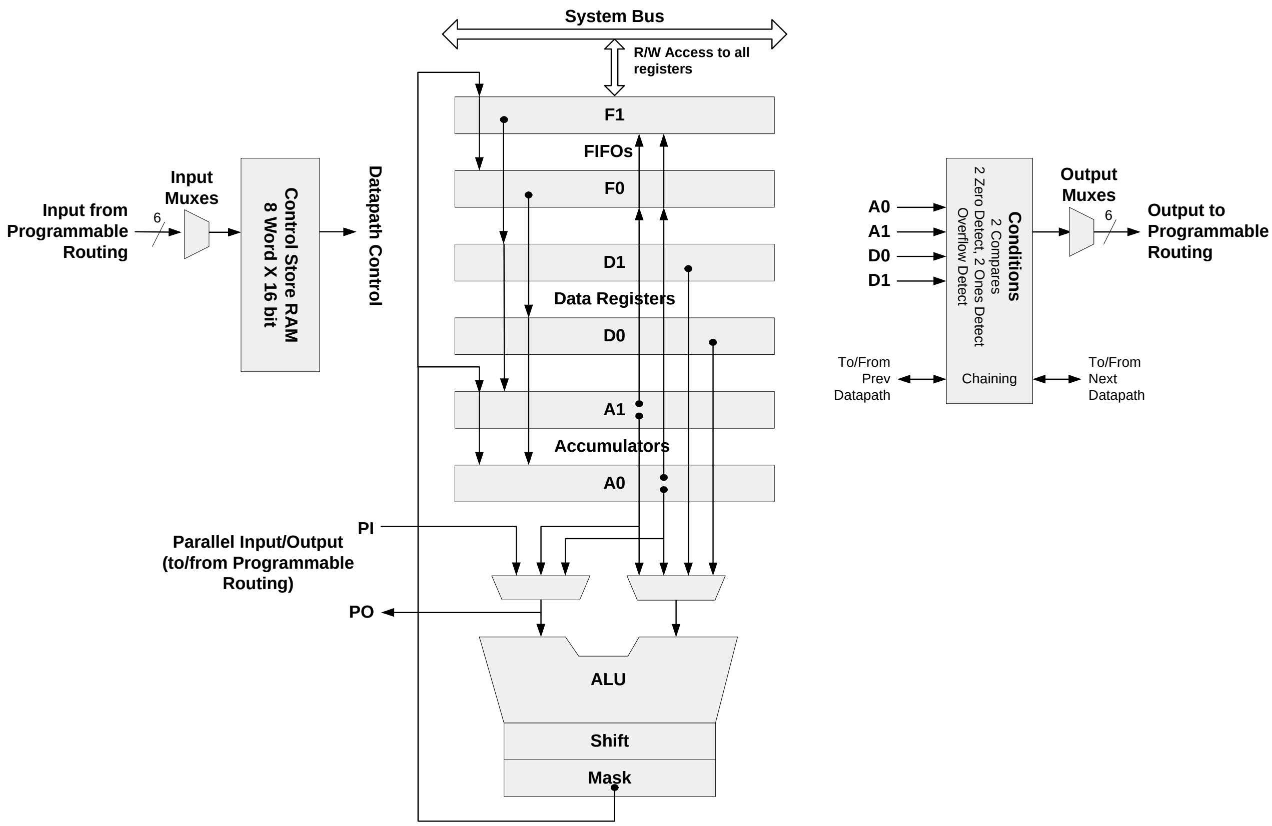
The PLDs can be configured by accessing a set of 16 or 32-bit registers; see UDB Addressing on page 160.

16.2.2 Datapath

The datapath, shown in Figure 16-6, contains an 8-bit single-cycle ALU, with associated compare and condition generation circuits. A datapath may be chained with datapaths in neighboring UDBs to achieve higher precision functions. The datapath includes a small RAM-based control store, which can dynamically select the operation to perform in a given cycle.

The datapath is optimized to implement typical embedded functions such as timers, counters, PWMs, PRS, CRC, shifters, and dead band generators. The add and subtract functions allow support for digital delta-sigma operations.

Figure 16-6. Datapath Top Level



16.2.2.1 Overview

The following are key datapath features:

Dynamic Configuration

Dynamic configuration is the ability to change the datapath function and interconnect on a cycle-by-cycle basis, under sequencer control. This is implemented using the configuration RAM, which stores eight unique configurations. The address input to this RAM can be routed from any block connected to the routing fabric, typically PLD logic, I/O pins, or other datapaths.

ALU

The ALU can perform eight general-purpose functions: increment, decrement, add, subtract, AND, OR, XOR, and PASS. Function selection is controlled by the configuration RAM on a cycle-by-cycle basis. Independent shift (left, right, nibble swap) and masking operations are available at the output of the ALU.

Conditionals

Each datapath has two comparators with bit masking options, which can be configured to select a variety of datapath register inputs for comparison. Other detectable conditions include all zeros, all ones, and overflow. These conditions form the primary datapath output selects to be routed to the digital routing fabric as inputs to other functions.

Built-in CRC/PRS

The datapath has built-in support for single-cycle cyclic redundancy check (CRC) computation and pseudo random sequence (PRS) generation of arbitrary width and arbitrary polynomial specification. To achieve longer than 8-bit CRC/PRS widths, signals may be chained between datapaths. This feature is controlled dynamically and therefore, can be interleaved with other functions.

Variable MSB

The most significant bit of an arithmetic and shift function can be programmatically specified. This supports variable width CRC/PRS functions and, in conjunction with ALU output masking, can implement arbitrary width timers, counters, and shift blocks.

Input/Output FIFOs

Each datapath contains two 4-byte FIFOs, which can be individually configured for direction as an input buffer (CPU writes to the FIFO, datapath internals read the FIFO), or an output buffer (datapath internals write to the FIFO, the CPU reads from the FIFO). These FIFOs generate full or empty status signals that can be routed to interact with sequencers or interrupts.

Chaining

The datapath can be configured to chain conditions and signals with neighboring datapaths. Shift, carry, capture, and other conditional signals can be chained to form higher precision arithmetic, shift, and CRC/PRS functions.

Time Multiplexing

In applications that are oversampled or do not need the highest clock rates, the single ALU block in the datapath can be efficiently shared between two sets of registers and condition generators. ALU and shift outputs are registered and can be used as inputs in subsequent cycles. Usage examples include support for 16-bit functions in one (8-bit) datapath, or interleaving a CRC generation operation with a data shift operation.

Datapath Inputs

The datapath has three types of inputs: configuration, control, and serial and parallel data. The configuration inputs select the control store RAM address. The control inputs load the data registers from the FIFOs and capture ***accumulator*** outputs into the FIFOs. Serial data inputs include shift in and carry in. A parallel data input port allows up to eight bits of data to be brought in from routing.

Datapath Outputs

A total of 16 signals are generated in the datapath. Some of these signals are conditional signals (for example, compares), some are status signals (for example, FIFO status), and the rest are data signals (for example, shift out). These 16 signals are multiplexed into the six datapath outputs and then driven to the routing matrix. By default, the outputs are single synchronized (pipelined). A combinational output option is also available for these outputs.

Datapath Working Registers

Each datapath module has six 8-bit working registers. All registers are readable and writable by CPU:

Table 16-1. Datapath Working Registers

Type	Name	Description
Accumulator	A0, A1	The accumulators may be both a source and a destination for the ALU. They may also be loaded from a Data register or a FIFO. The accumulators typically contain the current value of a function, such as a count, CRC, or shift. These registers are non-retention; they lose their values in sleep and are reset to 0x00 on wakeup.
Data	D0, D1	The Data registers typically contain constant data for a function, such as a PWM compare value, timer period, or CRC polynomial. These registers retain their values across sleep intervals.
FIFOs	F0, F1	The two 4-byte FIFOs provide both a source and a destination for buffered data. The FIFOs can be configured as both input buffers, both output buffers, or as one input buffer and one output buffer. Status signals indicate the full/empty status of these registers. Usage examples include buffered TX and RX data in the SPI or UART and buffered PWM compare and buffered timer period data. These registers are non-retention; they lose their values in sleep and are reset to 0x00 on wakeup.

16.2.2.2 Datapath FIFOs

FIFO Modes and Configurations

Each FIFO has a variety of operation modes and configurations.

Table 16-2. FIFO Modes and Configurations

Mode	Description
Input/Output	In input mode, the CPU writes to the FIFO and the data is read and consumed by the datapath internals. In output mode, the FIFO is written to by the datapath internals and is read and consumed by the CPU.
Single Buffer	The FIFO operates as a single-byte buffer with no status. Data written to the FIFO is immediately available for reading, and can be overwritten at anytime.
Level/Edge	The control to load the FIFO from the datapath internals can be either level or edge triggered.
Normal/Fast	The control to load the FIFO from the datapath source is sampled on the currently selected datapath clock (normal) or the bus clock (fast). This allows captures to occur at the highest rate in the system (bus clock), independent of the datapath clock.
Software Capture	When this mode is enabled and the FIFO is in output mode, a read by the CPU of the associated accumulator (A0 for F0, A1 for F1) initiates a synchronous transfer of the accumulator value into the FIFO. The captured value may then be immediately read from the FIFO. If chaining is enabled, the operation follows the chain to the MS block for atomic reads by datapaths of multi-byte values.
Asynch	When the datapath is being clocked asynchronously to the bus clock, the FIFO status signals can be routed to the rest of the datapath either directly, single sampled to the datapath clock, or double sampled in the case of an asynchronous datapath clock
Independent Clock Polarity	Each FIFO has a control bit to invert polarity of the FIFO clock with respect to the datapath clock.

Figure 16-7 shows the possible FIFO configurations controlled by the input/output modes. The TX/RX mode has one FIFO in input mode and the other in output mode. The primary example of this configuration is SPI. The dual capture configuration provides independent capture of A0 and A1, or two separately controlled captures of either A0 or A1. Finally, the dual buffer mode can provide buffered periods and compares, or two independent periods/compares.

Figure 16-7. FIFO Configurations

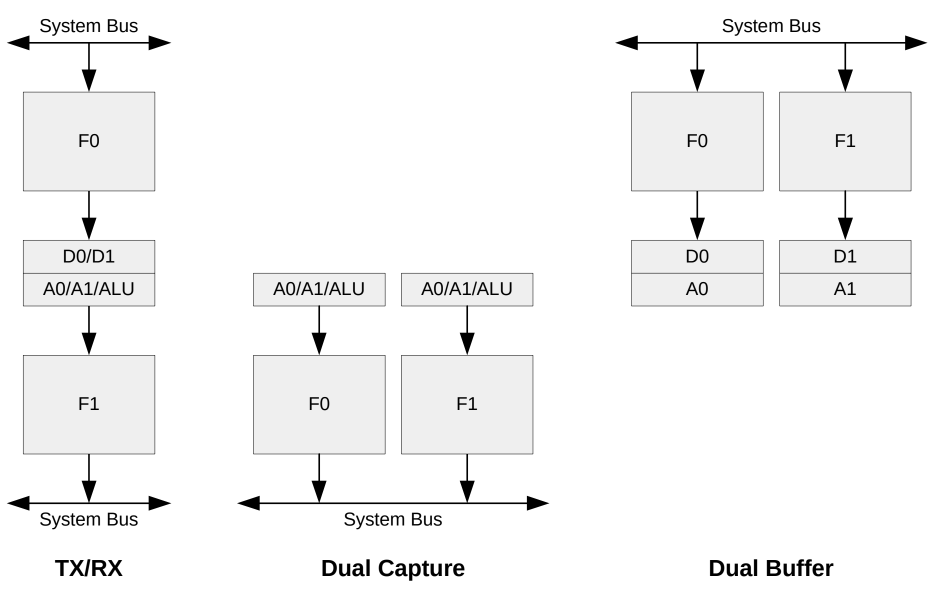
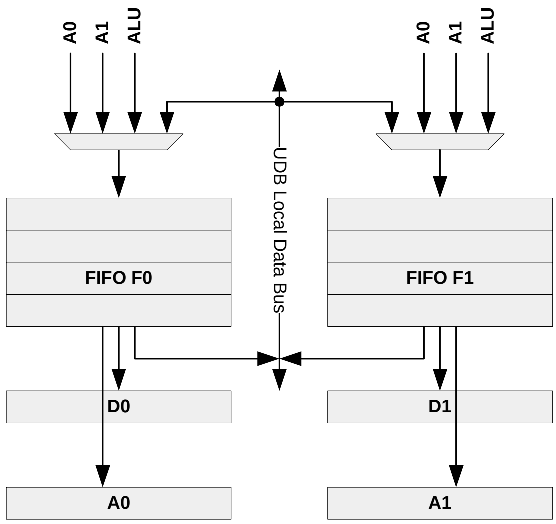


Figure 16-8 shows a detailed view of FIFO sources and sinks.

Figure 16-8. FIFO Sources and Sinks



When the FIFO is in input mode, the source is the system bus and the sinks are the Dx and Ax registers. When in output mode, the sources include the Ax registers and the ALU, and the sink is the system bus. The multiplexer selection is statically set in UDB configuration register CFG15, as shown in Table 16-3 for the F0_INSEL[1:0] or F1_INSEL[1:0].

Table 16-3. FIFO Multiplexer Set in UDB Configuration Register

Fx_INSEL[1:0]	Description
00	**Input mode** - System bus writes the FIFO, FIFO output destination is Ax or Dx.
01	**Output Mode** - FIFO input source is A0, FIFO output destination is the system bus.
10	**Output Mode** - FIFO input source is A1, FIFO output destination is the system bus.
11	**Output Mode** - FIFO input source is the ALU output, FIFO output destination is the system bus.

FIFO Status

Each FIFO generates two status signals, “bus” and “block,” which are sent to the UDB routing through the datapath output multiplexer. The “bus” status can be used to assert an interrupt request to read/write the FIFO. The “block” status is primarily intended to provide the FIFO state to the UDB internals. The meanings of the status bits depend on the configured direction (Fx_INSEL[1:0]) and the FIFO level bits. The FIFO level bits (Fx_LVL) are set in the Auxiliary Control Working register in working register space. Table 16-4 shows the options.

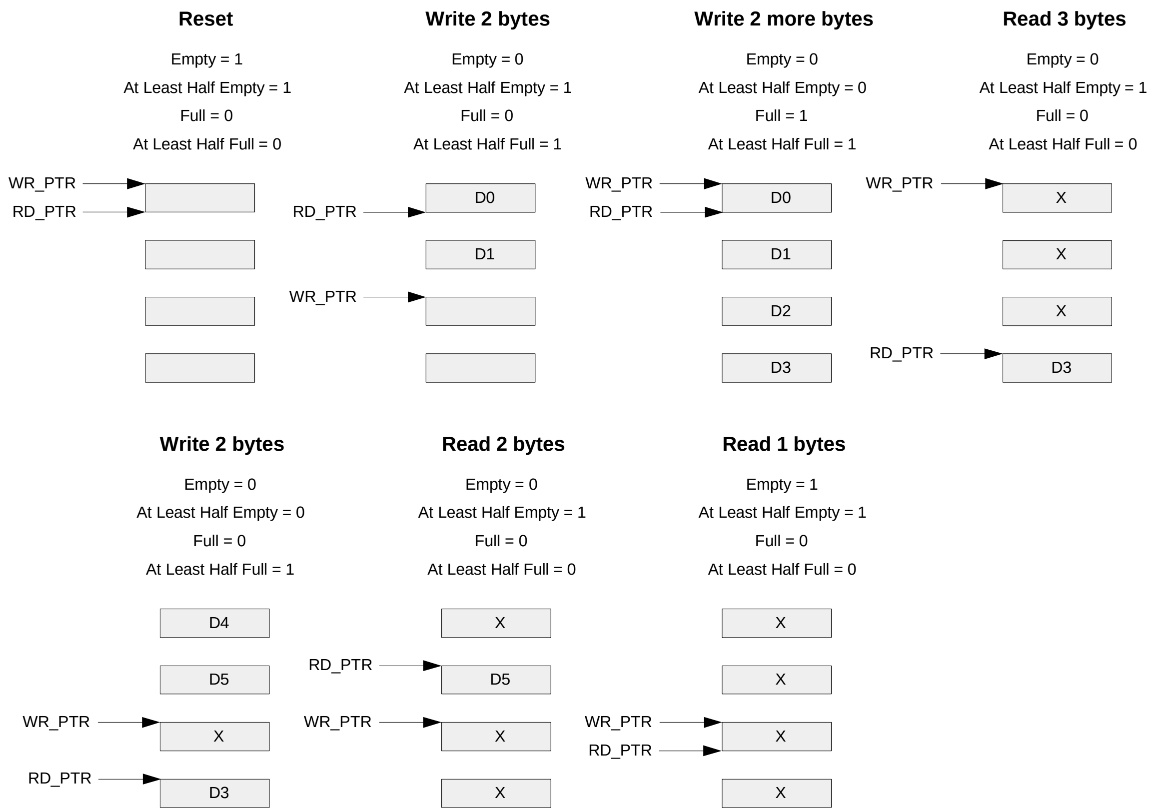
Table 16-4. FIFO Status Options

Fx_INSEL[1:0]	Fx_LVL	Status	Signal	Description
Input	0	Not Full	Bus Status	Asserted when there is room for at least 1 byte in the FIFO.
Input	1	At Least Half Empty	Bus Status	Asserted when there is room for at least 2 bytes in the FIFO.
Input	NA	Empty	Block Status	Asserted when there are no bytes left in the FIFO. When not empty, the datapath internals may consume bytes. When empty the datapath may idle or generate an underrun condition.
Output	0	Not Empty	Bus Status	Asserted when there is at least 1 byte available to be read from the FIFO.
Output	1	At Least Half Empty	Bus Status	Asserted when there are at least 2 bytes available to be read from the FIFO.
Output	NA	Full	Block Status	Asserted when the FIFO is full. When not full, the datapath internals may write bytes to the FIFO. When full, the datapath may idle or generate an overrun condition.

FIFO Operation

Figure 16-9 illustrates a typical sequence of reads and writes and the associated status generation. Although the figure shows reads and writes occurring at different times, a read and write can also occur simultaneously.

Figure 16-9. Detailed FIFO Operation Sinks

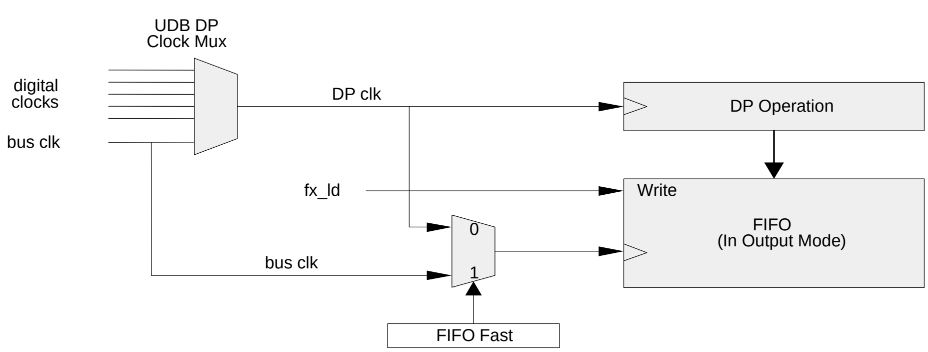


FIFO Fast Mode (FIFO FAST)

When the FIFO is configured for output, the FIFO load operation normally uses the currently selected datapath clock for sampling the write signal. As shown in Figure 16-10, with the FIFO fast mode set, the bus clock can be optionally selected for this operation. Used in conjunction with edge sensitive mode, this operation reduces the latency of accumulator-to-FIFO transfer from the resolution of the datapath clock to the resolution of the bus clock, which can be much higher. This allows the CPU to read the captured result in the FIFO with minimal latency.

Figure 16-10 illustrates that the fast load operation is independent of the currently selected datapath clock; however, using the bus clock may cause higher power consumption. Note that the incoming fx_ld signal must be able to meet bus clock timing, which can require local resynchronization.

Figure 16-10. FIFO Fast Configuration Sinks



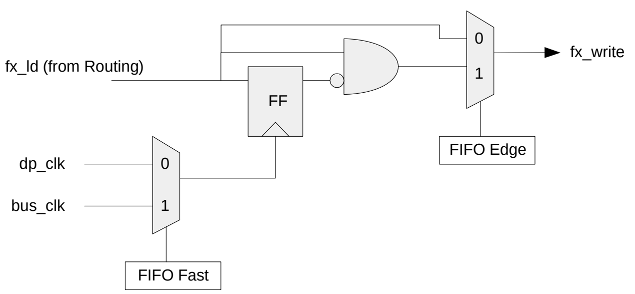
FIFO Edge/Level Write Mode

Two modes are available for writing the FIFO from the datapath. In the first mode, data is synchronously transferred from the accumulators to the FIFOs. The control for that write (fx_ld) is typically generated from a state machine or condition that is synchronous to the datapath clock. The FIFO is written in any cycle where the input load control is a '1'.

In the second mode, the FIFO is used to capture the value of the accumulator in response to a positive edge of the fx_ld signal. In this mode the duty cycle of the waveform is arbitrary (however, it must be at least one datapath clock cycle in width). An example of this mode is capturing the value of the accumulator using an external pin input as a trigger. The limitation of this mode is that the input control must revert to '0' for at least one cycle before another positive edge is detected.

Figure 16-11 shows the edge detect option on the fx_ld control input. One bit for this option sets the mode for both FIFOs in a UDB. Note that edge detection is sampled at the rate of the selected FIFO clock.

Figure 16-11. Edge Detect Option for Internal FIFO Write



FIFO Software Capture Mode

A common and important requirement is to allow the CPU the ability to reliably read the contents of an accumulator during normal operation. This is done with software capture and is enabled by setting the FIFO Cap configuration bit. This bit applies to both FIFOs in a UDB, but is only operational when a FIFO is in output mode. When using software capture, F0 should be set to load from A0 and F1 from A1.

As shown in Figure 16-12, reading the accumulator triggers a write to the FIFO from that accumulator. This signal is chained so that a read of a given byte simultaneously captures accumulators in all chained UDBs. This allows the CPU to reliably read 16 bits or more simultaneously. The data returned in the read of the accumulator should be ignored; the captured value may be read from the FIFOs immediately.

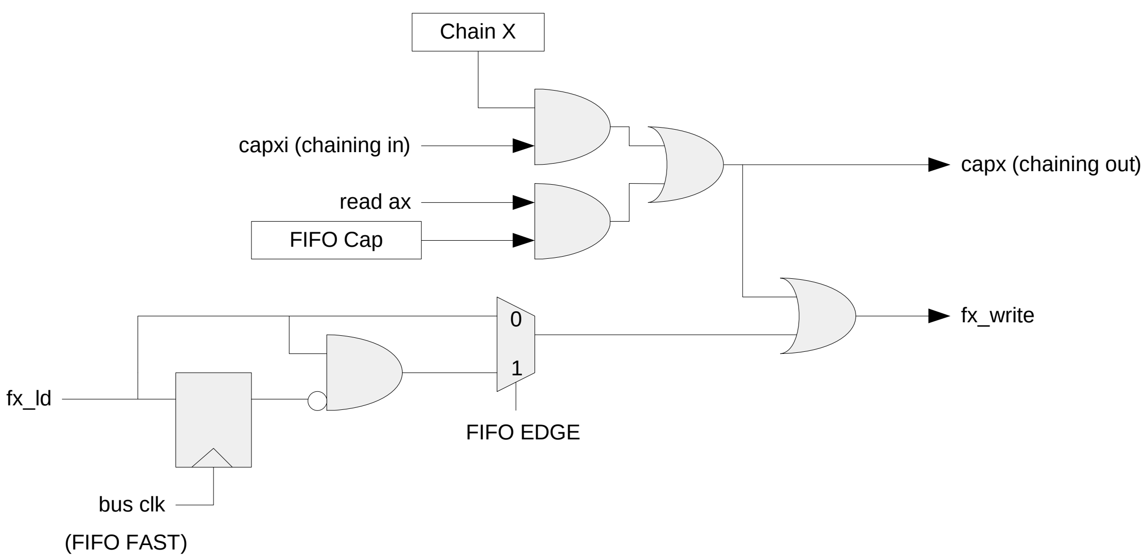
The fx_ld signal, which generates a FIFO load, is ORed with the software capture signal; the results can be unpredictable when both hardware and software capture are used at the same time. As a general rule, these functions should be mutually exclusive; however, hardware and software capture can be used simultaneously with the following settings:

- FIFO capture clocking mode is set to FIFO FAST
- FIFO write mode is set to FIFO EDGE

With these settings, hardware and software capture work essentially the same and in any given bus clock cycle, either signal asserted initiates a capture.

It is also recommended to clear the target FIFO in firmware (ACTL register) before initiating a software capture. This initializes the FIFO read and write pointers to a known state.

Figure 16-12. Software Capture Configuration



FIFO Control Bits

The Auxiliary Control register has four bits that may be used by the CPU firmware to control the FIFO during normal operation.

The FIFO0 CLR and FIFO1 CLR bits are used to reset or flush the FIFO. When a '1' is written to one of these bits, the associated FIFO is reset. The bit must be written back to '0' for FIFO operation to continue. If the bit is left asserted, the given FIFO is disabled and operates as a one byte buffer without status. Data can be written to the FIFO; the data is immediately available for reading and can be overwritten at anytime. Data direction using the Fx INSEL[1:0] configuration bits is still valid.

The FIFO0 LVL and FIFO1 LVL bits control the level at which the 4-byte FIFO asserts bus status (when the bus is either reading or writing to the FIFO) to be asserted. The meaning of FIFO bus status depends on the configured direction, as shown in Table 16-5.

Table 16-5. FIFO Level Control Bits

FIFOx LVL	Input Mode (Bus is Writing FIFO)	Output Mode (Bus is Reading FIFO)
0	Not Full At least 1 byte can be written	Not Empty At least 1 byte can be read
1	At least Half Empty At least 2 bytes can be written	At least Half Full At least 2 bytes can be read

FIFO Asynchronous Operation

Figure 16-13 illustrates the concept of asynchronous FIFO operation. As an example, assume F0 is set for input mode and F1 is set for output mode, which is a typical configuration for TX and RX registers.

On the TX side, the datapath state machine uses "empty" to determine if there are any bytes available to consume. Empty is set synchronously to the DP state machine, but is cleared asynchronously due to a bus write. When cleared, the status is synchronized back to the DP state machine.

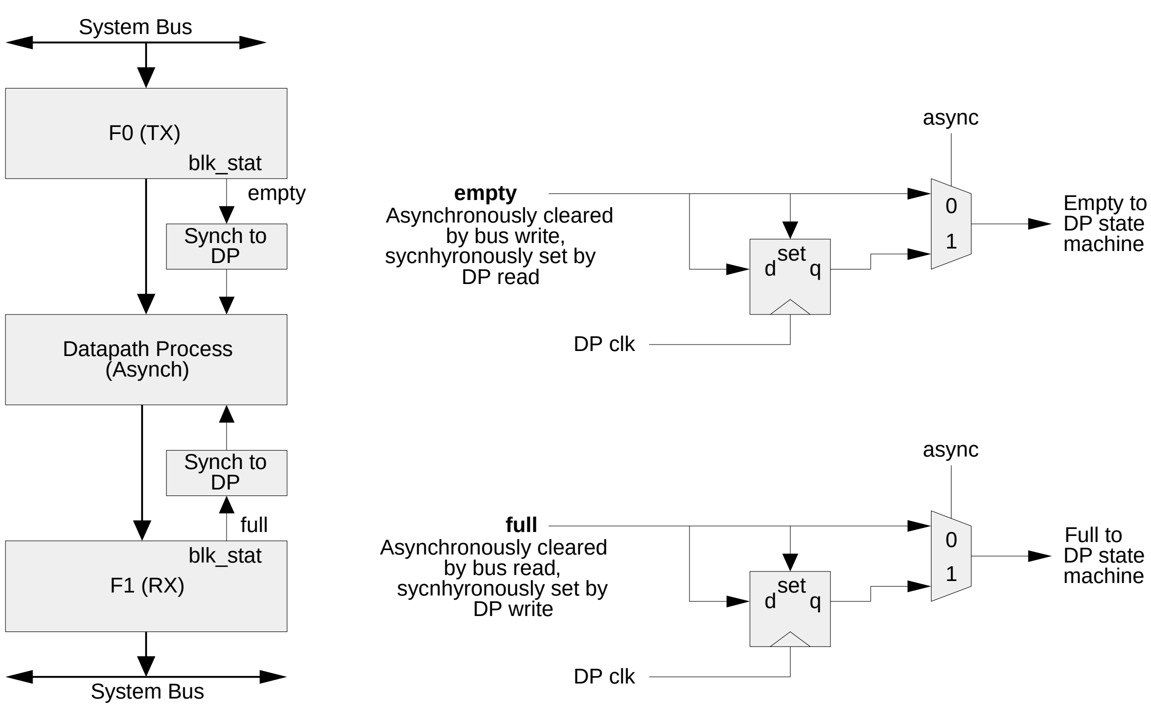
On the RX side, the datapath state machine uses “full” to determine whether there is a space left to write to the FIFO. Full is set synchronously to the DP state machine, but is cleared asynchronously due to a bus read. When cleared, the status is synchronized back to the DP state machine.

A single FIFO ASYNCH bit is used to enable this synchronization method; when set it applies to both FIFOs. It is only applied to the block status, as it is assumed that bus status is naturally synchronized by the interrupt process.

FIFO Overflow Operation

Use FIFO status signaling to safely implement both internal (datapath) and external (CPU) reads and writes. There is no built-in protection from underflow and overflow conditions. If the FIFO is full and subsequent writes occur (overflow), the new data overwrites the front of the FIFO (the data currently being output, the next data to read). If the FIFO is empty and subsequent reads occur (underflow), the read value is undefined. FIFO pointers remain accurate regardless of underflow and overflow.

Figure 16-13. FIFO Asynchronous Operation



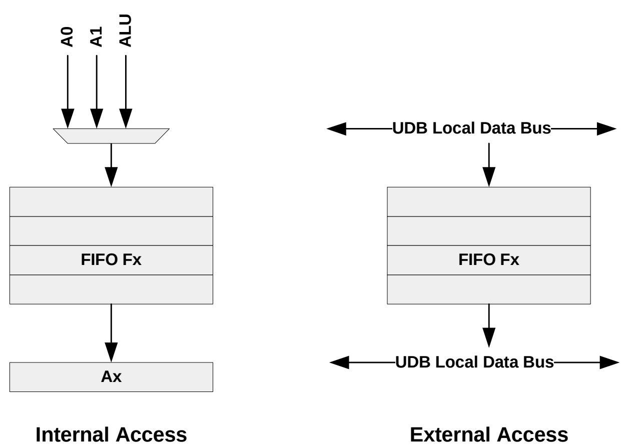
FIFO Clock Inversion Option

Each FIFO has a control bit called Fx CK INV that controls the polarity of the FIFO clock, with respect to the polarity of the DP clock. By default, the FIFO operates at the same polarity as the DP clock. When this bit is set, the FIFO operates at the opposite polarity as the DP clock. This provides support for "both clock edge" communication protocols, such as SPI.

FIFO Dynamic Control

Normally, the FIFOs are configured statically in either input or output mode. As an alternative, each FIFO can be configured into a mode where the direction is controlled dynamically, that is, by routed signals. One configuration bit per FIFO (Fx DYN) enables the mode. Figure 16-13 shows the configurations available in dynamic FIFO mode.

Figure 16-14. FIFO Dynamic Mode



In internal access mode, the datapath can read and write the FIFO. In this configuration, the Fx INSEL bits must be configured to select the source for the FIFO writes. Fx INSEL = 0 (CPU bus source) is invalid in this mode; they can only be 1, 2, or 3 (A0, A1, or ALU). Note that the only read access is to the associated accumulator; the data register destination is not available in this mode.

In external access mode, the CPU can both read and write the FIFO. The configuration between internal and external access is dynamically switchable using datapath routing signals. The datapath input signals d0_load and d1_load are used for this control. Note that in the dynamic control mode, d0_load and d1_load are not available for their normal use in loading the D0/D1 registers from F0/F1. The dx_load signals can be driven by any routed signal, including constants.

In one usage example, starting with external access (dx_load == 1), the CPU can write one or more bytes of data to the FIFO. Then toggling to internal access (dx_load == 0), the datapath can perform operations on the data. Then toggling back to external access, the CPU can read the result of the computation.

Because the Fx INSEL must always be set to 01, 10, or 11 (A0, A1, or ALU), which is "output mode" in normal operation, the FIFO status signals have the following definitions (also dependent on Fx LVL control).

Table 16-6. FIFO Status

Status Signal	Meaning	Fx LVL = 0	Fx LVL = 1
fx_blk_stat	Write Status	FIFO full	FIFO full
fx_bus_stat	Read Status	FIFO not empty	At least ½ full

Because the datapath and CPU may both write and read the FIFO, these signals are no longer considered "block" and "bus" status. The blk_stat signal is used for write status and the bus_stat signal is used for read status.

16.2.2.3 FIFO Status

There are four FIFO status signals, two for each FIFO: fifo0_bus_stat, fifo0_blk_stat, fifo1_bus_stat, and fifo1_blk_stat. The meaning of these signals depends on the direction of the given FIFO, which is determined by static configuration.

16.2.2.4 Datapath ALU

The ALU core consists of three independent 8-bit programmable functions, which include an arithmetic/logic unit, a shifter unit, and a mask unit.

Arithmetic and Logic Operation

The ALU functions, which are configured dynamically by the RAM control store, are shown in Table 16-7.

Table 16-7. ALU Functions

Func[2:0]	Function	Operation
000	PASS	srca
001	INC	++srca
010	DEC	--srca
011	ADD	srca +srcb
100	SUB	srca – srcb
101	XOR	srca ^ srcb
110	AND	srca and srcb
111	OR	srca | srcb

Carry In

The carry in is used in arithmetic operations. Table 16-8 shows the default carry in value for certain functions.

Table 16-8. Carry In Functions

Function	Operation	Default Carry In Implementation
INC	++srca	srca + 00h + ci, where ci is forced to 1
DEC	--srca	srca + ffh + ci, where ci is forced to 0
ADD	srca + srcb	srca + srcb + ci, where ci is forced to 0
SUB	srca – srcb	srca + ~srcb + ci, where ci is forced to 1

In addition to this default arithmetic mode for carry operation, there are three additional carry options. The CI SELA and CI SELB configuration bits determine the carry in for a given cycle. Dynamic configuration RAM selects either the A or B configuration on a cycle-by-cycle basis. The options are defined in Table 16-9.

Table 16-9. Additional Carry In Functions

CI SEL A CI SEL B	Carry Mode	Description
00	Default	Default arithmetic mode as described in Table 16-8.
01	Registered	Carry Flag, result of the carry from the previous cycle. This mode is used to implement add with carry and subtract with borrow operations. It can be used in successive cycles to emulate a double precision operation.
10	Routed	Carry is generated elsewhere and routed to this input. This mode can be used to implement controllable counters.
11	Chained	Carry is chained from the previous datapath. This mode can be used to implement single cycle operations of higher precision involving two or more datapaths.

When a routed carry is used, the meaning with respect to each arithmetic function is shown in Table 16-10. Note that in the case of the decrement and subtract functions, the carry is active low (inverted).

Table 16-10. Routed Carry In Functions

Function	Carry In Polarity	Carry In Active	Carry In Inactive
INC	True	++srca	srca
DEC	Inverted	--srca	srca
ADD	True	(srca + srcb) + 1	srca + srcb
SUB	Inverted	(srca – srcb) – 1	(srca – srcb)

Carry Out

The carry out is a selectable datapath output and is derived from the currently defined MSB position, which is statically programmable. This value is also chained to the next most significant block as an optional carry in. Note that in the case of decrement and subtract functions, the carry out is inverted.

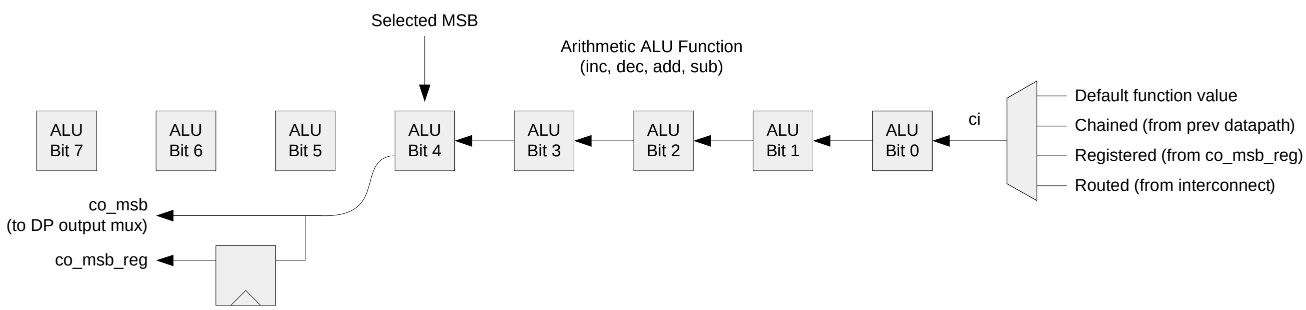
Table 16-11. Carry Out Functions

Function	Carry Out Polarity	Carry Out Active	Carry Out Inactive
INC	True	++srca == 0	srca
DEC	Inverted	--srca == –1	srca
ADD	True	srca + srcb > 255	srca + srcb
SUB	Inverted	srca – srcb < 0	(srca – srcb)

Carry Structure

Figure 16-15 shows the options for carry in, and for MSB selection for carry out generation. The registered carry out value may be selected as the carry in for a subsequent arithmetic operation. This feature can be used to implement higher precision functions in multiple cycles.

Figure 16-15. Carry Operation



Shift Operation

The shift operation occurs independent of the ALU operation, according to Table 16-12.

Table 16-12. Shift Operation Functions

Shift[1:0]	Function
00	Pass
01	Shift Left
10	Shift Right
11	Nibble Swap

A shift out value is available as a datapath output. Both shift out right (sor) and shift out left (sol_msb) share that output selection. A static configuration bit (SHIFT SEL in register CFG15) determines which shift output is used as a datapath output. When no shift is occurring, the sor and sol_msb signal is defined as the LSB or MSB of the ALU function, respectively.

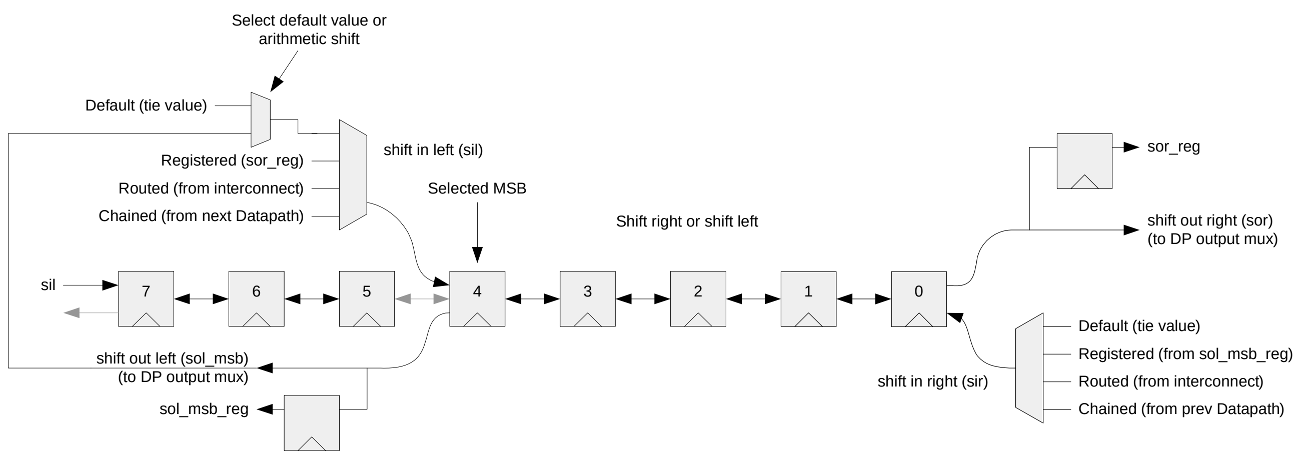
The SI SELA and SI SELB configuration bits determine the shift in data for a given operation. Dynamic configuration RAM selects the A or B configuration on a cycle-by-cycle basis. Shift in data is only valid for left and right shift; it is not used for pass and nibble swap. Table 16-13 shows the selections and usage that apply to both left and right shift directions.

Table 16-13. Shift In Functions

SI SEL A SI SEL B	Shift In Source	Description
00	Default/Arithmetic	The default input is the value of the DEF SI configuration bit (fixed 1 or 0). However, if the MSB SI bit is set, then the default input is the currently defined MSB (for right shift only).
01	Registered	The shift in value is driven by the current registered shift out value (from the previous cycle). The shift left operation uses the last shift out left value. The shift right operation uses the last shift out right value.
10	Routed	Shift in is selected from the routing channel (the SI input).
11	Chained	Shift in left is routed from the right datapath neighbor and shift in right is routed from the left datapath neighbor.

The shift out left data comes from the currently defined MSB position, and the data that is shifted in from the left (in a shift right operation) goes into the currently defined MSB position. Both shift out data (left or right) are registered and can be used in a subsequent cycle. This feature can be used to implement a higher precision shift in multiple cycles.

Figure 16-16. Shift Operation



Note that the bits that are isolated by the MSB selection are still shifted. In the example shown, bit 7 still shifts in the sil value on a right shift and bit 5 shifts in bit 4 on a left shift. The shift out either right or left from the isolated bits is lost.

ALU Masking Operation

An 8-bit mask register in the UDB static configuration register space defines the masking operation. In this operation, the output of the ALU is masked (ANDed) with the value in the mask register. A typical use for the ALU mask function is to implement free-running timers and counters in power of two resolutions.

16.2.2.5 Datapath Inputs and Multiplexing

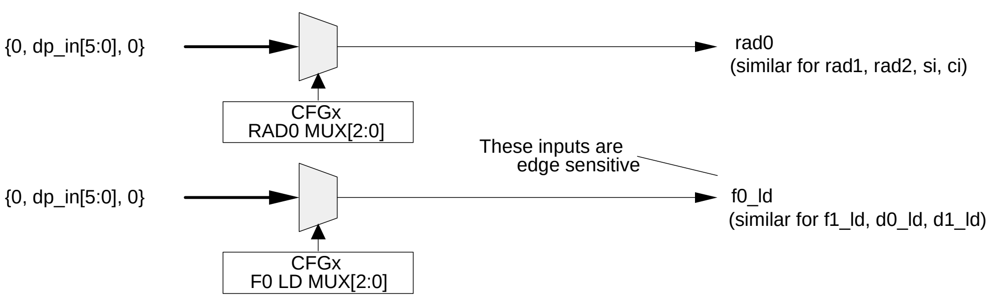
The datapath has a total of nine inputs, as shown in Table 16-14, including six inputs from the channel routing. These consist of the configuration RAM address, FIFO and data register load control signals, and the data inputs shift in and carry in.

Table 16-14. Datapath Inputs

Input	Description
RAD2 RAD1 RAD0	Asynchronous dynamic configuration RAM address. There are eight 16-bit words, which are user-programmable. Each word contains the datapath control bits for the current cycle. Sequences of instructions can be controlled by these address inputs.
F0 LD F1 LD	When asserted in a given cycle, the selected FIFO is loaded with data from one of the A0 or A1 accumulators or from the output of the ALU. The source is selected by the Fx INSEL[1:0] configuration bits. This input is edge sensitive. It is sampled at the datapath clock; when a '0' to '1' transition is detected, a load occurs at the subsequent clock edge.
D0 LD D1 LD	When asserted in a given cycle, the Dx register is loaded from associated FIFO Fx. This input is edge sensitive. It is sampled at the datapath clock; when a '0' to '1' transition is detected, a load occurs at the subsequent clock edge.
SI	This is a data input value that can be used for either shift in left or shift in right.
CI	This is the carry in value used when the carry in select control is set to "routed carry."

As shown in Figure 16-17, each input has a 6-to-1 multiplexer, therefore, all inputs are permutable. Inputs are handled in one of two ways, either level sensitive or edge sensitive. RAM address, shift in and data in values are level sensitive; FIFO and data register load signals are edge sensitive.

Figure 16-17. Datapath Input Select



16.2.2.6 CRC/PRS Support

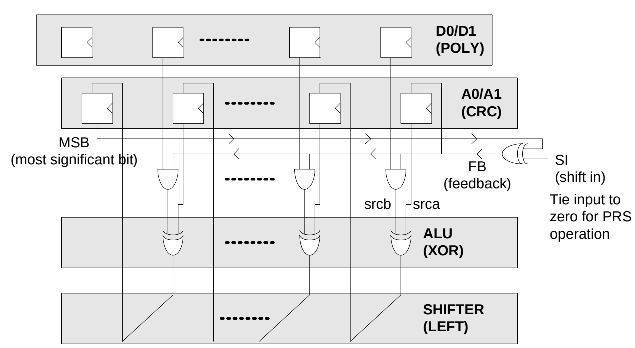
The datapath can support cyclic redundancy checking (CRC) and pseudo random sequence (PRS) generation. Chaining signals are routed between datapath blocks to support CRC/PRS bit lengths of longer than eight bits.

The most significant bit (MSB) of the most significant block in the CRC/PRS computation is selected and routed (and chained across blocks) to the least significant block. The MSB is then XORed with the data input (SI data) to provide the feedback (FB) signal. The FB signal is then routed (and chained across blocks) to the most significant block. This feedback value is used in all blocks to gate the XOR of the polynomial (from the Data0 or Data1 register) with the current accumulator value.

Figure 16-18 shows the structural configuration for the CRC operation. The PRS configuration is identical except that the shift in (SI) is tied to '0'. In the PRS configuration, D0 or D1 contain the polynomial value, while A0 or A1 contain the initial (seed) value and the CRC residual value at the end of the computation.

To enable CRC operation, the CFB_EN bit in the dynamic configuration RAM must be set to '1'. This enables the AND of SRCB ALU input with the CRC feedback signal. When set to zero, the feedback signal is driven to '1', which allows for normal arithmetic operation. Dynamic control of this bit on a cycle-by-cycle basis gives the capability to interleave a CRC/PRS operation with other arithmetic operations.

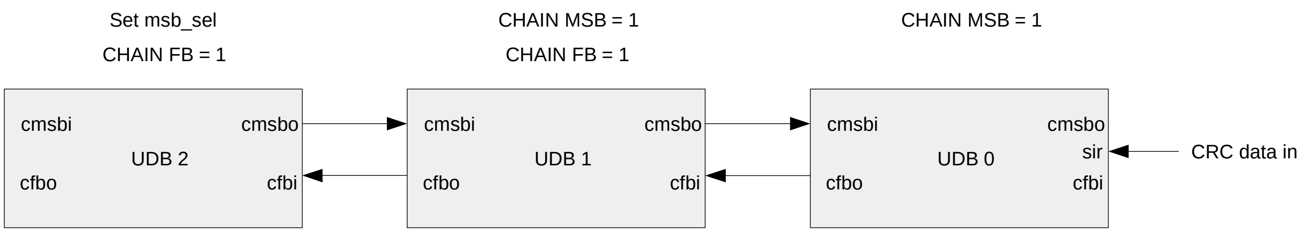
Figure 16-18. CRC Functional Structure



CRC/PRS Chaining

Figure 16-19 illustrates an example of CRC/PRS chaining across three UDBs. This scenario can support a 17- to 24-bit operation. The chaining control bits are set according to the position of the datapath in the chain as shown in the figure.

Figure 16-19. CRC/PRS Chaining Configuration



The CRC/PRS feedback signal (cfbo, cfbi) is chained as follows:

- If a given block is the least significant block, then the feedback signal is generated in that block from the built-in logic that takes the shift in from the right (sir) and XORs it with the MSB signal. (For PRS, the "sir" signal is tied to '0'.)
- If a given block is not the least significant block, the CHAIN FB configuration bit must be set and the feedback is chained from the previous block in the chain.

The CRC/PRS MSB signal (cmsbo, cmsbi) is chained as follows:

- If a given block is the most significant block, the MSB bit (according to the polynomial selected) is configured using the MSB_SEL configuration bits.
- If a given block is not the most significant block, the CHAIN MSB configuration bit must be set and the MSB signal is chained from the next block in the chain.

CRC/PRS Polynomial Specification

As an example of how to configure the polynomial for programming into the associated D0/D1 register, consider the CCITT CRC-16 polynomial, which is defined as $x^{16} + x^{12} + x^5 + 1$. The method for deriving the data format from the polynomial is shown in Figure 16-20.

The X^0 term is inherently always '1' and therefore does not need to be programmed. For each of the remaining terms in the polynomial, a '1' is set in the appropriate position in the alignment shown.

Note This polynomial format is slightly different from the format normally specified in Hex. For example, the CCITT CRC16 polynomial is typically denoted as 1021H. To convert to the format required for datapath operation, shift right by one and add a '1' in the MSB bit. In this case, the correct polynomial value to load into the D0 or D1 register is 8810H.

Figure 16-20. CCITT CRC16 Polynomial Format

X^{16}	X^{15}	X^{14}	X^{13}	X^{12}	X^{11}	X^{10}	X^9	X^8	X^7	X^6	X^5	X^4	X^3	X^2	X^1	X^0
X^{16}	+			X^{12}	+						X^5	+				1
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	

CCITT 16-Bit Polynomial is 0x8810

Example CRC/PRS Configuration

The following is a summary of CRC/PRS configuration requirements, assuming that D0 is the polynomial and the CRC/PRS is computed in A0:

1. Select a suitable polynomial and write it into D0.
2. Select a suitable seed value (for example, all zeros for CRC, all ones for PRS) and write it into A0.
3. Configure chaining if necessary.
4. Select the MSB position as defined in the polynomial from the MSB_SEL static configuration register bits and set the MSB_EN register bit.
5. Configure the dynamic configuration RAM word fields:
6. Select D0 as the ALU "SRCB" (ALU B Input Source)
7. Select A0 as the ALU "SRCA" (ALU A Input Source)
8. Select "XOR" for the ALU function
9. Select "SHIFT LEFT" for the SHIFT function
10. Select "CFB_EN" to enable the support for CRC/PRS
11. Select ALU as the A0 write source

If a CRC operation, configure "shift in right" for input data from routing and supply input on each clock. If a PRS operation, tie "shift in right" to '0'.

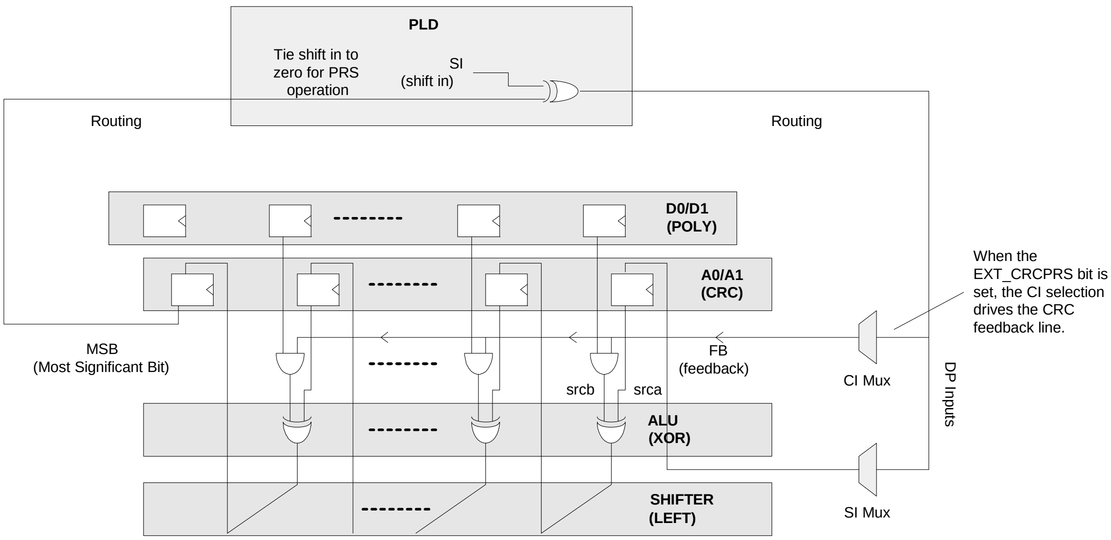
Clocking the UDB with this configuration generates the required CRC or outputs the MSB, which may be output to the routing for the PRS sequence.

External CRC/PRS Mode

A static configuration bit may be set (EXT CRCPRS) to enable support for external computation of a CRC or PRS. As shown in Figure 16-21, computation of the CRC feedback is done in a PLD block. When the bit is set, the CRC feedback signal is driven directly from the CI (Carry In) datapath input selection mux, bypassing the internal computation. The figure shows a simple configuration that supports up to an 8-bit CRC or PRS. Normally the built-in circuitry is used, but this feature gives the capability for more elaborate configurations, such as up to a 16-bit CRC/PRS function in one UDB using time division multiplexing.

In this mode, the dynamic configuration RAM bit CFB_EN still controls whether the CRC feedback signal is ANDed with the SRCB ALU input. Therefore, as with the built-in CRC/PRS operation, the function can be interleaved with other functions if required.

Figure 16-21. External CRC/PRS Mode



16.2.2.7 Datapath Outputs and Multiplexing

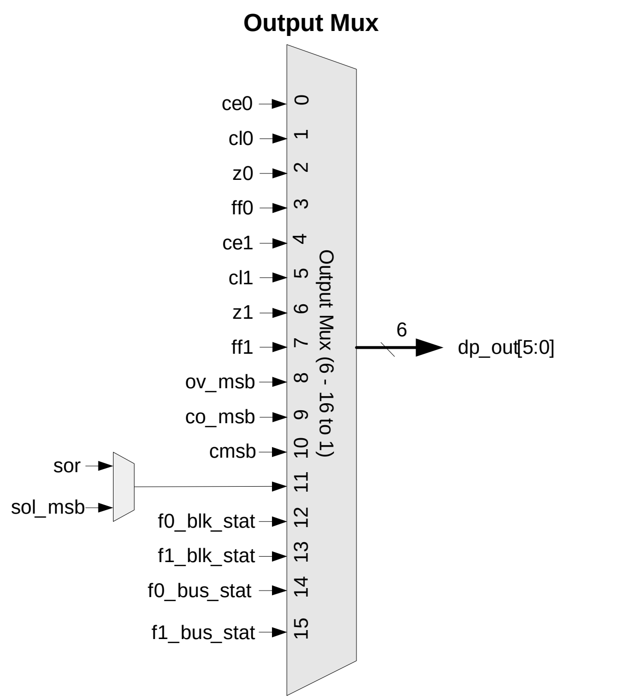
Conditions are generated from the registered accumulator values, ALU outputs, and FIFO status. These conditions can be driven to the digital routing for use in other UDB blocks, for use as interrupts, or to I/O pins. The 16 possible conditions are shown in Table 16-15.

Table 16-15. Datapath Condition Generation

Name	Condition	Chain	Description
ce0	Compare Equal	Y	A0 == D0
cl0	Compare Less Than	Y	A0 < D0
z0	Zero Detect	Y	A0 == 00h
ff0	Ones Detect	Y	A0 == FFh
ce1	Compare Equal	Y	A1 or A0 == D1 or A0 (dynamic selection)
cl1	Compare Less Than	Y	A1 or A0 < D1 or A0 (dynamic selection)
z1	Zero Detect	Y	A1 == 00h
ff1	Ones Detect	Y	A1 == FFh
ov_msb	Overflow	N	Carry(msb) ^ Carry(msb–1)
co_msb	Carry Out	Y	Carry out of MSB defined bit
cmsb	CRC MSB	Y	MSB of CRC/PRS function
so	Shift Out	Y	Selection of shift output
f0_blk_stat	FIFO0 Block Status	N	Definition depends on FIFO configuration
f1_blk_stat	FIFO1 Block Status	N	Definition depends on FIFO configuration
f0_bus_stat	FIFO0 Bus Status	N	Definition depends on FIFO configuration
f1_bus_stat	FIFO1 Bus Status	N	Definition depends on FIFO configuration

There are a total of six datapath outputs. As shown in Figure 16-22, each output has a 16-1 multiplexer that allows any of these 16 signals to be routed to any of the datapath outputs.

Figure 16-22. Output Mux Connections



Compares

There are two compares, one of which has fixed sources (Compare 0) and the other has dynamically selectable sources (Compare 1). Each compare has an 8-bit statically programmed mask register, which enables the compare to occur in a specified bit field. By default, the masking is off (all bits are compared) and must be enabled.

Comparator 1 inputs are dynamically configurable. As shown in Table 16-16, there are four options for Comparator 1, which applies to both the "less than" and the "equal" conditions. The CMP SELA and CMP SELB configuration bits determine the possible compare configurations. A dynamic RAM bit selects one of the A or B configurations on a cycle-by-cycle basis.

Table 16-16. Compare Configuration

CMP SEL A CMP SEL B	Comparator 1 Compare Configuration
00	A1 Compare to D1
01	A1 Compare to A0
10	A0 Compare to D1
11	A0 Compare to A0

Compare 0 and Compare 1 are independently chainable to the conditions generated in the previous datapath (in addressing order). Whether to chain compares is statically specified in UDB configuration registers. Figure 16-23 illustrates compare equal chaining, which is just an ANDing of the compare equal in this block with the chained input from the previous block.

Figure 16-23. Compare Equal Chaining

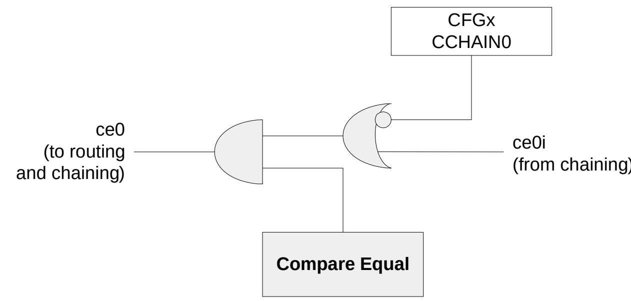
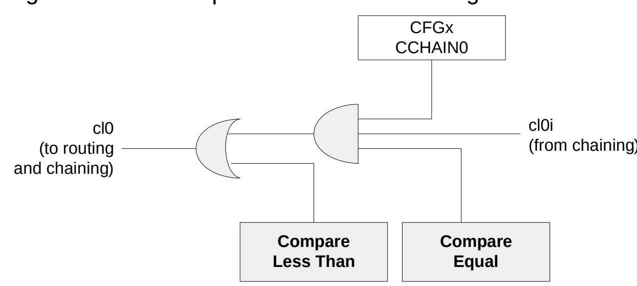


Figure 16-24 illustrates compare less than chaining. In this case, the “less than” is formed by the compare less than output in this block, which is unconditional. This is ORed with the condition where this block is equal, and the chained input from the previous block is asserted as less than.

Figure 16-24. Compare Less Than Chaining



All Zeros and All Ones Detect

Each accumulator has dedicated all zeros detect and all ones detect. These conditions are statically chainable as specified in UDB configuration registers. Whether to chain these conditions is statically specified in UDB configuration registers. Chaining of zero detect is the same concept as the compare equal. Successive chained data is ANDed if the chaining is enabled.

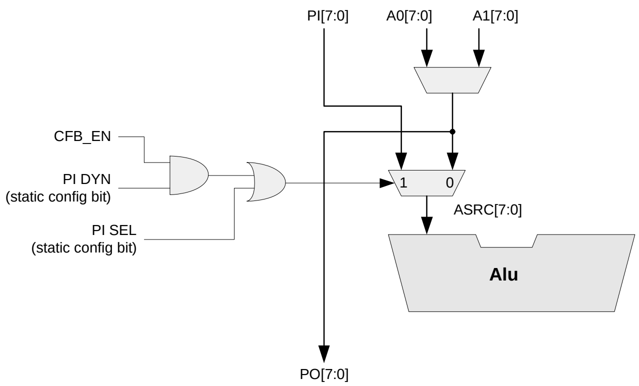
Overflow

Overflow is defined as the XOR of the carry into the MSB and the carry out of the MSB. The computation is done on the currently defined MSB as specified by the MSB_SEL bits. This condition is not chainable, however the computation is valid when done in the most significant datapath of a multi-precision function as long as the carry is chained between blocks.

16.2.2.8 Datapath Parallel Inputs and Outputs

As shown in Figure 16-25, the datapath Parallel In (PI) and Parallel Out (PO) signals give limited capability to bring routed data directly into and out of the Datapath. Parallel Out signals are always available for routing as the ALU asrc selection between A0 and A1.

Figure 16-25. Datapath Parallel In/Out

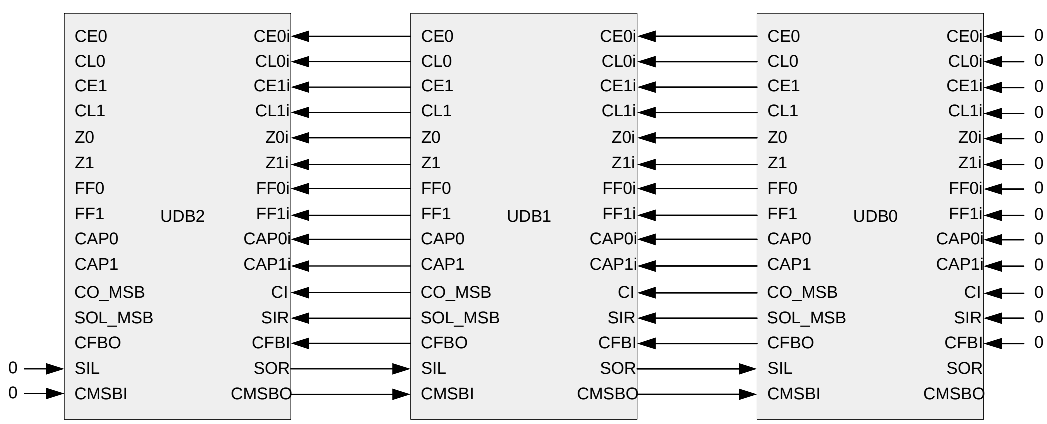


Parallel In needs to be selected for input to the ALU. The two options available are static operation or dynamic operation. For static operation, the PI SEL bit forces the ALU asrc to be PI. The PI DYN bit is used to enable the PI dynamic operation. When it is enabled, and assuming the PI SEL is 0, the PI multiplexer may then be controlled by the CFB_EN dynamic control bit. The primary function of the CFB_EN bit is to enable PRS/CRC functionality.

16.2.2.9 Datapath Chaining

Each datapath block contains an 8-bit ALU, which is designed to chain carries, shifted data, capture triggers, and conditional signals to the nearest neighbor datapaths, to create higher precision arithmetic functions and shifters. These chaining signals, which are dedicated signals, allow single-cycle 16-, 24- and 32-bit functions to be efficiently mplemented without the timing uncertainty of channel routing resources. In addition, the capture chaining supports the ability to perform an atomic read of the accumulators in chained blocks. As shown in Figure 16-26, all generated conditional and capture signals chain in the direction of least significant to most significant blocks. Shift left also chains from least to most significant. Shift right chains from most to least significant. The CRC/PRS chaining signal for feedback chains least to most significant; the MSB output chains from most to least significant.

Figure 16-26. Datapath Chaining Flow

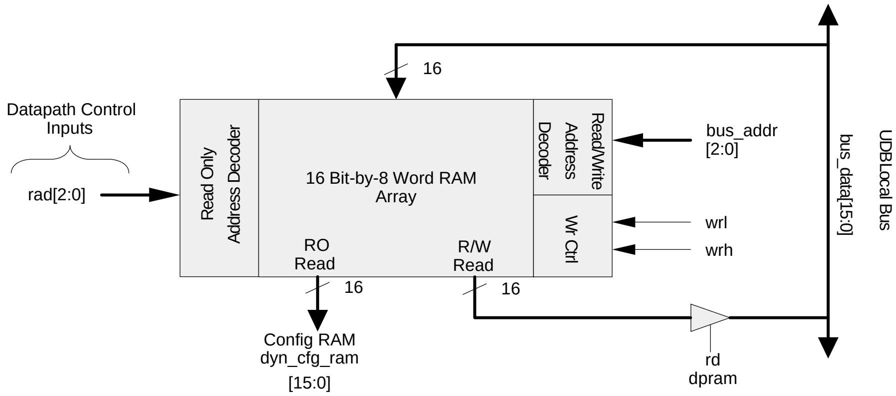


16.2.2.10 Dynamic Configuration RAM

Each datapath contains a 16 bit-by-8 word dynamic configuration RAM, which is shown in Figure 16-27. The purpose of this RAM is to control the datapath configuration bits on a cycle-by-cycle basis, based on the clock selected for that datapath. This RAM has synchronous read and write ports for purposes of loading the configuration via the system bus.

An additional asynchronous read port is provided as a fast path to output these 16-bit words as control bits to the datapath. The asynchronous address inputs are selected from datapath inputs and can be generated from any of the possible signals on the channel routing, including I/O pins, PLD outputs, control block outputs, or other datapath outputs. The primary purpose of the asynchronous read path is to provide a fast single-cycle decode of datapath control bits.

Figure 16-27. Configuration RAM I/O



The fields of this dynamic configuration RAM word are shown here. A description of the usage of each field follows.

Register	Address	15	14	13	12	11	10	9	8
CFGRAM	61h - 6Fh (odd)	FUNC[2:0]			SRCA	SRCB[1:0]		SHIFT[1:0]	

Register	Address	7	6	5	4	3	2	1	0
CFGRAM	60h - 6Eh (even)	A0 WR SRC[1:0]		A1 WR SRC[1:0]		CFB EN	CI SEL	SI SEL	CMP SEL

Table 16-17. Dynamic Configuration Quick Reference

Field	Bits	Parameter	Values
FUNC[2:0]	3	ALU Function	000 PASS 001 INC SRCA 010 DEC SRCA 011 ADD 100 SUB 101 XOR 110 AND 111 OR
SRCA	1	ALU A Input Source	0 A0 1 A1
SRCB	2	ALU B Input Source	00 D0 01 D1 10 A0 11 A1
SHIFT[1:0]	2	SHIFT Function	00 PASS 01 Left Shift 10 Right Shift 11 Nibble Swap
A0 WR SRC[1:0]	2	A0 Write Source	00 None 01 ALU 10 D0 11 F0
A1 WR SRC[1:0]	2	A1 Write Source	00 None 01 ALU 10 D1 11 F1
CFB EN	1	CRC Feedback Enable	0 Enable 1 Disable
CI SEL	1	Carry In Configuration Select	0 ConfigA 1 ConfigB[a]
SI SEL	1	Shift In Configuration Select	0 ConfigA 1 ConfigB[a]
CMP SEL	1	Compare Configuration Select	0 ConfigA 1 ConfigB[a]

a. For CI, SI, and CMP, the RAM fields select between two predefined static settings. See Static Register Configuration

16.2.3 Status and Control Module

Figure 16-28 shows a high-level view of the Status and Control module. The Control register drives into the routing to provide firmware control inputs to UDB operation. The Status register read from routing provides firmware a method of monitoring the state of UDB operation.

Figure 16-28. Status and Control Registers

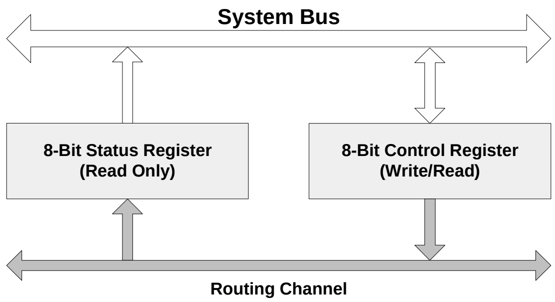
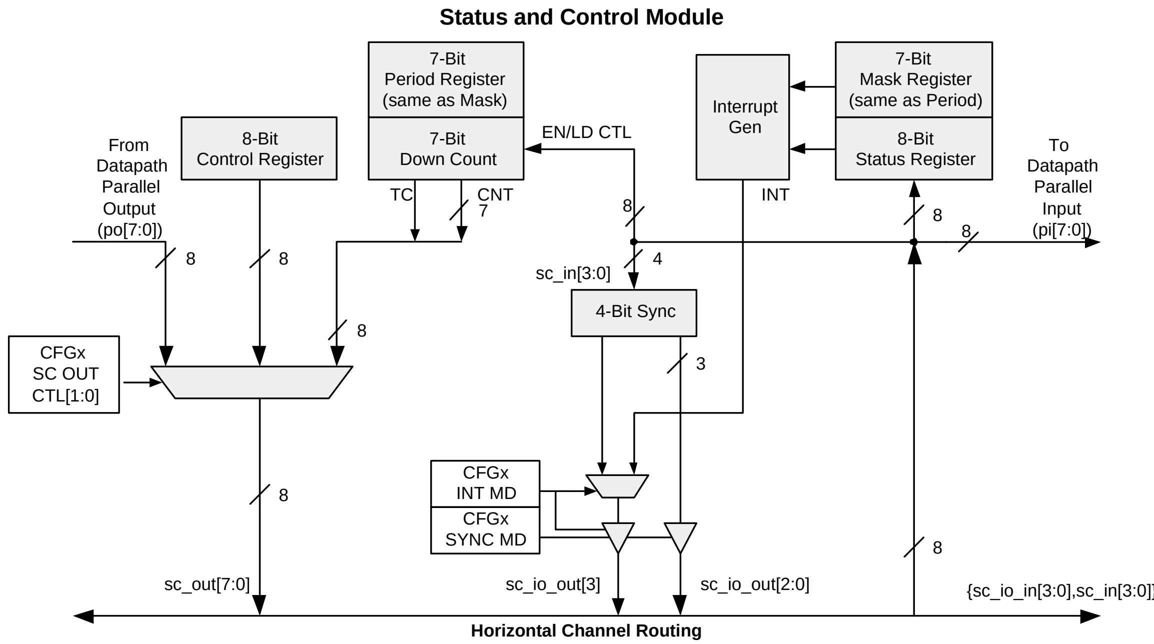


Figure 16-29 shows a more detailed view of the Status and Control module. The primary purpose of this block is to coordinate CPU firmware interaction with internal UDB operation. However, due to its rich connectivity to the routing matrix, this block may be configured to perform other functions.

Figure 16-29. Status and Control Module



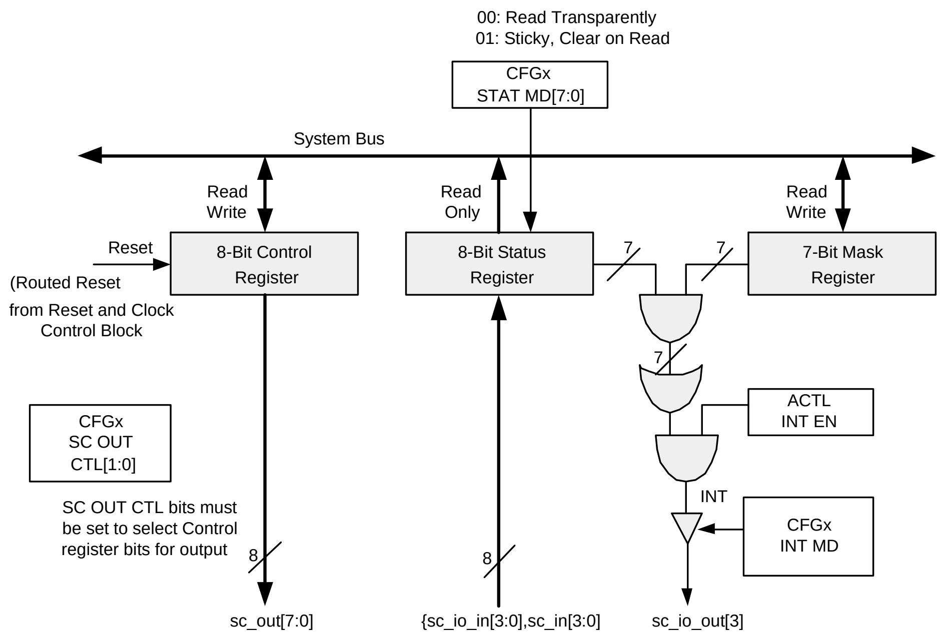
Modes of operation include:

- **Status Input** – The state of routing signals can be input and captured as status and read by the CPU.
- **Control Output** – The CPU can write to the control register to drive the state of the routing.
- **Parallel Input** – To datapath parallel input.
- **Parallel Output** – From datapath parallel output.
- **Counter Mode** – In this mode, the control register operates as a 7-bit down counter with programmable period and automatic reload. Routing inputs can be configured to control both the enable and reload of the counter. When this mode is enabled, control register operation is not available.
- **Sync Mode** – In this mode, the status register operates as a 4-bit double synchronizer. When this mode is enabled, status register operation is not available.

16.2.3.1 Status and Control Mode

When operating in status and control mode, this module functions as a status register, interrupt mask register, and control register in the configuration shown in Figure 16-30.

Figure 16-30. Status and Control Operation



Status Register Operation

One 8-bit, read-only status register is available for each UDB. Inputs to this register come from any signal in the digital routing fabric. The status register is nonretention; it loses its state across sleep intervals and is reset to 0x00 on wakeup. Each bit can be independently programmed to operate in one of two ways, as shown in Table 16-18.

Table 16-18. Status Register

STAT MD	Description
0	Transparent read. A read returns the current value of the routed signal
1	Sticky, clear on read. A high on the input is sampled and captured. It is cleared when the register is read.

An important feature of the status register clearing operation is to note that the clear of status is only applied to the bits that are set. This allows other bits that are not set to continue to capture status, so that a coherent view of the process can be maintained.

Transparent Status Read

By default, a CPU read of this register transparently reads the state of the associated routing. This mode can be used for a transient state that is computed and registered internally in the UDB.

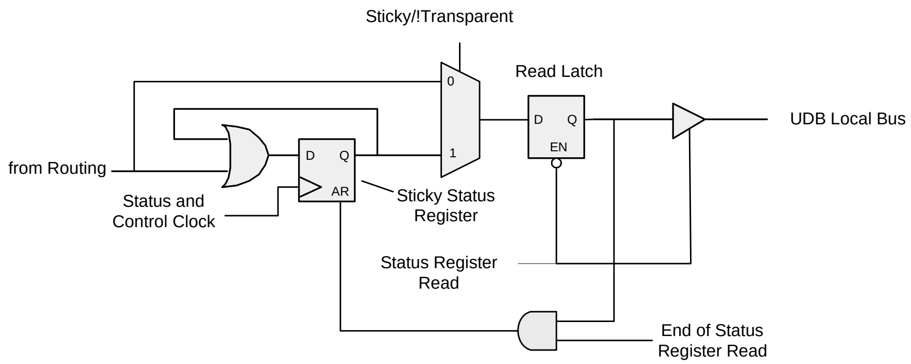
Sticky Status, with Clear on Read

In this mode, the status register inputs are sampled on each cycle of the status and control clock. If the signal is high in a given sample, it is captured in the status bit and remains high, regardless of the subsequent state of the input. When the CPU reads the status register the bit is cleared. The status register clearing is independent of mode and occurs even if the UDB clock is disabled; it is based on the bus clock and occurs as part of the read operation.

Status Latching During Read

Figure 16-31 shows the structure of the status read logic. The sticky status register is followed by a latch, which latches the status register data and holds it stable during the duration of the read cycle, regardless of the number of wait states in a given read.

Figure 16-31. Status Read Logic



Interrupt Generation

In most functions, interrupt generation is tied to the setting of status bits. As shown in Figure 16-31, this feature is built into the status register logic as the masking and OR reduction of status. Only the lower seven bits of status input can be used with the built-in interrupt generation circuitry. The most significant bit is typically used as the interrupt output and may be routed to the interrupt controller through the digital routing. In this configuration, the MSB of the status register is read as the state of the interrupt bit.

16.2.3.2 Control Register Operation

One 8-bit control register is available for each UDB. This operates as a standard read/write register on the system bus, where the output of these register bits are selectable as drivers into the digital routing fabric.

The Control register is nonretention; it loses its contents across sleep intervals and is reset to 0x00 on wakeup.

Control Register Operating Modes

Three modes are available that may be configured on a bit-by-bit basis. The configuration is controlled by the concatenation of the bits of the two 8-bit registers CTL_MD1[7:0] and CTL_MD0[7:0]. For example, {CTL_MD1[0],CTL_MD0[0]} controls the mode for Control Register bit 0, as shown in Table 16-19.

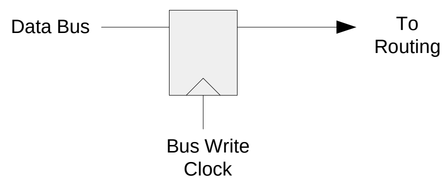
Table 16-19. Mode for Control Register Bit 0

CTL MD	Description
00	Direct mode
01	Sync mode
10	Double sync mode
11	Pulse mode

Control Register Direct Mode

The default mode is Direct mode. As shown in Figure 16-32, when the Control Register is written by the CPU the output of the control register is driven directly to the routing on that write cycle.

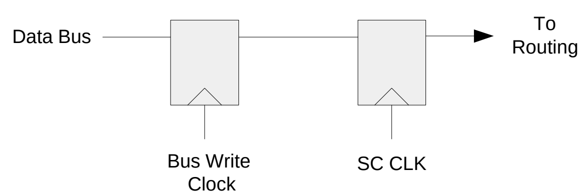
Figure 16-32. Control Register Direct Mode



Control Register Sync Mode

In Sync mode, as shown in Figure 16-33, the control register output is driven by a re-sampling register clocked by the currently selected Status and Control (SC) clock. This allows the timing of the output to be controlled by the selected SC clock, rather than the bus clock.

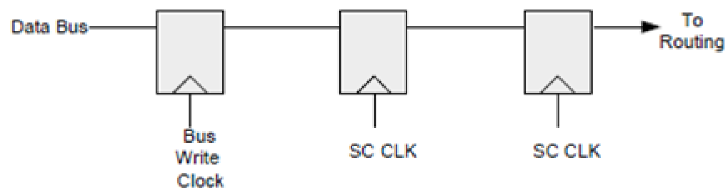
Figure 16-33. Control Register Sync Mode



Control Register Double Sync Mode

In Double Sync mode, as shown in Figure 16-34, a second register clocked by the selected SC clock is added after the re-sampling register. This allows the circuit to perform robustly when bus clock and SC clock are asynchronous.

Figure 16-34. Control Register Double Sync Mode



Control Register Pulse Mode

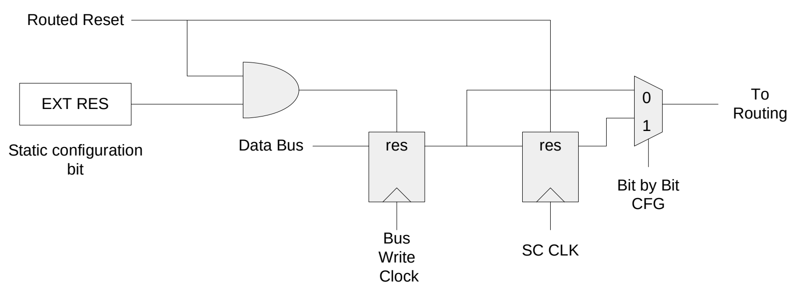
Pulse mode is similar to Sync mode in that the control bit is re-sampled by the SC clock; the pulse starts on the first SC clock cycle following the bus write cycle. The output of the control bit is asserted for one full SC clock cycle. At the end of this clock cycle, the control bit is automatically reset.

With this mode of operation, firmware can write a ‘1’ to a control register bit to generate a pulse. After it is written as a ‘1’, it is read back by firmware as a ‘1’ until the completion of the pulse, after which it is read back as a ‘0’. The firmware can then write another ‘1’ to start another pulse. A new pulse cannot be generated until the previous one is completed. Therefore, the maximum frequency of pulse generation is every other SC clock cycle.

Control Register Reset

The control register has two reset modes, controlled by the EXT RES configuration bit, as shown in Figure 16-35. When EXT RES is 0 (the default) then in sync or pulse mode the routed reset input resets the synced output but not the actual control bit. When EXT RES is 1 then the routed reset input resets both the control bit and the synced output.

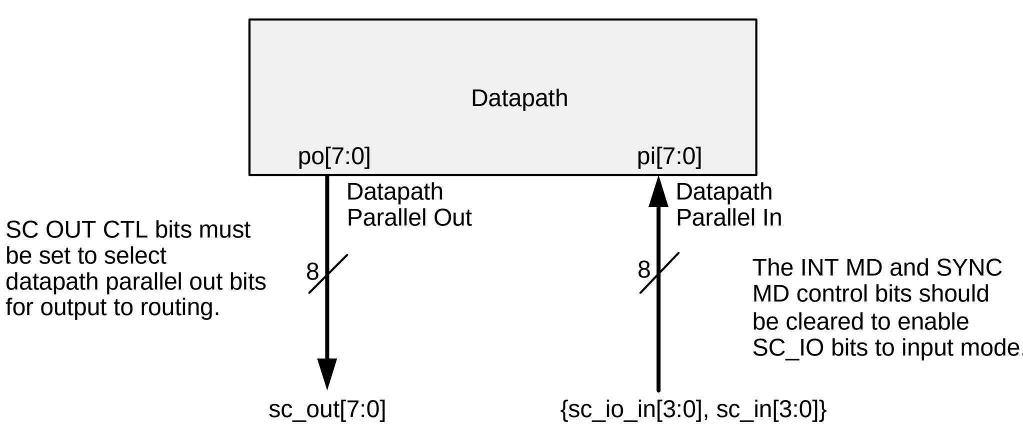
Figure 16-35. Control Register Reset



16.2.3.3 Parallel Input/Output Mode

In this mode, as Figure 16-36 shows, the status and control routing is connected to the datapath parallel in and parallel out signals. To enable this mode, the SC OUT configuration bits are set to select datapath parallel out. The parallel input connection is always available, but these routing connections are shared with the status register inputs, counter control inputs, and the interrupt output.

Figure 16-36. Parallel Input/Output Mode



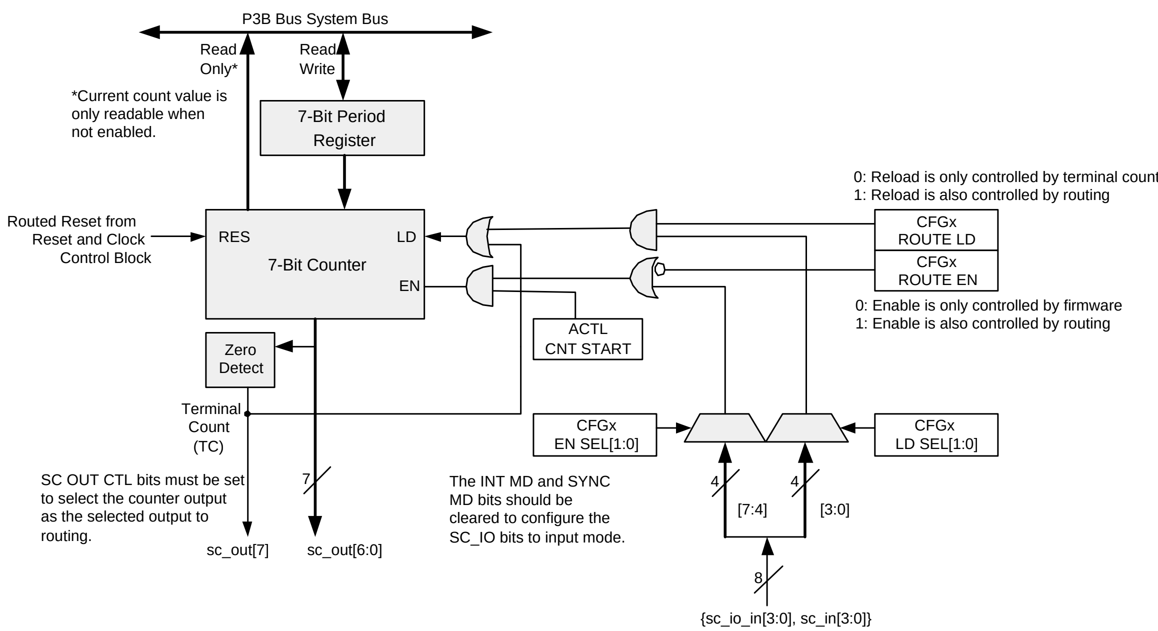
16.2.3.4 Counter Mode

As shown in Figure 16-37, when the block is in counter mode, a 7-bit down counter is exposed for use by UDB internal operation or firmware applications. This counter has the following features:

- A 7-bit read/write period register.
- A 7-bit read/write count register. It can be accessed only when the counter is disabled.
- Automatic reload of the period to the count register on terminal count (0).
- A firmware control bit in the Auxiliary Control Working register called CNT START, to start and stop the counter. (This is an overriding enable and must be set for optional routed enable to be operational.)
- Selectable bits from the routing for optional dynamic control of the counter enable and load functions:
  - EN, routed enable to start or stop counting.
  - LD, routed load signal to force the reload of period. When this signal is asserted, it overrides a pending terminal count. It is level sensitive and continues to load the period while asserted.
- The 7-bit count may be driven to the routing fabric as sc_out[6:0].
- The terminal count may be driven to the routing fabric as sc_out[7].
- In default mode, the terminal count is registered. In alternate mode the terminal count is combinational.
- In default mode, the routed enable, if used, must be asserted for routed load to operate. In alternate mode the routed enable and routed load signals operate independently.

To enable the counter mode, the SC_OUT_CTl[1:0] bits must be set to counter output. In this mode the normal operation of the control register is not available. The status register can still be used for read operations, but should not be used to generate an interrupt because the mask register is reused as the counter period register. The Period register is retention and maintains its state across sleep intervals. For a period of N clocks, the period value of N–1 should be loaded. N = 1 (period of 0) is not supported as a clock divide value, and results in the terminal count output of a constant 1.The use of SYNC mode depends on whether the dynamic control inputs (LD/EN) are used. If they are not used, SYNC mode is unaffected. If they are used, SYNC mode is unavailable.

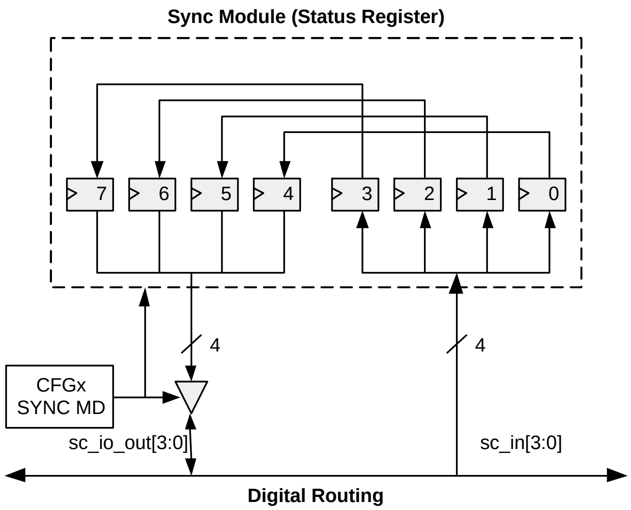
Figure 16-37. Counter Mode



16.2.3.5 Sync Mode

As shown in Figure 16-38, the status register can operate as a 4-bit double synchronizer, clocked by the current SC_CLK, when the SYNC MD bit is set. This mode may be used to implement local synchronization of asynchronous signals, such as GPIO inputs. When enabled, the signals to be synchronized are selected from SC_IN[3:0], the outputs are driven to the SC_IO_OUT[3:0] pins, and SYNC MD automatically puts the SC_IO pins into output mode. When in this mode, the normal operation of the status register is not available, and the status sticky bit mode is forced off, regardless of the control settings for this mode. The control register is not affected by the mode. The counter can still be used with limitations. No dynamic inputs (LD/EN) to the counter can be enabled in this mode.

Figure 16-38. Sync Mode



16.2.3.6 *Status and Control Clocking*

The status and control registers require a clock selection for any of the following operating modes:

- Status register with any bit set to sticky, clear on read mode.
- Control register in counter mode.
- Sync mode.

The clock for this is allocated in the reset and clock control module. See Reset and Clock Control Module on page 153.

16.2.3.7 *Auxiliary Control Register*

The read-write Auxiliary Control register is a special register that controls fixed function hardware in the UDB. This register allows CPU to dynamically control the interrupt, FIFO, and counter operation. The register bits and descriptions are as follows:

Auxiliary Control Registers							
7	6	5	4	3	2	1	0
		CNT START	INT EN	FIFO1 LVL	FIFO0 LVL	FIFO1 CLR	FIFO0 CLR

FIFO0 Clear, FIFO1 Clear

The FIFO0 CLR and FIFO1 CLR bits are used to reset the state of the associated FIFO. When a '1' is written to these bits, the state of the associated FIFO is cleared. These bits must be written back to '0' to allow FIFO operation to continue. When these bits are left asserted, the FIFOs operate as simple one-byte buffers, without status.

FIFO0 Level, FIFO1 Level

The FIFO0 LVL and FIFO1 LVL bits control the level at which the 4-byte FIFO asserts bus status (when the bus is either reading or writing to the FIFO) to be asserted. The meaning of FIFO bus status depends on the configured direction, as shown in Table 16-20.

Table 16-20. FIFO Level Control Bits

FIFOx LVL	Input Mode (Bus is Writing FIFO)	Output Mode (Bus is Reading FIFO)
0	Not Full At least 1 byte can be written	Not Empty At least 1 byte can be read
1	At Least Half Empty At least 2 bytes can be written	At Least Half Full At least 2 bytes can be read

Interrupt Enable

When the status register's generation logic is enabled, the INT EN bit gates the resulting interrupt signal.

Count Start

The CNT START bit may be used to enable and disable the counter (only valid when the SC_OUT_CTL[1:0] bits are configured for counter output mode).

16.2.3.8 *Status and Control Register Summary*

Table 16-21 summarizes the function of the status and control registers. Note that the control and mask registers are shared with the count and period registers and the meaning of these registers is mode dependent.

Table 16-21. Status, Control Register Function Summary

Mode	Control/Count	Status/SYNC	Mask/Period
Control	Control Out	Status In or SYNC	Status Mask
Control	Count Out		Count Period[a]
Status	Control Out or Count Out	Status In	Status Mask
SYNC		SYNC	NA[b]

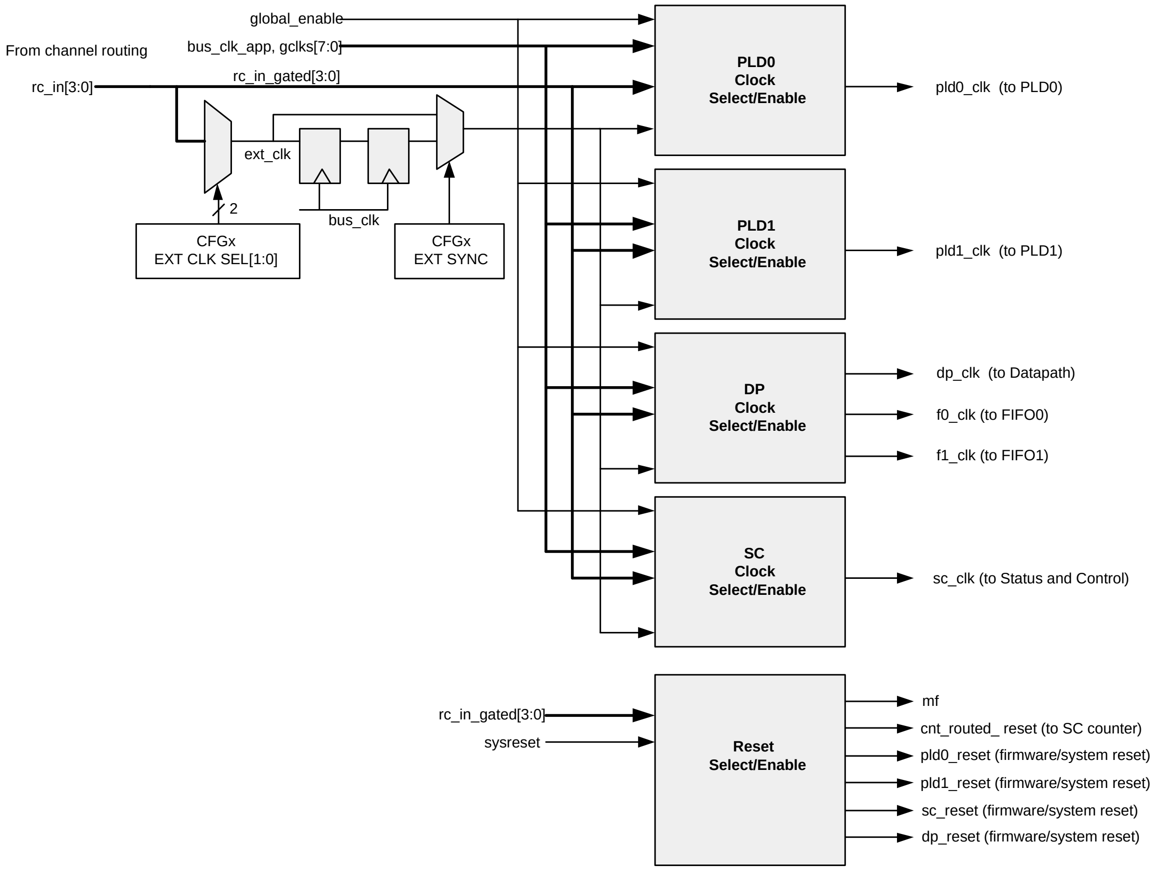
a. Note that in counter mode, the mask register is operating as a period register and cannot function as a mask register. Therefore, interrupt output is not available when counter mode is enabled.
b. Note that in SYNC mode, the status register function is not available, and therefore, the mask register is unusable. However, it can be used as a period register for count mode.

16.2.4 Reset and Clock Control Module

The primary function of the reset and clock block is to select a clock from the available global system clocks or bus clock for each of the PLDs, the datapath, and the status and control block. It also supplies dynamic and firmware-based resets to the UDB blocks. As shown in Figure 16-39, there are four clock control blocks, and one reset block. Four inputs are available for use from the routing matrix (RC_IN[3:0]). Each clock control block can select a clock enable source from these routing inputs, and there is also a multiplexer to select one of the routing inputs to be used as an external clock source. As shown, the external clock source selection can be optionally synchronized. There are a total of 10 clocks that can be selected for each UDB component: eight global digital clocks, bus clock, and the selected external clock (ext clk). Any of the routed input signals (rc_in) can be used as either a level sensitive or edge sensitive enable. The reset function of this block provides a routed reset for the PLD blocks and SC counter, and a firmware reset capability to each block to support reconfiguration.

The bus clock input to the reset and clock control is distinct from the system bus clock. This clock is called "bus_clk_app" because it is gated similar to the other global digital clocks and used for UDB applications. The system bus clock is only used for I/O access and is automatically gated, per access. The datapath clock generator produces three clocks: one for the datapath in general, and one for each of the FIFOs.

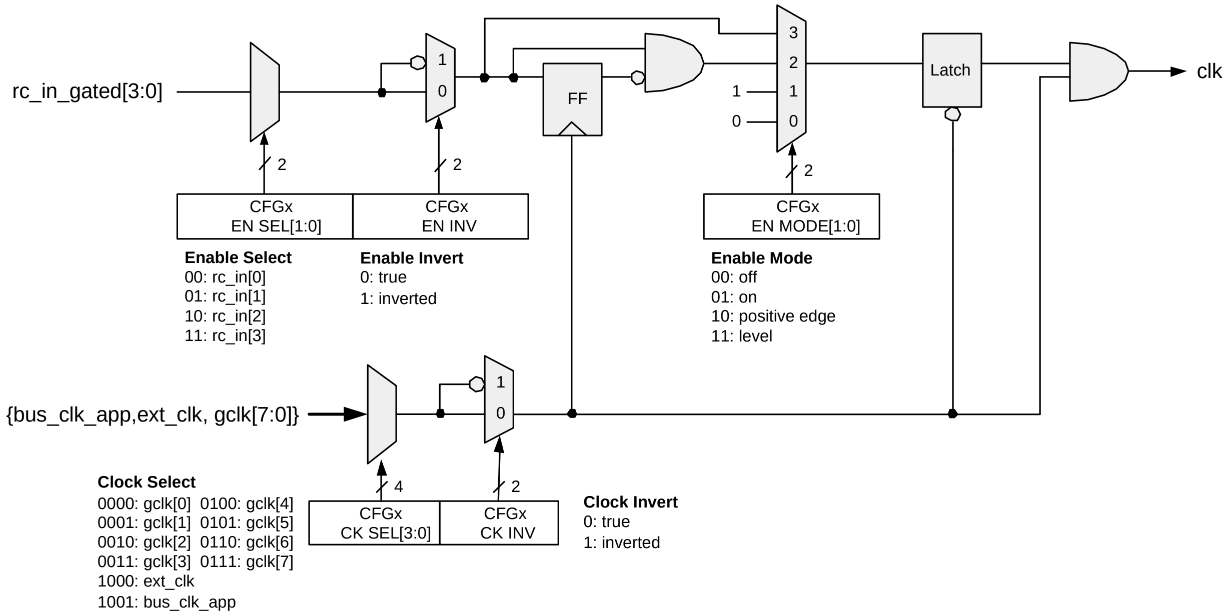
Figure 16-39. Reset and Clock Control



16.2.4.1 Clock Control

Figure 16-40 illustrates one instance of the clock selection and enable circuit. Each UDB has four of these circuits: one for each of the PLD blocks, one for the datapath, and one for the status and control block. The main components of this circuit are a global clock selection multiplexer, clock inversion, clock enable selection multiplexer, clock enable inversion, and edge detect logic.

Figure 16-40. Clock Select/Enable Control



Clock Selection

Eight global digital clocks are routed to all UDBs; any of these clocks may be selected. Global digital clocks are the output of user-selectable clock dividers. Another selection is bus clock, which is the highest frequency in the system. Called “bus_clk_app,” this signal is routed separately from the system bus clock. In addition, an external routing signal can be selected as a clock input to support direct-clocked functions such as SPI. Because application functions are mapped to arbitrary boundaries across UDBs, individual clock selection for each UDB subcomponent block supports a fine granularity of programming.

Clock Inversion

The selected clock may be optionally inverted. This limits the maximum frequency of operation due to the existence of one half cycle timing paths. Simultaneous bus writes and internal writes (for example writing a new count value while a counter is counting) are not supported when the internal clock is inverted and the same frequency as bus clock. This limitation affects A0, A1, D0, D1, and the Control register in counter mode.

Clock Enable Selection

The clock enable signal may be routed to any synchronous signal and can be selected from any of the four inputs from the routing matrix that are available to this block.

Clock Enable Inversion

The clock enable signal may be optionally inverted. This feature allows the clock enable to be generated in any polarity.

Clock Enable Mode

By default, the clock enable is OFF. After configuring the target block operation, software can set the mode to one of the following using the CFGxEN MODE[1:0] register shown in Figure 16-39.

Table 16-22. Clock Enable Mode

Clock Enable Mode	Description
OFF	Clock is OFF.
ON	Clock is ON. The selected global clock is free running.
Positive Edge	A gated clock is generated on each positive edge detect of the clock enable input. Maximum frequency of enable input is the selected global clock divided by two.
Level	Clocks are generated while the clock enable input is high ('1').

Clock Enable Usage

The two general usage scenarios for the clock enable are:

Firmware Enable – It is assumed that most functions require a firmware clock enable to start and stop the function. Because the boundary of a function mapped into the UDB array is arbitrary–it may span multiple UDBs and/or portions of UDBs–there must be a way to enable a given function atomically. This is typically implemented from a bit in a control register routed to one or more clock enable inputs. This scenario also supports the case where applications require multiple, unrelated blocks to be enabled simultaneously.

Emulated Local Clock Generation – This feature allows local clocks to be generated by UDBs, and distributed to other UDBs in the array by using a synchronous clock enable implementation scheme, rather than directly clocking from one UDB to another. Using the positive edge feature of the clock enable mode eliminates restrictions on the duty cycle of the clock enable waveform.

Special FIFO Clocking

The datapath FIFOs have special clocking considerations. By default, the FIFO clocks follow the same configuration as the datapath clock. However, the FIFOs have special control bits that alter the clock configuration:

- Each FIFO clock can be inverted with respect to the selected datapath clock polarity.
- When FIFO FAST mode is set, the bus clock overrides the datapath clock selection normally in use by the FIFO.

16.2.4.2 Reset Control

The two modes of reset control are: compatible mode and alternate mode. The modes are controlled by the ALT RES bit in each UDB configuration register CFG31. When this bit is '0', the compatible scheme is implemented. When this bit is '1', the alternate scheme is implemented.

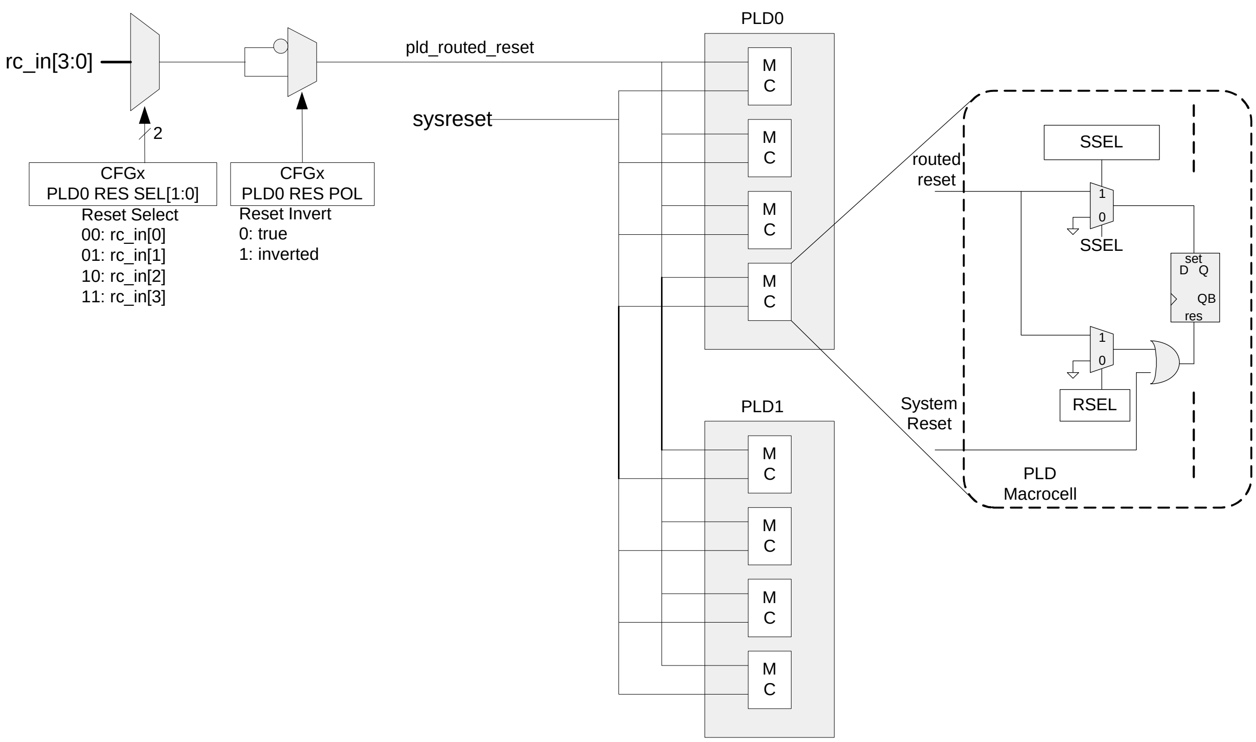
Compatible Reset Scheme

This scheme features a routed reset, for dynamically resetting the embedded state of block, which can be applied to each PLD macrocell and the SC counter.

Compatible PLD Reset Control

Figure 16-41 shows the compatible PLD reset system, using routed dynamic resets.

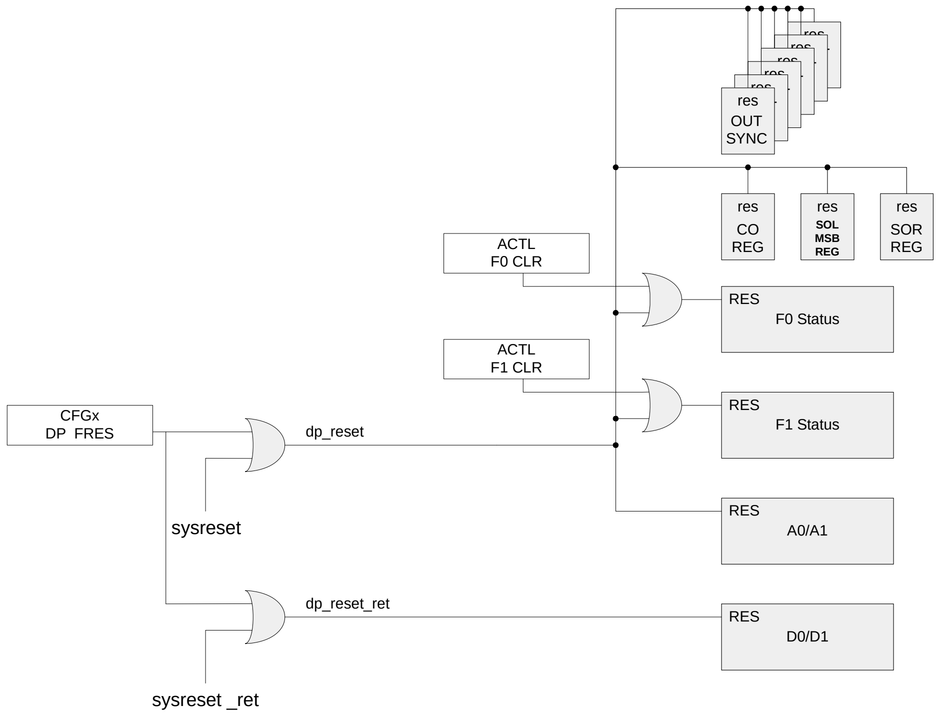
Figure 16-41. Compatible PLD Reset Structure



Compatible Datapath Reset Control

Figure 16-42 shows the compatible datapath reset system, using firmware reset. The firmware reset asynchronously clears the DP output registers, the carry and shift out flags, the FIFO state, accumulators, and data registers. Note that the DO and D1 registers are implemented as retention registers that maintain their state across sleep intervals. The FIFO data is unknown because it is RAM-based.

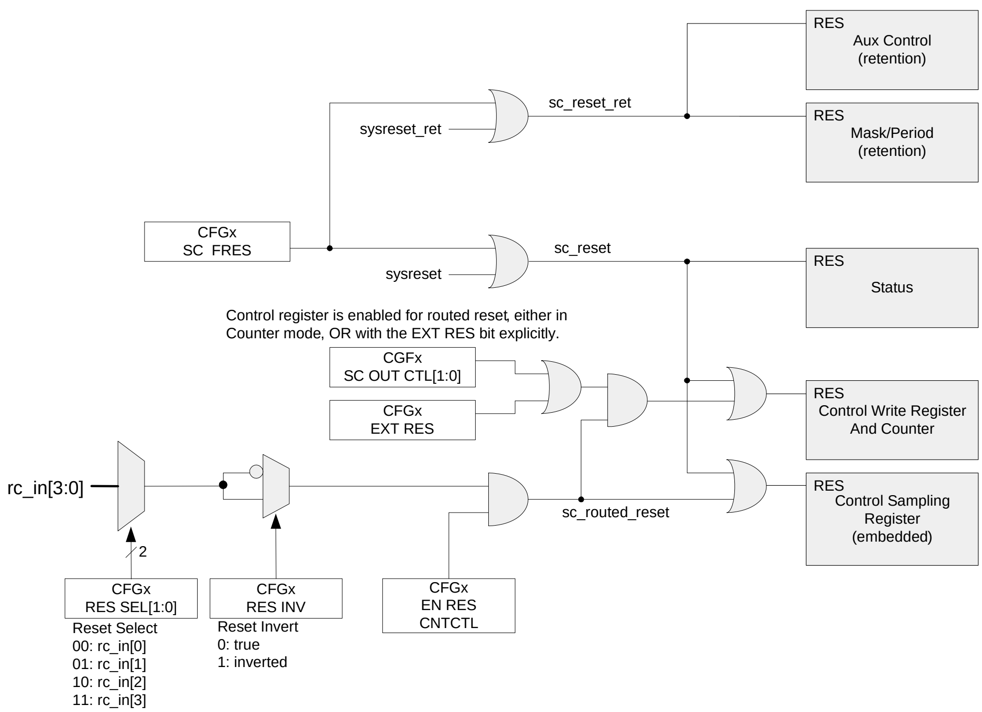
Figure 16-42. Compatible Datapath Reset Structure



Compatible Status and Control Reset Control

Figure 16-43 shows the compatible status and control block reset. The mask/period and auxiliary control registers are retention registers.

Figure 16-43. Compatible Status and Control Reset Control



The two modes of reset control are: compatible mode and alternate mode. The modes are controlled by the ALT RES bit in each UDB configuration register CFG31. When this bit is ‘0’, the compatible scheme is implemented. When this bit is ‘1’, the alternate scheme is implemented.

Alternate Reset Scheme

Table 16-23 shows a summary of the differences between the compatible reset scheme and the alternate reset scheme.

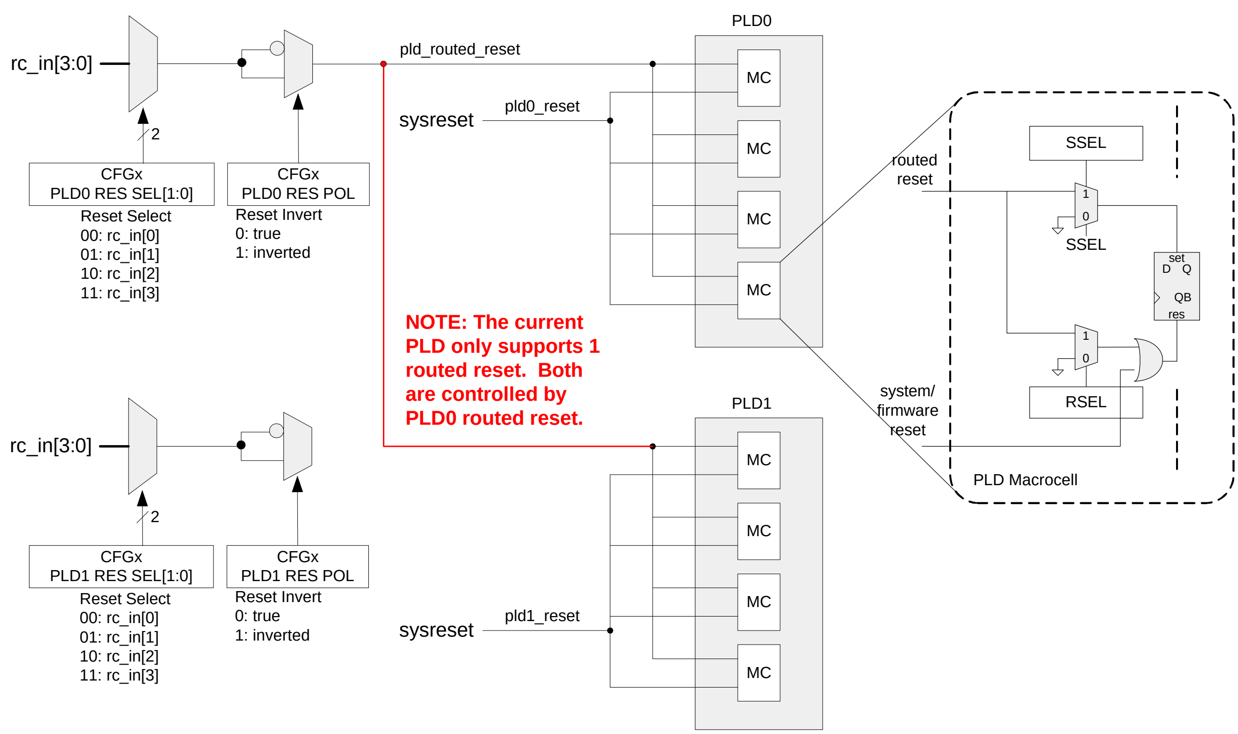
Table 16-23. Reset Schemes

Feature	Compatible	Alternate
Granularity	One routed reset is shared by all blocks in the UDB	Each UDB component block can select an individual reset
Status register	No routed reset capability	Optionally can use the selected SC routed reset
Datapath	No routed reset capability	Optionally can use the selected DP routed reset

Alternate PLD Reset Control

Figure 16-44 shows the alternate PLD reset system. Although there are provisions for individual resets for each PLD, this is not supported in the PLD block. Therefore, in the alternate reset scheme, the PLD0 reset control settings applies to both PLDs.

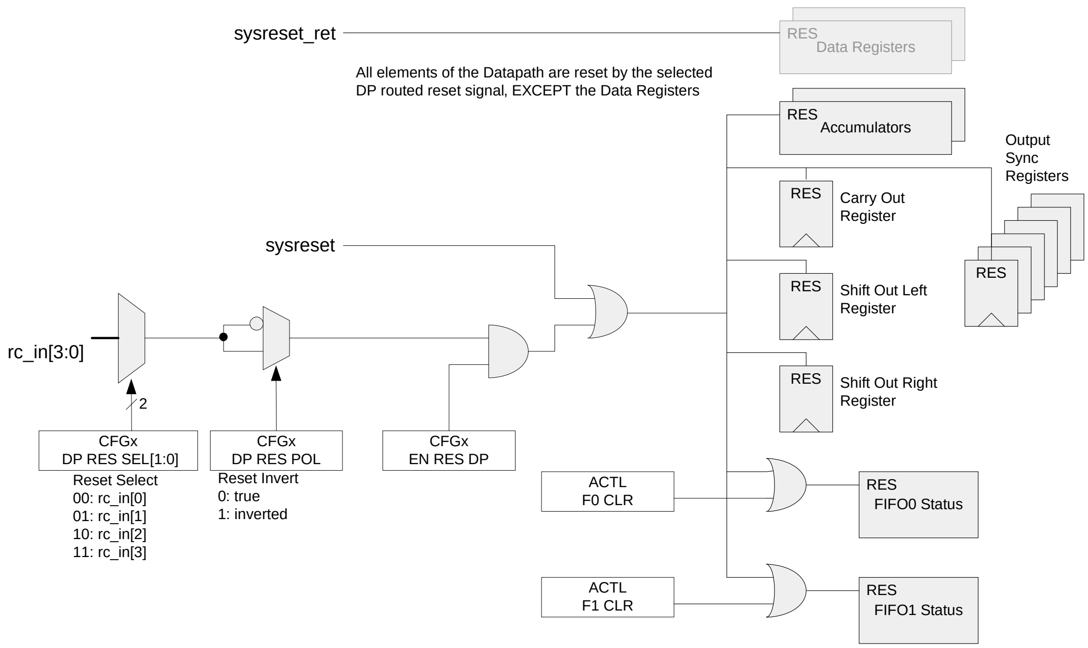
Figure 16-44. Alternate PLD Reset Structure



Alternate Datapath Reset Control

Figure 16-45 shows the alternate datapath reset system. The datapath routed reset applies to all datapath states, except the Data Registers, which are implemented as retention registers.

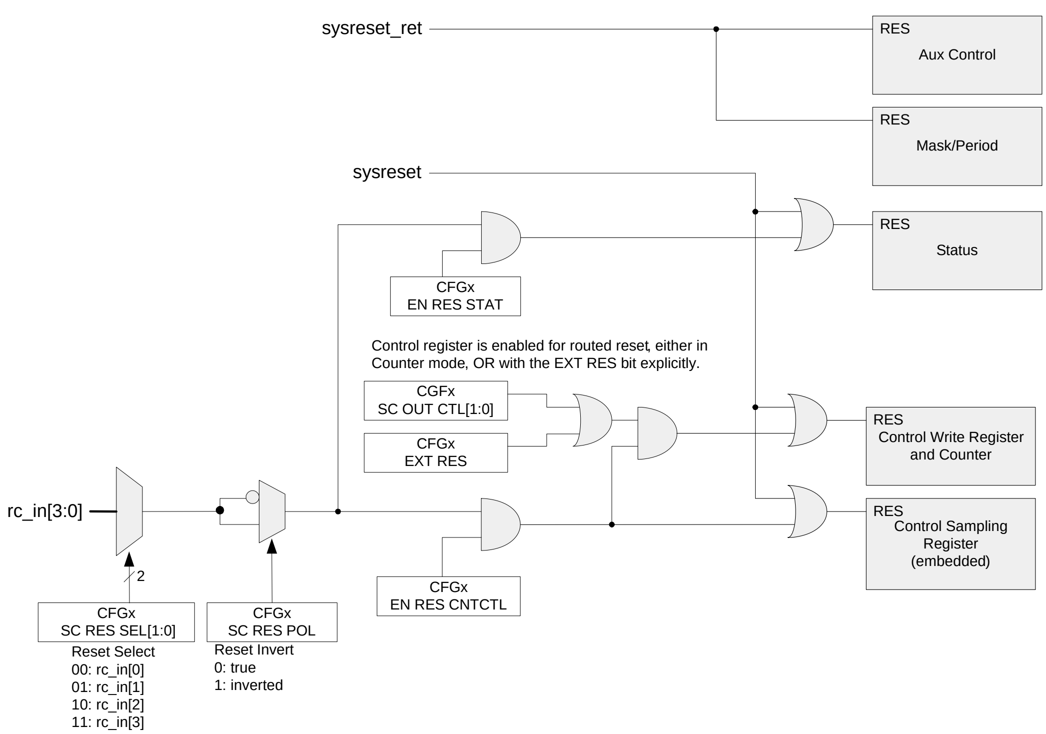
Figure 16-45. Alternate Datapath Reset Structure



Alternate Status and Control Reset Control

Figure 16-46 shows the alternate status and control block reset. The mask/period and auxiliary control registers are retention registers.

Figure 16-46. Alternate Status and Control Reset Control



16.2.4.3 UDB POR Initialization

Register and State Initialization

Table 16-24. UDB POR State Initialization

State Element	State Element	POR State
Configuration Latches	CFG 0 – 31	0
Ax, Dx, CTL, ACTL, MASK	Accumulators, data registers, auxiliary control register, mask register	0
ST, Macrocell	Status and macrocell read only registers	0
DP CFG RAM and Fx (FIFOs)	Datapath configuration RAM and FIFO RAM	Unknown
PLD RAM	PLD configuration RAM	Unknown

Routing Initialization

On POR, the state of input and output routing is as follows:

- All outputs from the UDB that drive into the routing matrix are held at '0'.
- All drivers out of the routing and into UDB inputs are initially gated to '0'.

As a result of this initialization, conflicting drive states on the routing are avoided and initial configuration occurs in an order-independent sequence.

16.2.5 UDB Addressing

The UDBs can be accessed through a number of address spaces, for 8, 16, and 32-bit accesses of both the working registers (A0, A1, D0, D1, FIFOs, and so on) and the configuration registers.

- 8-bit working registers – This address space allows access to individual working registers in a single UDB.
- 16-bit working registers consecutive – This address space allows access to the same working register in two consecutive UDBs, for example D0 of UDB n and D0 of UDB n + 1
- 16-bit working registers paired – This address space allows access to two working registers, for example A0 and A1, from the same UDB.
- 32-bit working registers – This address space allows access to the same working register, for example A1, in all four UDBs.
- 8, 16 or 32-bit configuration registers – This address space allows access to the configuration registers for a single UDB.

16.2.6 System Bus Access Coherency

UDB registers have dual access modes:

- System bus access, where the CPU is reading or writing a UDB register.
- UDB internal access, where the UDB function is updating or using the contents of a register.

16.2.6.1 Simultaneous System Bus Access

Table 16-25 lists the possible simultaneous access events and required behavior:

Table 16-25. Simultaneous System Bus Access

<table>
<tr><th>Register</th><th>UDB Write
Bus Write</th><th>UDB Write
Bus Read</th><th>UDB Read
Bus Write</th><th>UDB Read
Bus Read</th></tr>
<tr><td>Ax</td><td rowspan="2">Undefined result</td><td rowspan="2">Not allowed directly[a, b]</td><td rowspan="2">UDB reads previous value</td><td rowspan="2">Current value is read by both</td></tr>
<tr><td>Dx</td></tr>
<tr><td>Fx</td><td>Not supported (UDB and bus must be opposite access)</td><td colspan="2">If FIFO status flags are used, no simultaneous read/ write at the same location is possible</td><td>Not supported (UDB and bus must be opposite access)</td></tr>
<tr><td>ST</td><td>NA, bus does not write</td><td>Bus reads previous value</td><td colspan="2">NA, UDB does not read</td></tr>
<tr><td>CTL</td><td colspan="2">NA, UDB does not write</td><td rowspan="5">UDB reads previous value</td><td rowspan="6">Current value is read by both</td></tr>
<tr><td>CNT</td><td>Undefined result</td><td>Not allowed directly[c]</td></tr>
<tr><td>ACTL</td><td colspan="2" rowspan="3">NA, UDB does not write</td></tr>
<tr><td>MASK</td></tr>
<tr><td>PER</td></tr>
<tr><td>Macrocell (RO)</td><td>NA, bus does not write</td><td>Not allowed directly[d]</td><td>NA, bus does not write</td></tr>
</table>

a. The Ax registers can be safely read by using software capture feature of the FIFOs.
b. The Dx registers can only be written to dynamically by the FIFOs. When this mode is programmed, direct read of the Dx registers is not allowed.
c. The CNT register can only be safely read when it is disabled. An alternative for dynamically reading the CNT value is to route the output to the SC register (in transparent mode).
d. Macrocell register bits can also be routed to the status register (in transparent mode) inputs for safe reading.

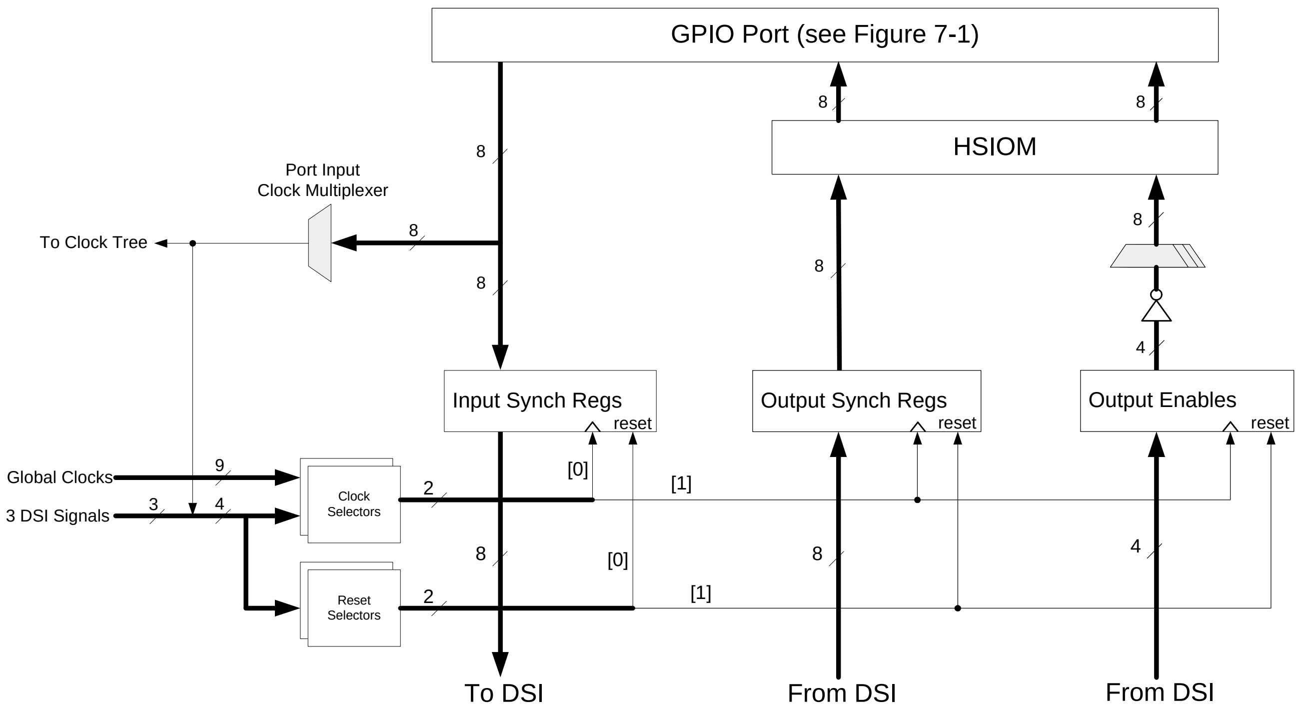
16.2.6.2 *Coherent Accumulator Access (Atomic Reads and Writes)*

The UDB accumulators are the primary target of data computation. Therefore, reading these registers directly during normal operation gives an undefined result, as indicated in Table 16-25. However, there is built-in support for atomic reads in the form of software capture, which is implemented across chained blocks. In this usage model, a read of the least significant accumulator transfers the data from all chained blocks to their associated FIFOs. Atomic writes to the accumulator can be implemented programmatically. Individual writes can be performed to the input FIFOs, and then the status signal of the last FIFO written can be routed to all associated blocks and simultaneously transfer the FIFO data into the Dx or Ax registers.

16.3 Port Adapter Block

The Port Adaptor block extends the UDBs to provide an interface to the GPIOs through the High-Speed I/O Matrix (HSIOM), described in High-Speed I/O Matrix on page 55. The HSIOM places registers for faster routing of DSI signals to GPIO outputs and output enables. The HSIOM also allows GPIOs to be shared amongst multiple blocks, for example port data registers and peripherals such as I2C. Figure 16-47 shows a high-level view.

Figure 16-47. Port Adapter Block Diagram



Each 8-bit GPIO port has one port adaptor (PA). There are eight inputs from the GPIO data in, eight outputs to the GPIO data out, and eight output enable (OE) connections. The registers in the PA are used for synchronizing inputs, outputs, and output enables.

Another feature is the port input clock multiplexer. This multiplexer selects one of the port inputs to be used as a clock. The clock can be used locally in the PA and routed to the global clocks (see Clocking System chapter on page 61).

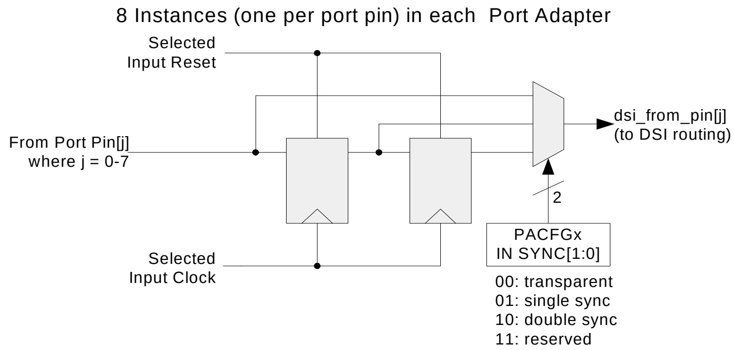
Two programmable clock selectors are available, to supply separate clocks for the input and output synchronization registers. The OE register uses the same clock as the output register.

Also, two programmable reset selectors are available, in the same manner as for the clock selectors.

16.3.1 PA Data Input Logic

Figure 16-48 shows the structure for the data input logic. Inputs are from each pin of an I/O port. The signal can be either single synchronized or double synchronized, or synchronization can be bypassed for asynchronous inputs. Synchronization is to the selected port input clock. The output of this circuit connects to the DSI routing.

Figure 16-48. Detail of GPIO Input Logic



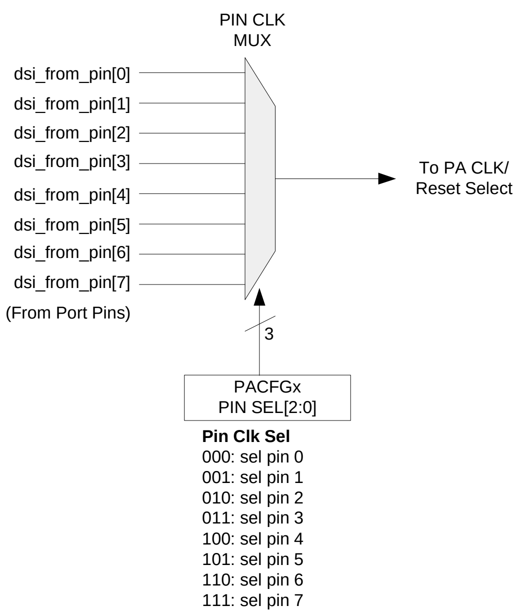
16.3.2 PA Port Pin Clock Multiplexer Logic

Figure 16-49 shows the Port Pin multiplexer. Each port has eight data input signals, one of which is selected for use as a clock. This selection is routed for use as:

- Programmable clock in the port adapter
- Source for the UDB clock tree
- Programmable reset in the port adapter
- For use as a clock enable in the port adapter.

Note that the selected signal does not pass through synchronizers and is asynchronous to other clock domains within the block. It should be used carefully for selected functions.

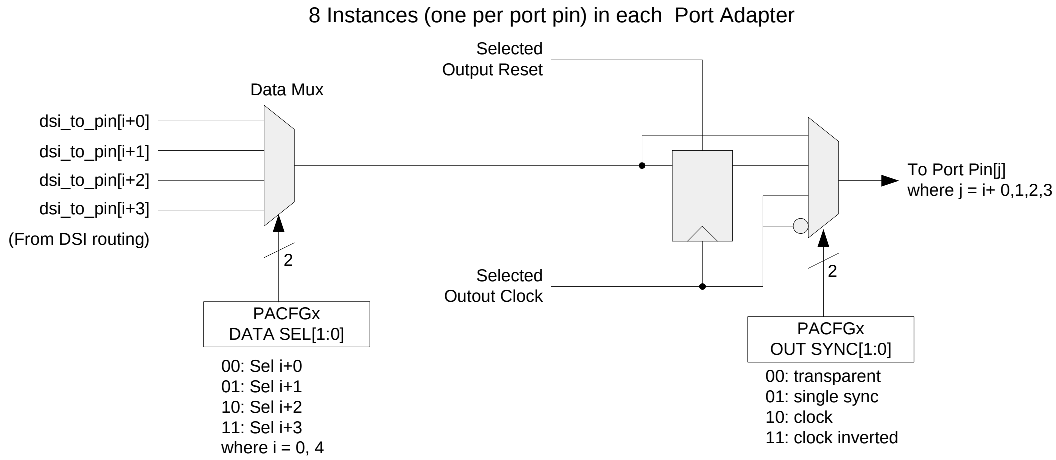
Figure 16-49. Detail of GPIO Pin Selection



16.3.3 PA Data Output Logic

Figure 16-50 shows the structure for the data output logic. Outputs go to each pin of an I/O port (through HSIOM). The signal can be single synchronized or synchronization can be bypassed for asynchronous outputs. Other options include the ability to output either the selected clock or an inverted version of the clock.

Figure 16-50. Detail of GPIO Output Data Logic



16.3.4 PA Output Enable Logic

Figure 16-51 shows the output enable (OE) logic. This circuit shares the clock and reset associated with data output. This connection is unique in that there are four DSI outputs associated with the OE, but these are muxed to a total of four OE connections to the I/O port pins, as Figure 16-52 shows.

Figure 16-51. GPIO Output Enable (OE) Sync Logic

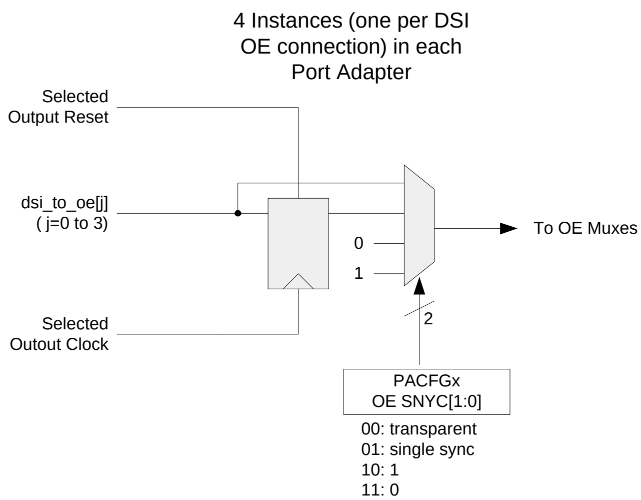
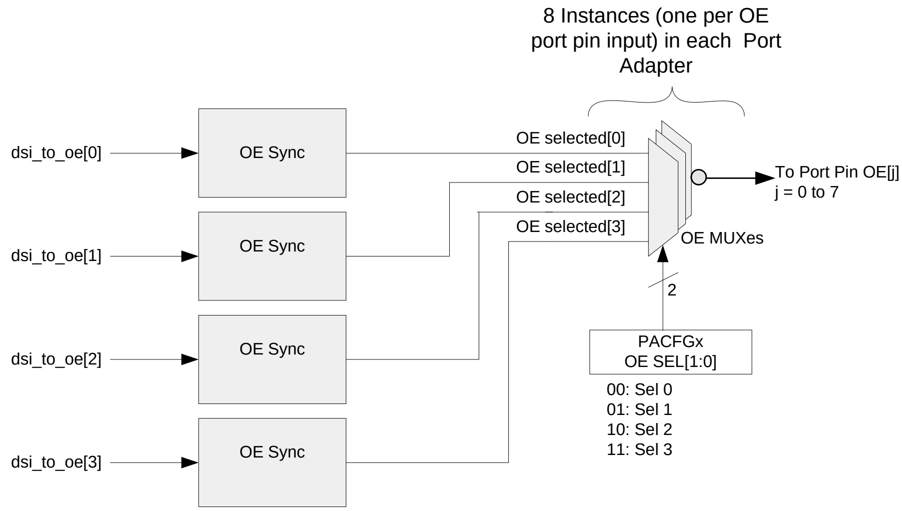


Figure 16-52. Output Enable (OE) Multiplexers

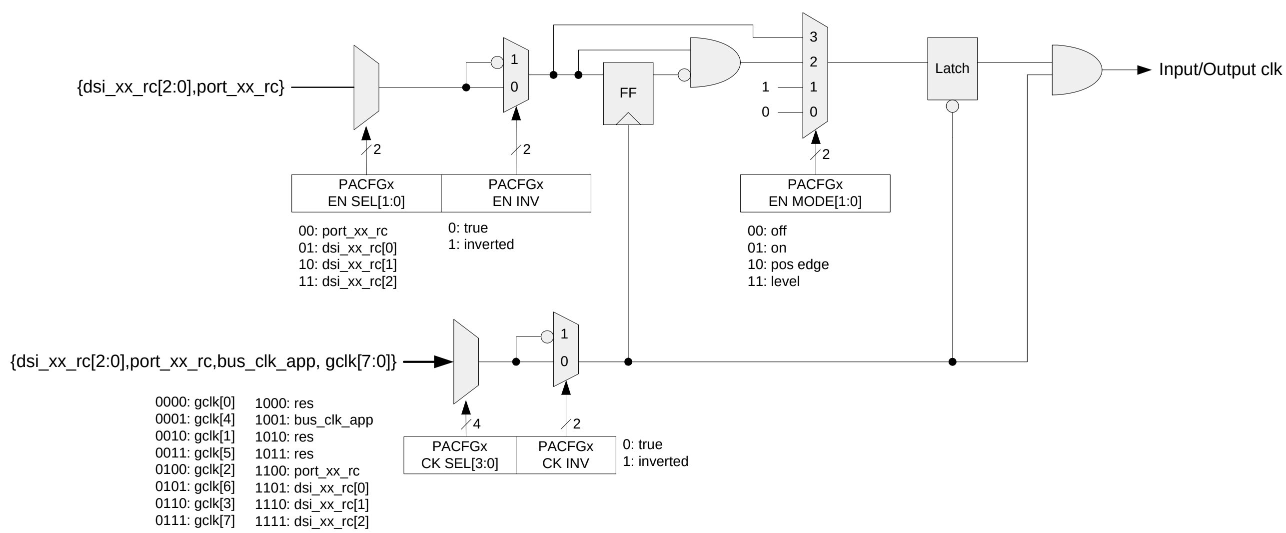


Note that due to the active low sense of the OE signals at the ports, there is an additional inversion in the path between the OE sync logic and the OE multiplexers.

16.3.5 PA Clock Multiplexer

Figure 16-53 shows the structure of the PA Clock Multiplexer. As noted previously, each PA has two programmable clock selectors, to supply separate clocks for port inputs and outputs and output enables (OEs).

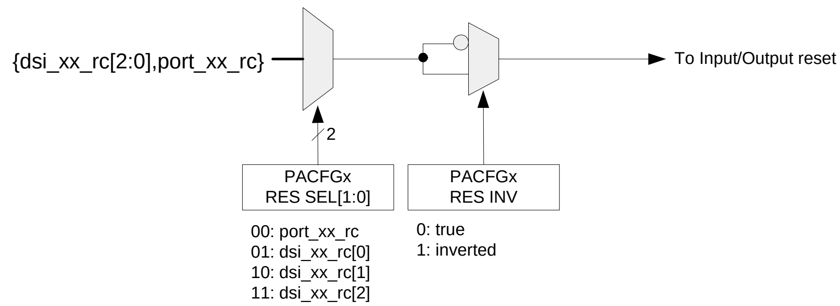
Figure 16-53. PA Clock Multiplexer Detail



16.3.6 PA Reset Multiplexer

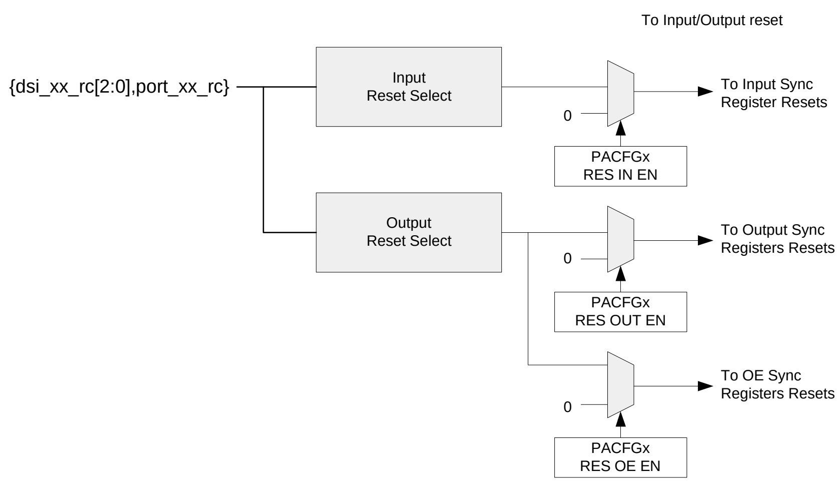
The structure of the PA Reset Multiplexer is shown in Figure 16-54.

Figure 16-54. PA Reset Multiplexer Detail



As shown in Figure 16-55, the reset selection logic is duplicated, one for input, and one that serves both output and output enable. Each of these resets has an individual enable, which applies to all the 8 bits in the associated category.

Figure 16-55. PA Reset System



17. Timer, Counter, and PWM

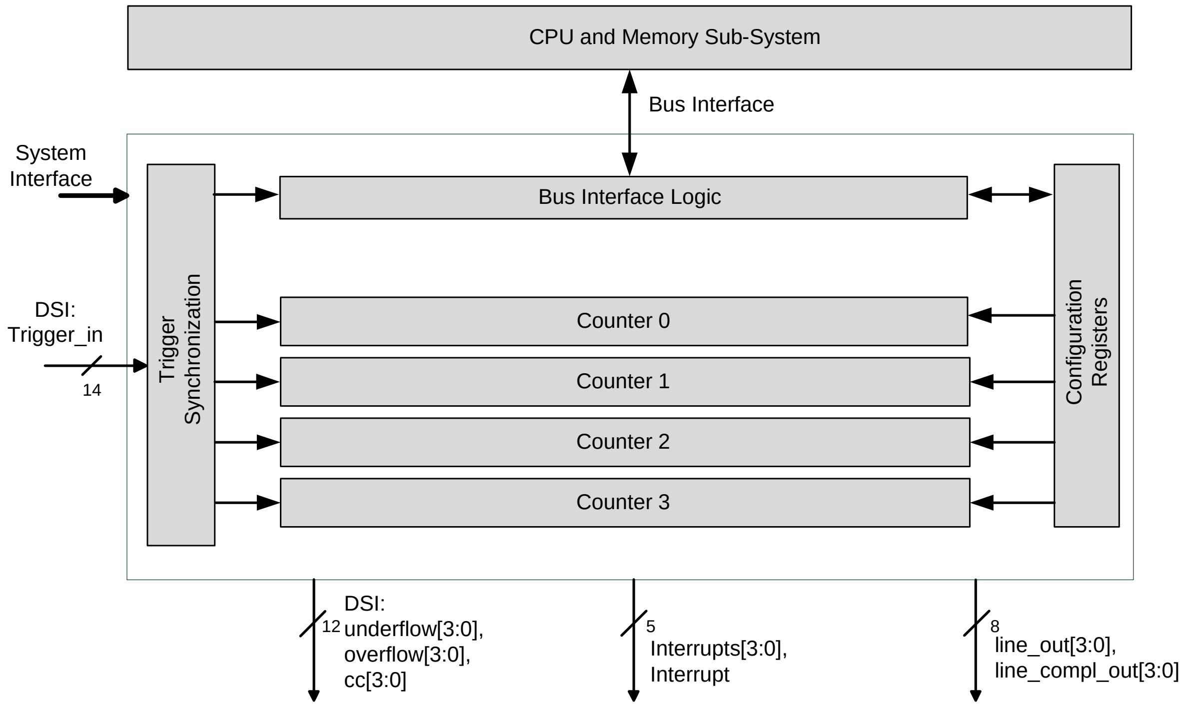
The Timer, Counter, and Pulse-Width Modulator (TCPWM) block in PSoC® 4 implements a 16-bit timer, counter, and pulse-width modulation functionality using four dedicated counters and corresponding logic circuits. The block can be used to measure period and pulse width of the input signal, for quadrature decoding, to find the time of occurrence of interrupt, and as a PWM generation unit. This chapter explains the operational modes of the TCPWM block.

17.1 Features

- Four 16-bit timers, counters, and pulse-width modulator (PWM)
- The TCPWM block supports the following operational modes for each counter independently:
  - Timer and counter
  - Capture
  - Quadrature decoding
  - Pulse-width modulation
  - Pseudo random PWM
  - PWM with dead time
- Multiple counting modes – Up, down, and up/down
- Clock pre-scaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count – The final value in the counter register is reached
  - Capture/Compare – The count is captured to the capture/compare register or the counter value equals the compare value
- Synchronized counters – The counters can reload, start, stop, and count at the same time
- DSI output signals for each counter to indicate underflow, overflow, and capture/compare condition
- Complementary line output for PWMs
- Selectable start, reload, stop, count, and capture event signals for each counter from up to 14 DSI signals with rising edge, falling edge, both edges, and level trigger options

17.2 Block Diagram

Figure 17-1. TCPWM Block Diagram



The block has these interfaces:

- Bus interface: Connects block to CPU and Memory sub-system.
- I/O signal interface with DSI: Used to route signals to or from the Universal Digital Block (UDB) and TCPWM block. It consists of input triggers (such as reload, start, stop, count, and capture) and output signals (such as overflow (OV), underflow (UV), and capture/compare (CC)). Any GPIO can be used as the input trigger signal.
- Interrupts: Provides interrupt request signals from each counter based on terminal count (TC) or CC conditions, and a combined interrupt signal generated by the logical OR of all four interrupt request signals.
- System interface: Consists of control signals such as clock and reset related signals from the system to the block.

The TCPWM block consists of four counters, which can be configured independently by writing to the registers. See TCPWM Registers on page 186 for more information on all registers required for this block.

17.2.1 Enabling and Disabling Counters in TCPWM Block

The counters can be enabled by setting the COUNTER_ENABLED[3:0] field of control register TCPWM_CTRL, as shown in Table 17-1. The bit position from the LSB corresponds to the counter number. The counter must be configured before enabling it. Enabling the counter after configuring it, updates the registers with the new configured value. Disabling the counter retains the values in the registers until it is enabled again.

For example, to update the count value of counter x with a new value:

1. Disable counter x
2. Update the count register (TCPWM_CNTx_COUNTER) of counter x
3. Enable counter x

Table 17-1. Bit-Field Settings to Enable/Disable Counter x

COUNTER_ENABLED[x]	Description
0	Disable counter x
1	Enable counter x

Note x can be 0, 1, 2, and 3

17.2.2 Clocking

The TCPWM receives the system clock through the system interface to synchronize all events in the block. Counter enable signals (counter_en[3:0]), generated when the counter is enabled, gate the system clock to provide counter specific clocks counter_clock[3:0] for each counter.

Clock Pre-Scaling: counter_clock can be pre-scaled, with 1, 2, 4… 64, 128 dividers. This can be done independently for all counters by modifying the GENERIC field of counter control (TCPWM_CNTx_CTRL) register, as shown in Table 17-2.

Table 17-2. Bit-Field Setting to Pre-Scale 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 pre-scaling cannot be done in pulse-width modulation with dead time (PWM-DT) and quadrature mode.

17.2.3 Events Based on Trigger Inputs

These are the events that can be triggered by hardware and software. Hardware triggers can be level signal, rising edge, falling edge, or both edges.

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

Figure 17-2. TCPWM Trigger Selection and Event Detection

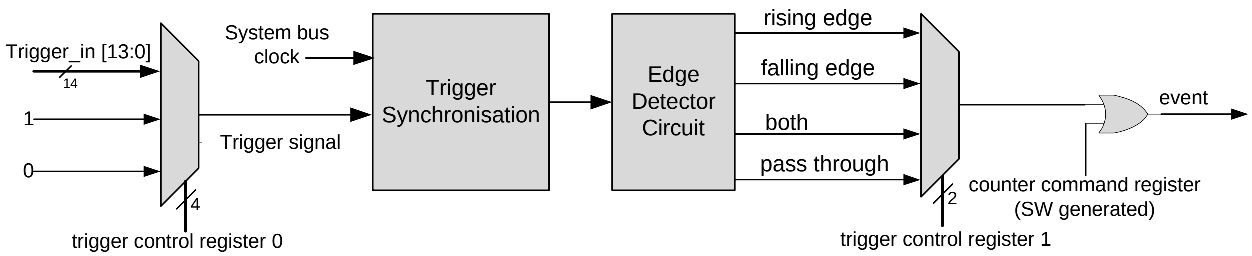


Figure 17-2 explains the flow of event detection in the TCPWM block. Trigger control register 0 (TCPWM_CNTx_TR_CTRL0) selects one of the 14 trigger inputs as the event signal. A constant '0' or '1' signal can also be used as event signal.

Any edge (rising, falling, or both) or level (high or low) can be selected for the occurrence of an event by modifying trigger control register 1 (TCPWM_CNTx_TR_CTRL1). This configuration is for each event and is present in each counter. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 17-2.

The events derived from these triggers can have different definitions in different modes.

- **Reload:** A reload event initializes and starts the counter.
  - In up counting mode, the count register (TCPWM_CNTx_COUNTER) is initialized with ‘0’.
  - In down counting mode, the counter is initialized with the period value stored in TCPWM_CNTx_PERIOD register.
  - In up/down counting mode, the count register is initialized with ‘0’.
  - 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 and can be used after a stop event or after re-initialization of the counter register to any value by software. The count register is not initialized with any value 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 176.
- **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 the PWM output lines.
- **Capture:** A capture event copies the counter register value to the capture register and capture register value to the buffer capture register. In the PWM modes, the capture event acts as a switch event. It switches the values of the capture/compare and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

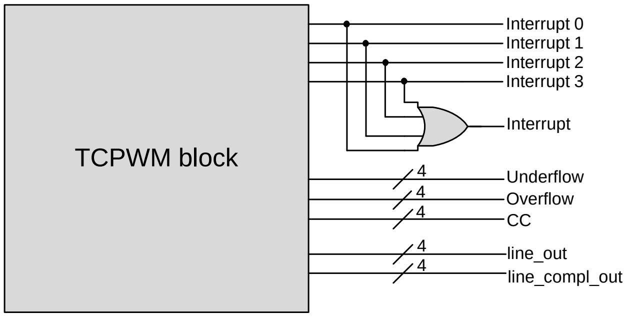
Notes

- All trigger inputs are synchronized to the system clock.
- 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 (in the counter frequency domain) and a timer configuration in which a pre-scaled (divided) counter clock is used.

17.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 17-3.

Figure 17-3. TCPWM Output Signals



17.2.4.1 Signals upon Trigger Conditions

- A counter generates an internal overflow (OV) condition when counting up and the count register reaches the period value.
- A counter generates an internal underflow (UN) condition when counting down and the count register reaches zero.
- The capture/compare (CC) condition is generated by the TCPWM when the counter is running.
  - The counter value equals the compare value - When this event occurs, the device behavior depends on the mode in which it is configured. This is explained in the respective sections.
  - A capture event occurs - When a capture event occurs, the TCPWM_CNTx_COUNTER register value is copied to the capture register and capture register value is copied to the buffer capture register.

17.2.4.2 Interrupts

The block provides a dedicated interrupt output signal for each of the four counters. An interrupt can be generated for a TC condition and a CC condition. The exact definition of these conditions is mode-specific. All four interrupt output signals are also OR’ed together to produce a single interrupt output signal.

Four registers are used for interrupt handling in this block, as shown in Table 17-3.

Table 17-3. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNTx _INTR	0	TC	Terminal count. Is set to '1', when a terminal count is detected. Write with '1' to clear bit.
	1	CC_MATCH	Counter matches Capture/Compare register. It is set to '1' when a CC condition is detected. Write with '1' to clear bit.
TCPWM_CNTX _INTR_SET	0	TC	Write with '1' to set corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
	1	CC_MATCH	Write with '1' to set corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
TCPWM_CNTX _INTR_MASK	0	TC	Mask bit for corresponding TC bit in the interrupt request register.
	1	CC_MATCH	Mask bit for corresponding CC_MATCH bit in the interrupt request register.
TCPWM_CNTx _INTR_MASKED	0	TC	Logical AND of corresponding TC request and mask bits.
	1	CC_MATCH	Logical AND of corresponding CC_MATCH request and mask bits.

17.2.4.3 Outputs

Each counter is provided with two outputs, line_out[3:0] and line_compl_out[3:0] (complementary of line_out). Note that the OV, UN, and CC conditions can be used to drive line_out and line_compl_out as needed, by configuring the TCPWM_CNTx_TR_CTRL2 register (see Table 17-4).

Table 17-4. 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	

Table 17-4. Configuring Output Line for OV, UN, and CC Conditions

Field	Bit	Value	Event	Description
OVERFLOW_MODE Default Value = 3	3:2	0	Set line_out to '1	Configures output line on a over-flow (OV) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	
UNDERFLOW_MODE Default Value = 3	5:4	0	Set line_out to '1	Configures output line on a under-flow (UN) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	

17.2.5 Power Modes

The TCPWM block works in Active and Sleep modes. The block's power is connected to V_{CCD}. The configuration registers and other logic are powered in deep-sleep mode to keep the states for registers, which need to retain their values. See Table 17-5.

Table 17-5. 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.
Hibernate	In this mode, the power to this block is switched off. Configuration registers will lose their state.

17.3 Modes of Operation

The counter block can function in six operational modes, depending on register configuration, as shown in Table 17-6. MODE [26:24] field of counter control register (TCPWM_CNTx_CTRL) register configures the counter in specific operational mode.

Table 17-6. Operational Mode Configuration

Mode	MODE Field [26:24]	Description
Timer	000	Increments or decrements by ‘1’ every counter clock cycle in which a count event is detected.
Capture	010	Increments or decrements by ‘1’ every counter clock cycle in which a count event is detected. A capture event copies the counter value into the capture register.
Quadrature Decoder	011	Quadrature decoding. Counter is decremented or incremented, based on two phase inputs according to X1, X2, or X4 encoding scheme.
PWM	100	Pulse-width modulation - Modulates the duty cycle and period.
PWM-DT	101	Pulse-width modulation with dead time. Disables the PWM outputs for some configured time.
PWM-PR	110	Pseudo random pulse-width modulation. Uses 16-bit LFSR to generate pseudo random values.

The block can be set to count using up, down, and up/down counting modes by setting UP_DOWN_MODE[17:16] field in TCPWM_CNTx_CTRL register, as shown in Table 17-7.

Table 17-7. Counting Mode Configuration

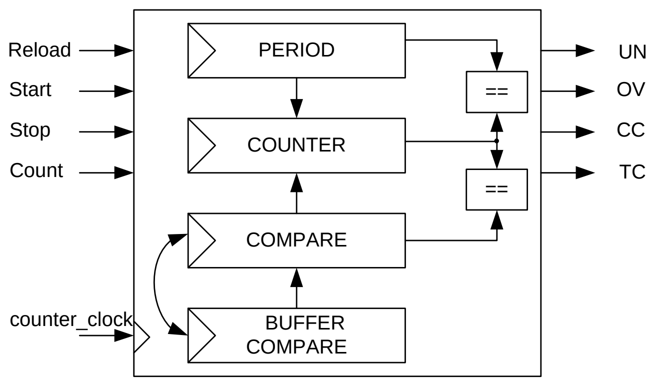
Counting Modes	UP_DOWN_MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the period value is reached.
DOWN Counting Mode	01	Decrements the counter until the value 0 is reached. A TC condition is generated when the value '0' is reached.
UP/DOWN Counting Mode 0	10	Increments the counter until the period value is reached, then decrements the counter until '0' is reached. A TC condition is generated when '0' is reached but not when the period value 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.

17.3.1 Timer Mode

The timer mode is used to time the occurrence of events or to measure time difference between events using interrupt requests.

17.3.1.1 Block Diagram

Figure 17-4. Timer Mode Block Diagram



17.3.1.2 How it Works

The timer can be configured to count in up, down, and up/down counting modes. It can be configured to run in either continuous mode or one-shot mode.

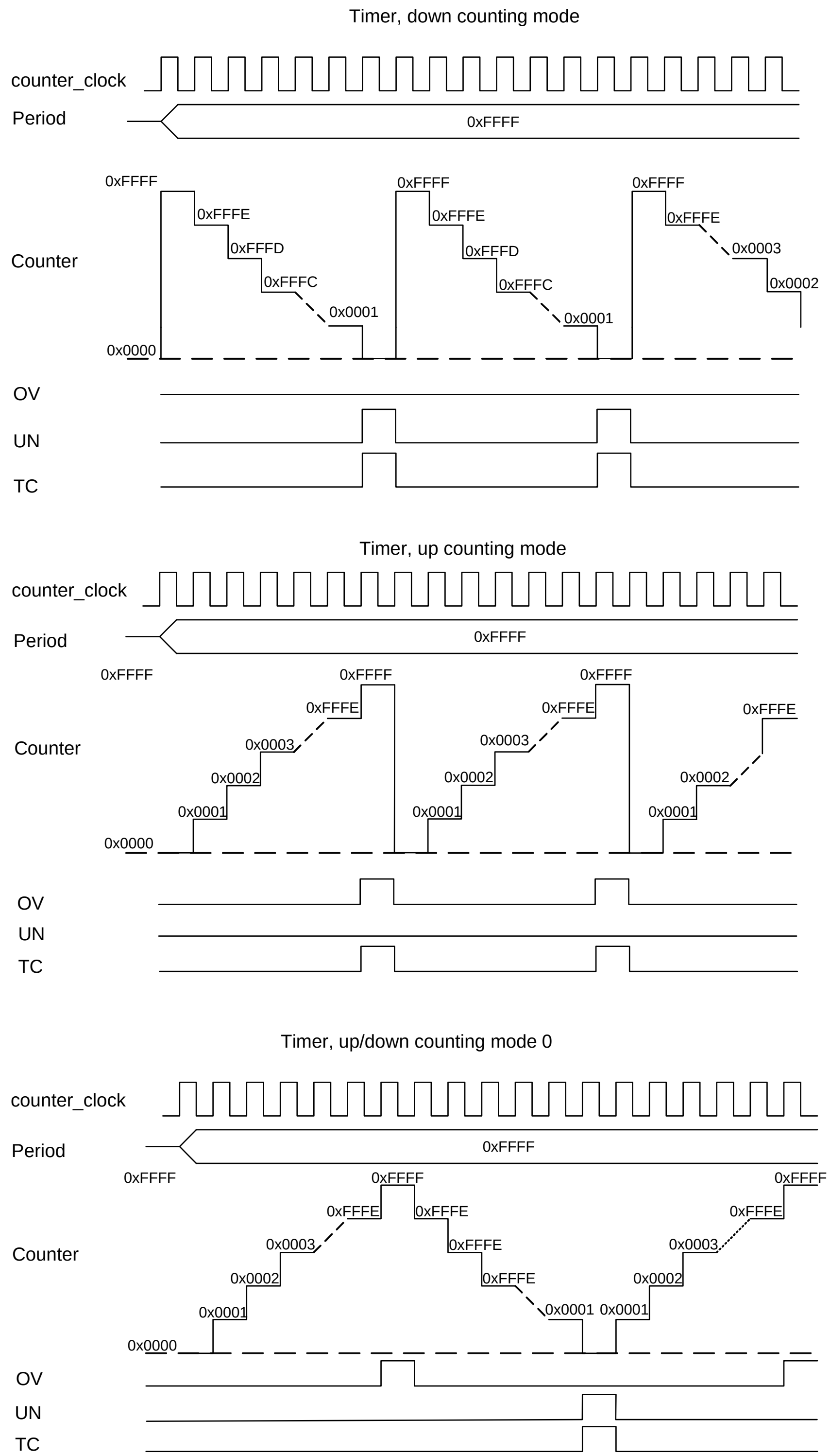
The following explains the working of the timer:

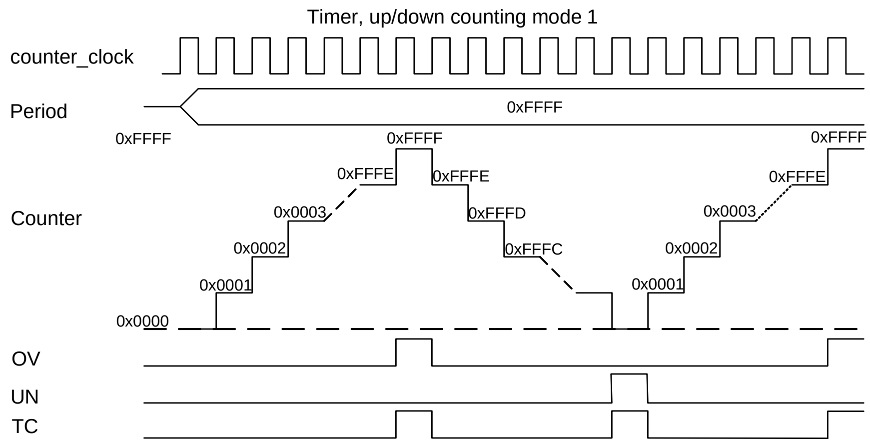
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER). **Note** Do not write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with the value in the period register.
  - In the up/down counting modes, the count register value is not updated upon reaching the terminal values. Instead the direction of counting changes when the count value reaches 0 or the period value.
- The CC condition is generated when 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 bitfield of the counter control (TCPWM_CNTx_CTRL) register. This condition can be used to generate an interrupt request.

Figure 17-5 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 (both modes 0 and 1 both), a period value of A results in 2*A counter cycles (0 to A and back to 0).

Figure 17-5. Timing Diagram for Timer in Multiple Counting Modes





17.3.1.3 Configuring Counter for Timer Mode

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

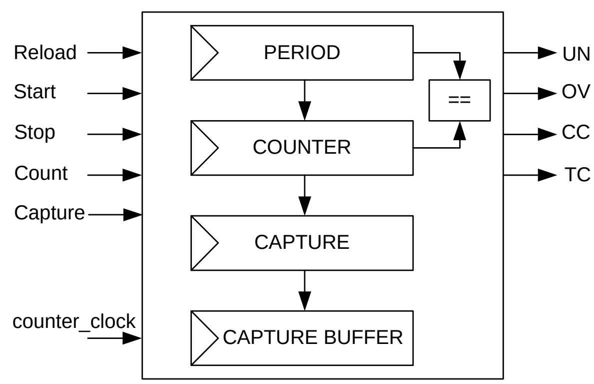
1. Disable the counter by writing '0' to COUNTER_ENABLED[x] field of TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register. Set AUTO_RELOAD_CC field of counter control register, if required to switch values at every CC condition.
5. Set clock pre-scaling by writing to the GENERIC[10:8] field of the counter control (TCPWM_CNTx_CTRL) register, as shown in Table 17-2.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-7.
7. 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 the TCPWM_CNTx_CTRL register.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Reload, Start, Stop, Capture, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge of the trigger, which causes the event (Reload, Start, Stop, Capture, and Count).
10. If required, set the interrupt upon TC or CC condition, as shown in Interrupts on page 170.
11. Enable the Counter by writing '1' to the COUNTER_ENABLED[x] field of TCPWM_CTRL register.

17.3.2 Capture Mode

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

17.3.2.1 Block Diagram

Figure 17-6. Capture Mode Block Diagram



17.3.2.2 How it Works

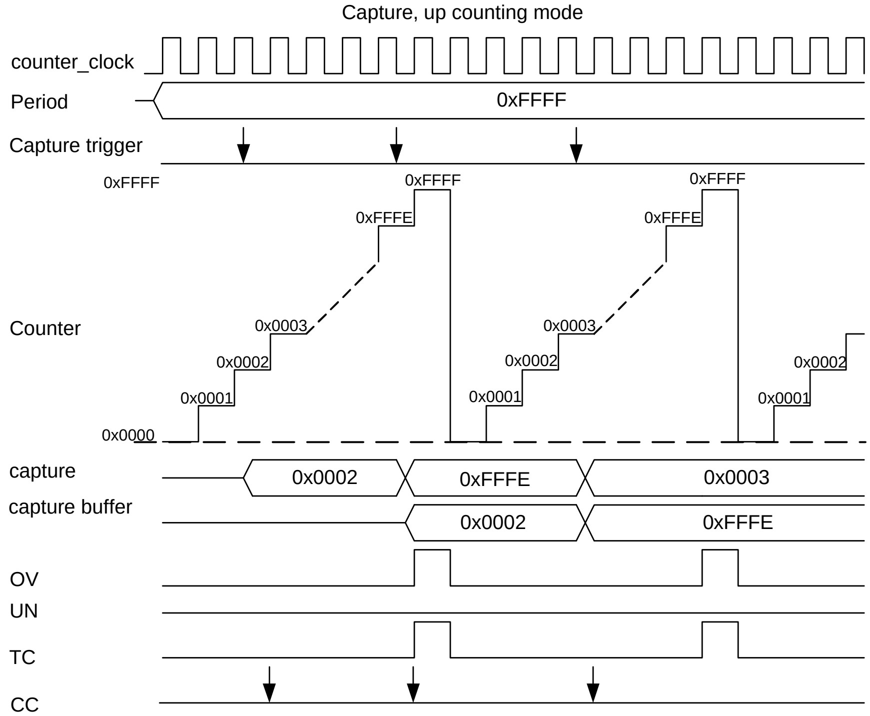
The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNTx_CTRL).

Operation in capture mode occurs as follows:

- During a capture event, generated either by hardware (HW) or software (SW), the current count register value is copied to the capture register (TCPWM_CNTx_CC) and the capture register value is copied to the buffer capture register (TCPWM_CNTx_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 17-7 illustrates the capture behavior in the up counting mode.

Figure 17-7. Timing Diagram of Counter in Capture Mode, Up Counting Mode



In the figure, observe that:

- The period register contains the maximum count register value.
- Internal overflow (OV) and TC conditions are generated when the counter reaches the period value.
- A capture event is only possible at the edges or through software. Trigger control register 1 should be configured for capture event on edges only.
- 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 counter_clock frequency.

17.3.2.3 Configuring Counter for Capture Mode

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

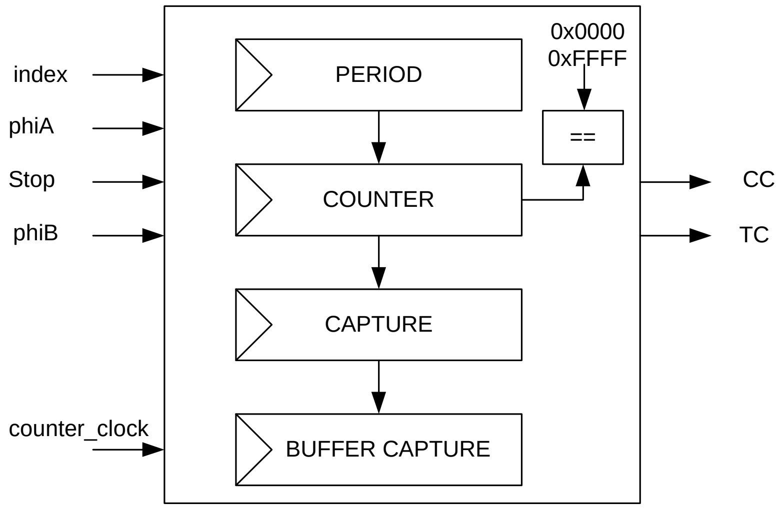
1. Disable the counter by writing '0' to COUNTER_ENABLED[x] field of TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set clock pre-scaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-2.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of TCPWM_CNTx_CTRL register, as shown in Table 17-7.
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 TCPWM_CNTx_CTRL register.
7. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Reload, Start, Stop, Capture, and Count).
8. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge, which 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 COUNTER_ENABLED[x] field of TCPWM_CTRL register.

17.3.3 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 inputs phiA and phiB to the decoder.

17.3.3.1 Block Diagram

Figure 17-8. Quadrature Mode Block Diagram



17.3.3.2 How it Works

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

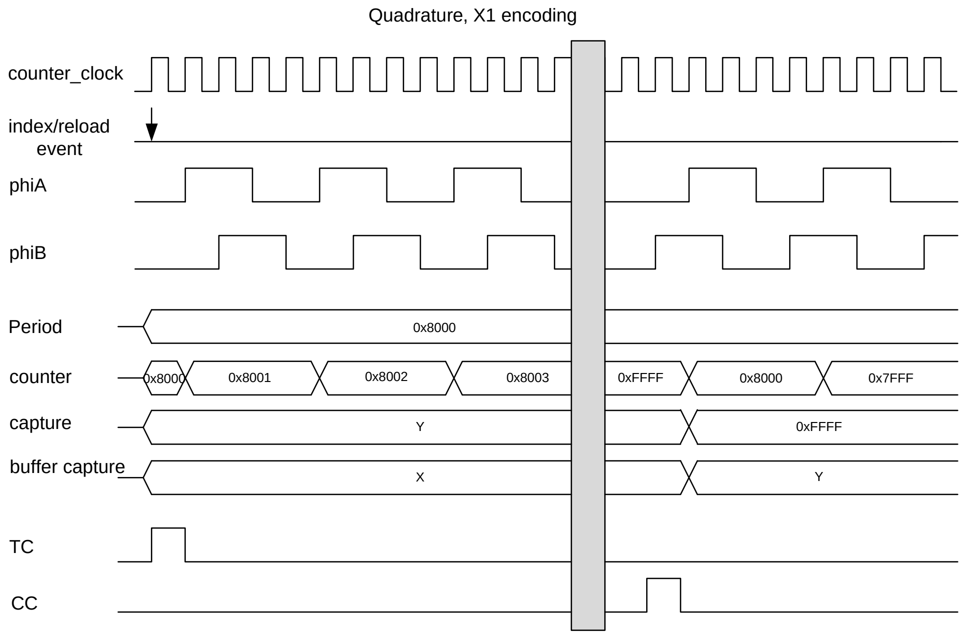
The Quadrature mode operation occurs as follows:

- Quadrature phases phiA and phiB: Counting direction is determined by the phase relationship between phiA and phiB. These phases are selected by the count and start trigger inputs, respectively as hardware input signals to the decoder.
- Quadrature index event: This is selected by the reload signal as a hardware input signal. This event generates a TC condition, as shown in Figure 17-9.

  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 can be used to generate an interrupt request
- The value in the capture register can be used to determine which condition caused the event and whether:
  - A counter underflow occurred (value 0)
  - A counter overflow occurred (value 0xFFFF)
  - An index/TC event occurred (value is not equal to either 0 or 0xFFFF)
- The DOWN bit field of counter status (TCPWM_CNTx_STATUS) register can be read to determine the current counting direction. Value '0' indicates a previous increment operation and value '1' indicates previous decrement operation. Figure 17-9 illustrates quadrature behavior in the X1 encoding mode.
  - A positive edge on phiA/count increments the counter when phiB/start is '0' and decrements the counter when phiB/start is '1'.
  - The count register is initialized with the period value on an index/reload event.
  - Terminal count is generated when the counter is initialized by index event. This event can be used to generate an interrupt.
  - When the count register reaches 0xFFFF (the maximum count register value), the count register value is copied to the capture register and the count register is initialized with period value (0x8000).

Figure 17-9. Timing Diagram for Quadrature Mode, X1 Encoding

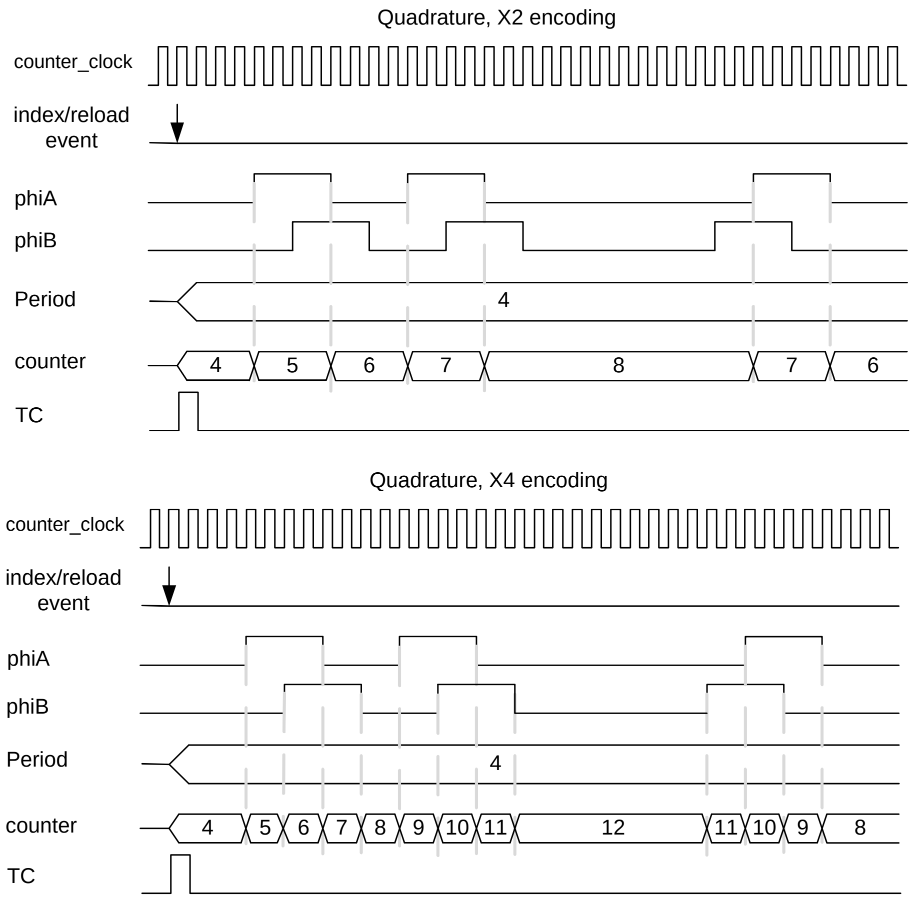


Because the quadrature phases are detected on counter_clock, they should not change value more than once, within a single counter_clock period.

The X2 and X4 quadrature encoding modes count twice and four times as fast as the X1 quadrature encoding mode.

Figure 17-10 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

Figure 17-10. Timing Diagram for Quadrature Mode, X2 and X4 Encoding



17.3.3.3 *Configuring Counter for Quadrature Mode*

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

1. Disable the counter by writing '0' to COUNTER_ENABLED[x] field of TCPWM_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of TCPWM_CNTx_CTRL register.
5. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Index and Stop).
6. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge, which 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 COUNTER_ENABLED[x] field of TCPWM_CTRL register.

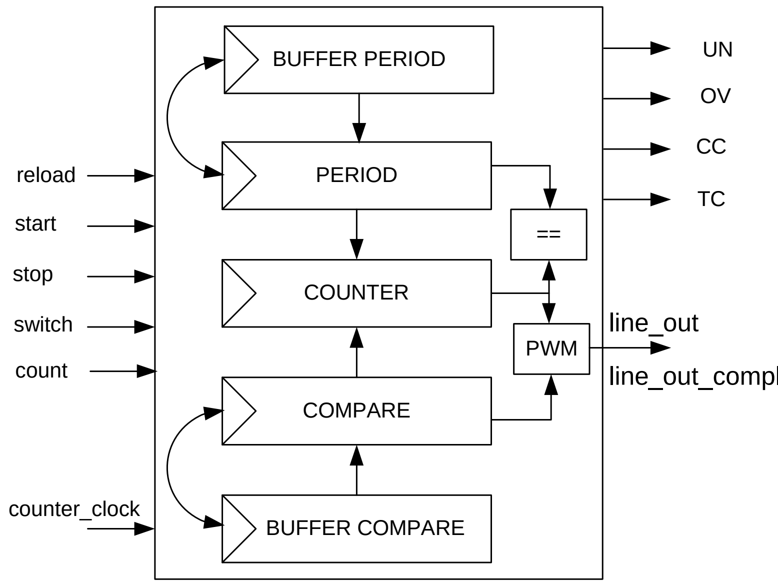
17.3.4 Pulse-Width Modulation Mode

PWM mode is also called the Comparator mode, because the comparison output is a PWM output with a varying duty cycle and a varying period. The period depends on the Period register. The duty cycle depends on the compare value and period value.

PWM Period = (Period Value × 1/Clock frequency)

17.3.4.1 Block Diagram

Figure 17-11. PWM Mode Block Diagram



17.3.4.2 How it Works

This mode can be used in up, down, and up/down counting modes by setting UP_DOWN_MODE [17:16] bits in TCPWM_CNTx_CTRL register, as shown in Table 17-7. These counting modes are used for left-aligned, right-aligned, and center-aligned pulse-width modulation.

At the reload event, the count register is initialized and starts counting in the appropriate mode. At every count, the count register value is compared with compare register value. This is used to toggle PWM output line or set it to '0' or '1'.

PWM output line is controlled by OV, UN, and CC conditions. The conditions can toggle the output line or set it to '0' or '1' by configuring the TCPWM_CNTx_TR_CTRL2 register.

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. AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of counter control register are set to value ‘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.
- Updating buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in Figure 17-13.

The output line is set to '0' at Terminal Count and toggled at the CC condition.

Figure 17-12 illustrates center aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 17-12. Timing Diagram for Center Aligned PWM

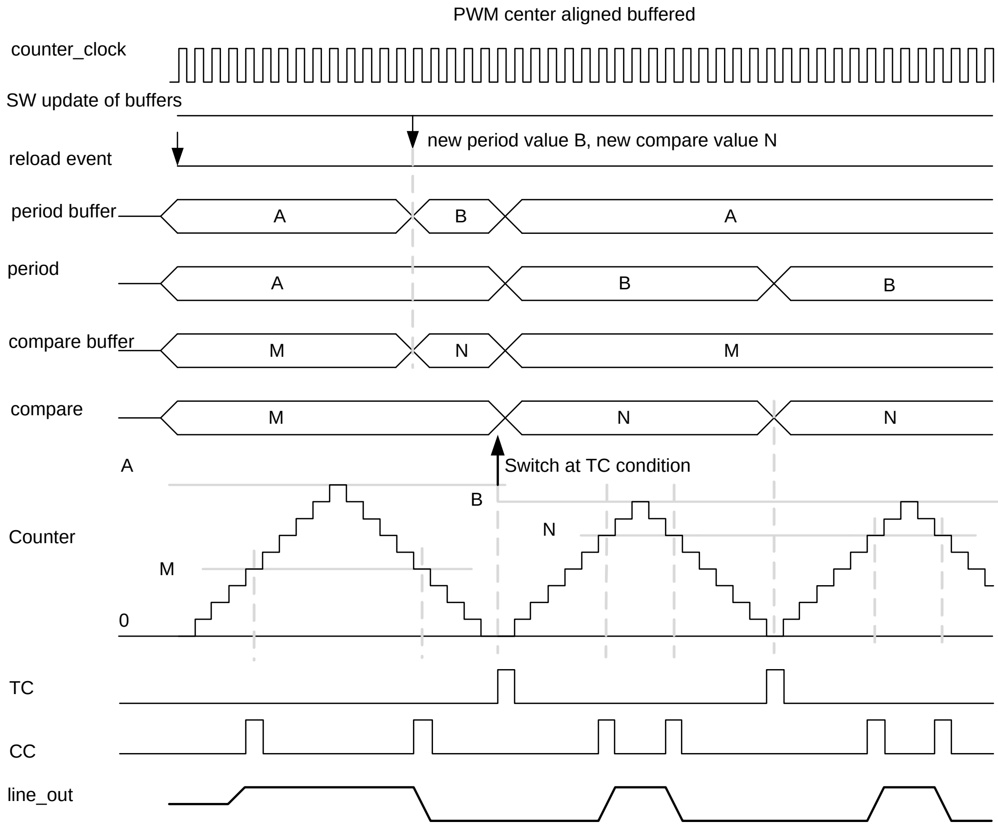
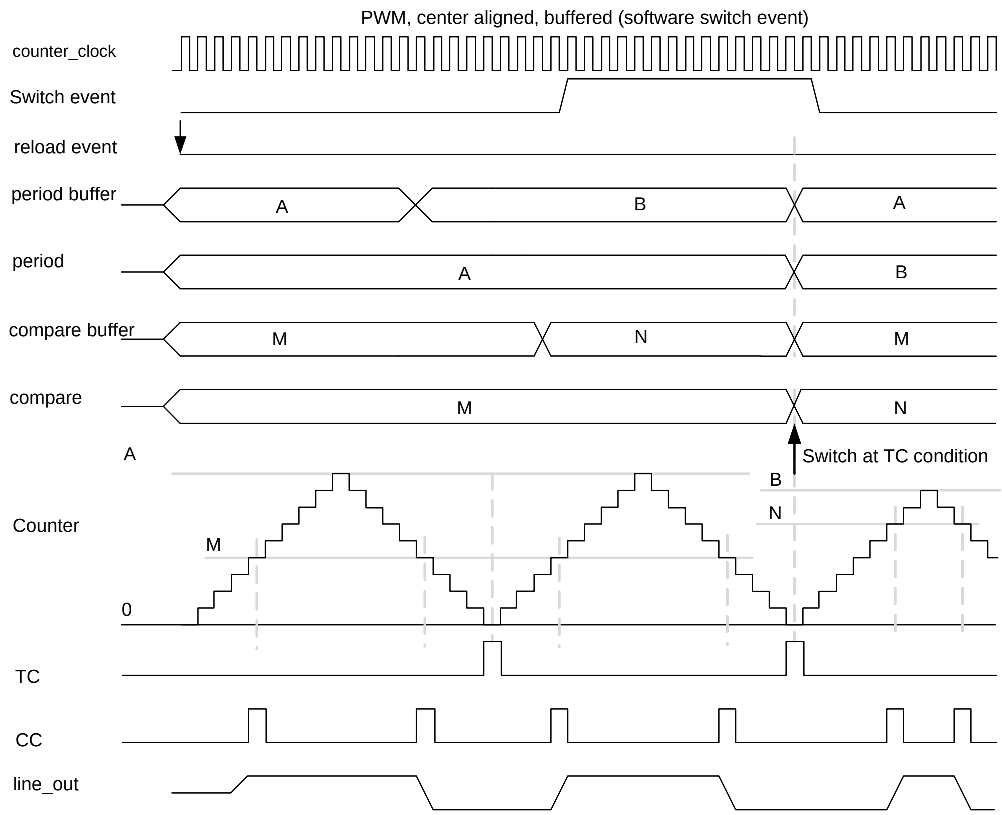


Figure 17-13 illustrates center-aligned PWM with software generated switch events:

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

Figure 17-13. Timing Diagram for Center Aligned PWM (software switch event)



17.3.4.3 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 17-4.
- 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 17-4.

17.3.4.4 Kill Feature

Kill feature implies 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 field of counter control register, as shown in Table 17-8.

Table 17-8. Field Setting for Stop on Kill Feature

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

A kill event can be programmed to be asynchronous or synchronous, as shown in Table 17-9.

Table 17-9. 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 synchronous kill, PWM cannot be started before the next TC. To restart the PWM immediately after kill input is removed, kill event should be asynchronous (see

Table 17-9). 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.

17.3.4.5 Configuring Counter for PWM Mode

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

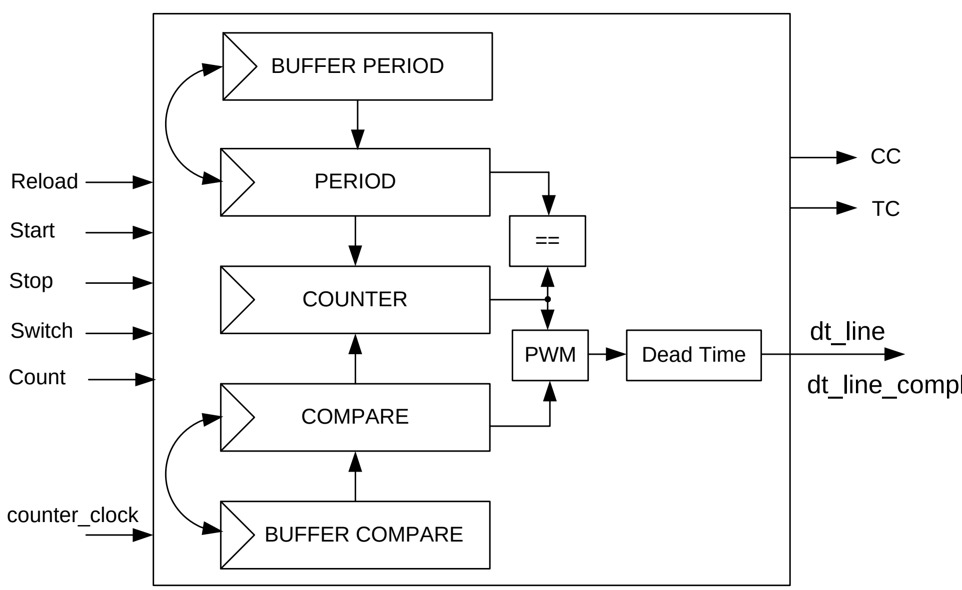
1. Disable the counter by writing '0' to COUNTER_ENABLED[x] field of TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set clock pre-scaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-2.
4. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register and buffer period value in TCPWM_CNTx_PERIOD_BUFF register, if required to switch values.
5. Set the 16-bit compare value in TCPWM_CNTx_CC register and buffer compare value in TCPWM_CNTx_CC_BUFF register, if required to switch values.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of TCPWM_CNTx_CTRL register to configure left-align, right align or center aligned PWM, as shown in Table 17-7.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL field of TCPWM_CNTx_CTRL register as required.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge, which causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line _out_compl can be controlled by TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, 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 COUNTER_ENABLED[x] field of TCPWM_CTRL register

17.3.5 Pulse-Width Modulation with Dead Time Mode

Dead time is used to delay the transitions of both "line_out" and "line_out_compl". It separates the transition edges of these two signals by a 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 low for a fixed period. The dead band feature allows generation of two PWM pulses with non-overlapping outputs. Dead time of maximum 255 clocks can be generated.

17.3.5.1 Block Diagram

Figure 17-14. PWM-DT Mode Block Diagram



17.3.5.2 How it Works

Operation in PWM with Dead Time mode occurs as follows:

- On the rising edge of PWM line_out depending upon UN, OV, and CC conditions, the dead time block sets the dt_line to '0' for the dead band period.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period has completed, dt_line is set to '1'.
- On the falling edge of PWM line_out depending upon UN, OV, and CC conditions, the dead time block sets the dt_line_compl to '0' for the dead band period.
- 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 same as line_out.
- When the duration of the dead time equals or exceeds the width of a pulse, the pulse is removed.

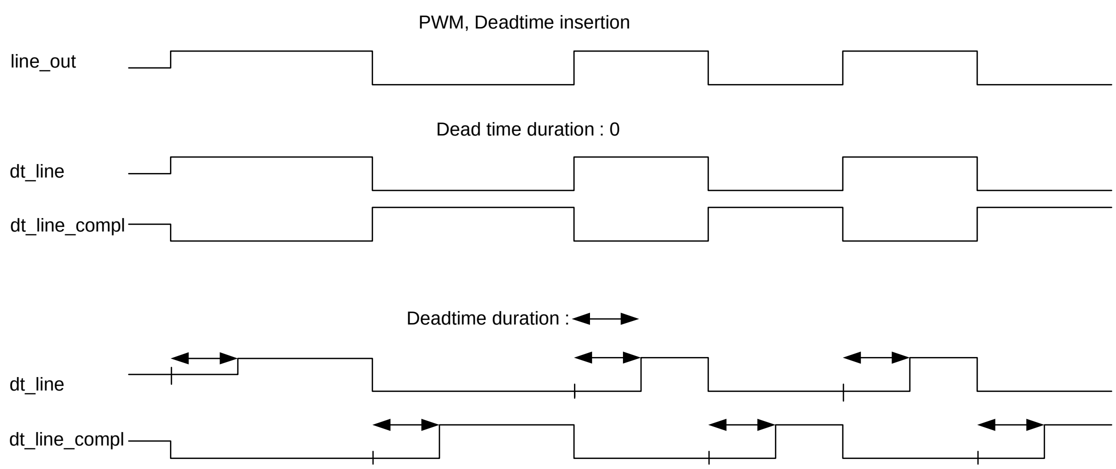
This mode follows PWM mode and supports the following features available with that mode:

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

This mode does not support clock pre-scaling.

Figure 17-15 illustrates how the complementary output lines "dt_line" and "dt_line_compl" are generated from the PWM output line "line_out".

Figure 17-15. Timing Diagram for PWM, with and without Dead Time



17.3.5.3 Configuring Counter for PWM with Dead Time Mode

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

1. Disable the counter by writing '0' to COUNTER_ENABLED[x] field of TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required Dead-Time by writing to the GENERIC[15:8] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-2.
4. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register and buffer period value in TCPWM_CNTx_PERIOD_BUFF register, if required to switch values.
5. Set the 16-bit compare value in TCPWM_CNTx_CC register and buffer compare value in TCPWM_CNTx_CC_BUFF register, if required to switch values.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of TCPWM_CNTx_CTRL register to configure left-align, right align or center aligned PWM, as shown in Table 17-7.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL field of TCPWM_CNTx_CTRL register as required, as shown in the Pulse-Width Modulation Mode on page 179.

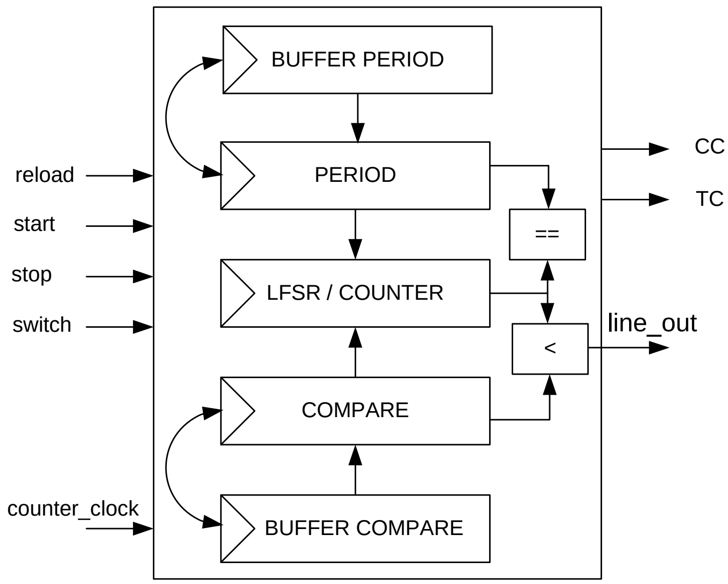
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge, which causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by TCPWM_CNTx_TR_CTRL2 register to set, reset or invert upon CC, OV, 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 COUNTER_ENABLED[x] field of TCPWM_CTRL register.

17.3.6 Pulse-Width Modulation Pseudo Random Mode

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

17.3.6.1 Block Diagram

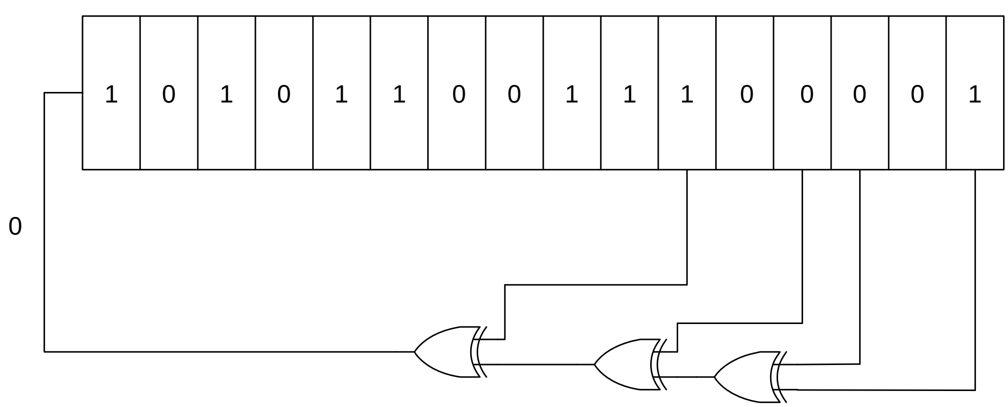
Figure 17-16. PWM-PR Mode Block Diagram



17.3.6.2 How it Works

The counter register is used to implement LFSR with the polynomial: x16+x14+x13+x11+1, as shown in Figure 17-17. It generates all the numbers in the range [1, 0xffff] in a pseudo-random sequence. Note that the counter register should be initialized with a value different from 0.

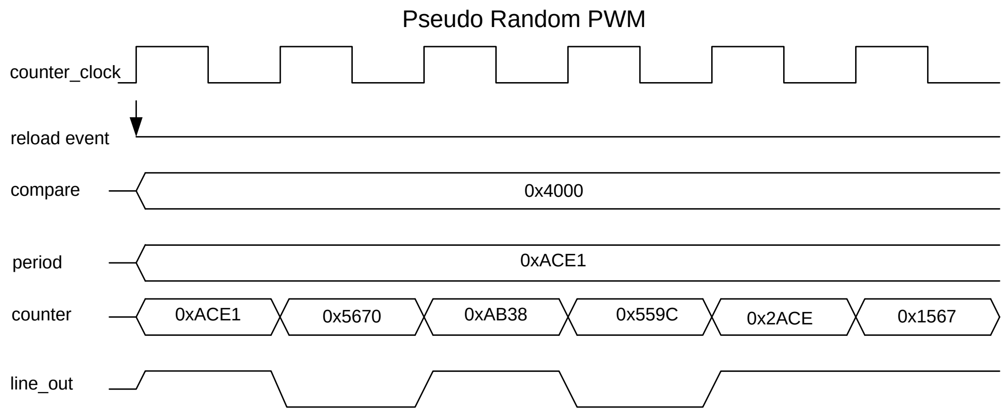
Figure 17-17. Pseudo Random Sequence Generation using Counter Register



The following steps describe the process:

- PWM output line "line_out" is driven with '1' when the lower 15 bits of the counter register are smaller than the value in the compare register (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; it starts the counter. However, it does not initialize the counter.
- Terminal count is generated when the counter equals the period value. LFSR generates a predictable pattern of counter values given a certain counter initialization. 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 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 AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD field of counter control register).
- A kill event can be programmed to stop the counter as shown earlier.
- One shot mode (configured by setting ONE_SHOT field of counter control register): At TC, the counter is stopped by hardware.
- In this mode, there is neither underflow and overflow internal event nor trigger condition.
- CC condition occurs when the counter is running and its value equals compare value. Figure 17-18 illustrates pseudo random noise behavior.
- With a compare value of 0x4000, resulting in 50 percent duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

Figure 17-18. Timing Diagram for Pseudo Random PWM



A capture/switch input signal may switch the values between the compare and compare buffer registers and the period and period buffer registers. This functionality can be used to modulate between two different compare values using a trigger input signal to control the modulation.

Note Capture/switch input signal can only be triggered by an edge (rising, falling, or both). This input signal is remembered until the next Terminal Count.

17.3.6.3 Configuring Counter for Pseudo Random PWM Mode

The steps to configure the counter 'x' 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[x] field of TCPWM_CTRL register.
2. Select Pseudo Random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNTx_PERIOD register and buffer period value in TCPWM_CNTx_PERIOD_BUFF register, if required to switch values.
4. Set the 16-bit compare value in TCPWM_CNTx_CC register and buffer compare value in TCPWM_CNTx_CC_BUFF register, to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL field of TCPWM_CNTx_CTRL register as required.
6. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger, which causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge, which causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, 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 COUNTER_ENABLED[x] field of TCPWM_CTRL register.

17.4 TCPWM Registers

Table 17-10. Block Registers

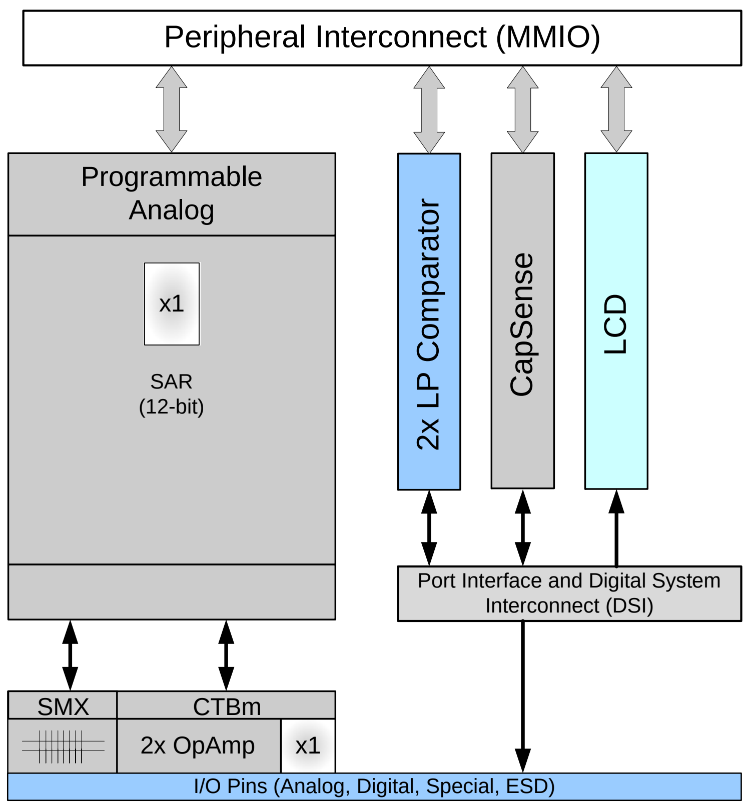
Register	Comment	Features
TCPWM_CTRL	TCPWM Control Register	Used to enable counter block
TCPWM_CMD	TCPWM Command Register	Used to generate software events
TCPWM_INTR_CAUSE	TCPWM Counter Interrupt Cause Register	To determine the source of combined interrupt signal
TCPWM_CNTx_CTRL	Counter control register	Configures mode of counter, encoding modes, one shot mode, switching, kill feature, dead time, clock pre-scaling, counting direction,
TCPWM_CNTx_STATUS	Counter status register	Read the direction of counting, dead time duration, clock pre-scaling and to check if counter is running
TCPWM_CNTx_COUNTER	Count register	Contains the 16-bit counter value
TCPWM_CNTx_CC	Counter compare/capture register	Captures the counter value or compare the value with counter value
TCPWM_CNTx_CC_BUFF	Counter buffered compare/capture register	Buffer register for counter CC register, used for switching compare value
TCPWM_CNTx_PERIOD	Counter period register	Contains upper value of the counter
TCPWM_CNTx_PERIOD_BUFF	Counter buffered period register	Buffer register for counter PERIOD register, used for switching compare value
TCPWM_CNTx_TR_CTRL0	Counter trigger control register 0	To select trigger for specific counter events
TCPWM_CNTx_TR_CTRL1	Counter trigger control register 1	To determine edge detection for specific counter input signals
TCPWM_CNTx_TR_CTRL2	Counter trigger control register 2	Used to control counter output lines upon CC, OV, UN conditions
TCPWM_CNTx_INTR	Interrupt request register.	Register bit is set when TC or CC condition is detected
TCPWM_CNTx_INTR_SET	Interrupt set request register.	Used to set corresponding bits in interrupt request register
TCPWM_CNTx_INTR_MASK	Interrupt mask register.	Mask for interrupt request register
TCPWM_CNTx_INTR_MASKED	Interrupt masked request register	Bitwise AND of interrupt request and mask registers

Section F: Analog System

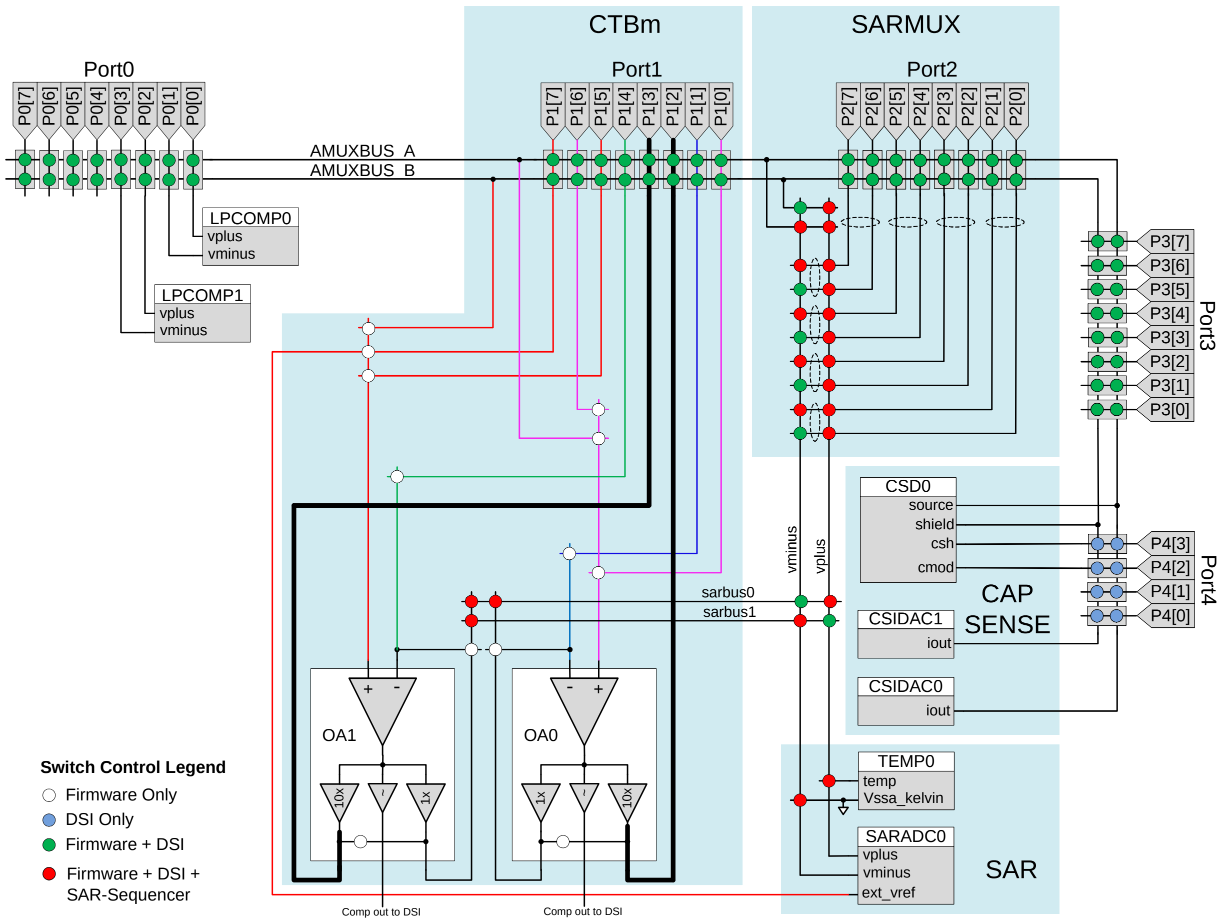
This section encompasses the following chapters:

Top Level Architecture

Analog System Block Diagram



Analog Routing Diagram



18. Precision Reference

A voltage or current reference with a value that is independent of supply voltage and temperature is an essential building block of many analog circuits in PSoC® 4. For example, accurate biasing voltages are critical for many circuit schemes. In an ADC, a reference voltage is required to quantify an input. In a VIDAC, the voltage or current reference is required to define the output full-scale range.

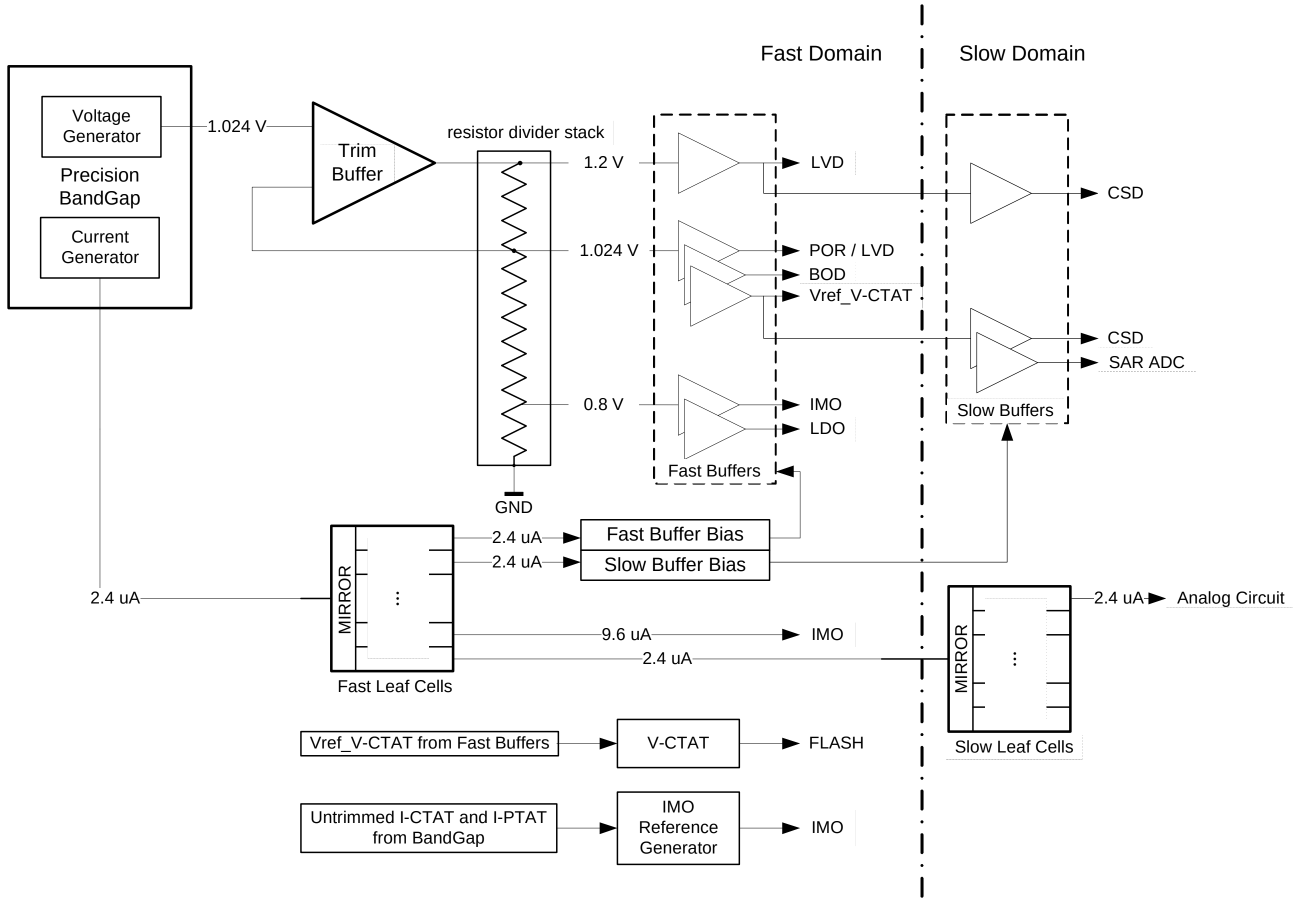
18.1 Block Diagram

PSoC 4 has a precision reference block, which creates multiple precision reference bias currents and voltages for the whole chip. Figure 18-1 illustrates the block diagram.

The precision reference is mainly composed of these blocks:

- A precision bandgap block, which generates the precision voltage and current references
- A trim buffer, which generates different output voltage references for various applications and trims the voltage magnitude of 1.024-V output
- A group of fast low-power buffers and slow low-power buffers, which not only enhance the drive capability of various reference outputs, but also isolate the noise from one another
- A group of fast leaf cells and slow leaf cells, which create multiple copies of current references in fast domain and slow domain, respectively
- A V-CTAT block, which provides a temperature-dependent voltage reference for the flash system
- A temperature trimmable current source, which generates the temperature-independent current reference for the IMO

Figure 18-1. Voltage Reference Block Diagram



18.2 How it Works

The work principles of the main components are detailed in this section.

18.2.1 Precision Bandgap

The principle of the bandgap circuit relies on two groups of diode-connected bipolar junction transistors running at different emitter current densities. By canceling the negative temperature dependence of the PN junctions in one group of transistors with the positive temperature dependence from a Proportional-to-Absolute-Temperature (PTAT) circuit (which includes the other group of transistors), a DC voltage that changes very little with variations in temperature is generated. In this block, the current reference is also provided.

18.2.2 Trim Buffer

The trim buffer is used to trim the voltage magnitude of the 1.024-V reference output. Besides, different output voltage references are generated by this buffer for various applications.

18.2.3 Low-Power Buffers

Due to the high-impedance nature of the trim buffer outputs, low-power buffers are used to drive each of the outputs to the destination blocks. They also act as isolation cells between various references.

Note that these low-power buffers are divided into fast buffers and slow buffers, which drive fast domain and slow domain in the chip, respectively. This design is to achieve faster references settling time for the system. If all the voltage references driven by the low-power buffers remain in the same domain, it causes high capacitive loads on the bias lines due to large number of buffers, which in turn increases the settling time. In practice, only a few blocks are required to ensure the system start up (such as flash, voltage monitors, and power system), so a separate fast start up domain is created for these voltage references who serve these blocks.

The fast domain starts up along with the bandgap and the slow domain starts up following the fast one.

The fast buffers are directly driven by trim buffer; meanwhile, a second cascaded layer of voltage buffers (slow buffers) are driven by the fast buffers. This topology also ensures that the extra load due to the non-startup related blocks in slow domain are completely isolated from fast domain.

18.2.4 Leaf Cells

Except for the voltage references, the bandgap provides a unit current of 2.4 µA, which is also the second order curvature corrected. The current reference goes through the leaf cells where multiple current references are generated. Two 2.4-µA current references are through fast and slow buffer bias module to generate bias voltage for fast buffers and slow buffers, respectively.

The leaf cells are also divided into fast leaf cells and slow leaf cells to achieve faster reference settling time. Through the fast leaf cells, the unit current of 2.4 µA generates the fast current references for the fast domain. One of the fast current references is input into slow leaf cells to generate all slow current references.

Settling time of fast references of voltage and current is 9 µs, which is the time to settle within 1 percent of the final value. Settling time of slow references is 40 µs. All the generated voltage and current references in precision reference block are summarized in Table 18-1.

Table 18-1. Voltage and Current References

Voltage or Current References	Accuracy Targets	Potential Block Usage/Destination
1.2 V	±2%	LVD – Low voltage detect on external supply
1.2 V	±2%	Capsense reference
1.024 V	±1%	SAR ADC
1.024 V	±2%	Flash
1.024 V	±2%	BOD – To detect brownouts on internal voltages
1.024 V	±2%	Capsense reference
0.8 V	±2%	IMO – Comparator threshold in relaxation oscillator
0.8 V	±2%	LDO – V_{CCD} and V_{CCA} regulator reference
2.4 µA	±2.5%	Bias current for analog circuits
3 µA	±2.5%	IREF for flash macro
9.6 µA	±5%	IREF for IMO, with programmable tempco

18.2.5 V-CTAT Block

The V-CTAT block provides a temperature-dependent voltage reference to ensure flash reliability. Its output voltage is complementary to ***ambient temperature*** (CTAT). Linear variation range of output voltage over the temperature range of –40 ºC to 150 ºC with reference to output voltage at 55 ºC is from ±0 to ±15 percent.

18.2.6 IMO Reference Generator

This generator produces a separate trimmable current reference for the internal main oscillator (IMO) block. It is implemented to cancel the temperature drift of the clock frequency so as to achieve the ±2 percent accuracy. The CTAT and PTAT current outputs of the bandgap block are used for this purpose. See Figure 18-1.

18.3 Configuration

During power-up, the precision reference block is initialized with default trim settings saved in nonvolatile latch (NVL) and SFLASH. These settings are programmed during manufacturing and no field adjustment is needed.

19. SAR ADC

The PSoC® 4 has one successive approximation register analog-to-digital convertor (SAR ADC). The SAR ADC is designed for applications that require moderate resolution and high data rate. It consists of the following blocks (see Figure 19-1):

- SARMUX
- SAR ADC core
- SARREF
- SARSEQ

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

The ninth channel is an injection channel that is used by firmware for infrequent and incidental sampling of pins and signals, for example, the internal temperature sensor.

The result from each channel is double-buffered and a complete scan may be configured to generate an interrupt at the end of the scan. Alternatively, the data can be routed to programmable digital blocks (UDBs) for further processing without CPU intervention. 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 UDBs or firmware. This makes it possible to implement an alternative sequencer with the UDBs or firmware.

19.1 Features

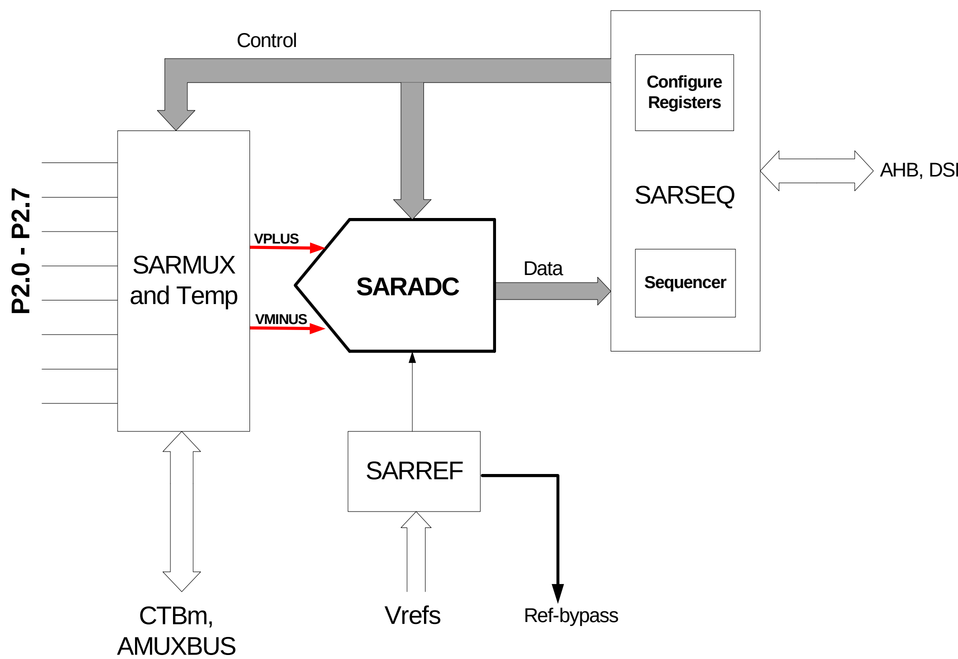
- Wide operation voltage range: 1.71 V to 5.5 V
- Maximum 1 Msps sample rate
- Eight individually configurable channels and one injection channel
- Per channel
  - Input from external pin or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Up to four programmable acquisition times
  - Default 12-bit resolution, selectable alternate resolution: either 8-bit or 10-bit
  - Single-ended or differential measurement
  - Averaging
  - Results are double-buffered
  - Result may be left or right aligned
- Scan triggered by firmware, timer, pin, or UDB
  - One shot–periodic or continuous mode
- 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 1.024-V reference with buffer
- External reference

- Interrupt generation
  - Finished scan conversion
  - Per channel saturation detect and over-range (configurable) detect
  - 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
- Option to process data in programmable digital blocks to off-load CPU
- Option to control switches from programmable digital blocks
- Option to control SAR ADC and switches from programmable digital blocks
  - Implement an alternative SAR sequencer
  - Able to achieve 1 Msps
- Low-power modes
  - ADC core and reference voltage has low-power mode separately

19.2 Block Diagram

Figure 19-1. Block Diagram



19.3 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 mode
  - Register mode
  - DSI mode
- Configuration examples

19.3.1 SAR ADC Core

PSoC 4 SAR ADC core is a 12-bit SAR ADC. Maximum sample rate for this ADC is 1 Msps operating at 18-MHz clock for PSoC 4200 and 806 ksps operating at 14.5 MHz for PSoC 4100.

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

19.3.1.1 Single-ended and Differential Mode

PSoC 4 SAR ADC can operate in single-ended and differential mode. 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 channel configuration register, SAR_CHANx_CONFIG.

The single-ended mode has six options of negative input: V_{SSA}, V_{REF}, P2.1, P2.3, P2.5, and P2.7. It is configured by the global configuration register SAR_CTRL. When Vminus is connected to P2.1..P2.7, the single-ended mode is equivalent to differential mode. Note that temperature sensor can only be used in single-ended mode; it will override the SAR_CTRL [11:9] to 0. The differential conversion is not available for temperature sensors; the result is undefined.

19.3.1.2 Input Range

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

Note that Vn ± V_{REF} should be in the range of V_{SSA} to V_{DDA}. For example, if negative input is connected to V_{SSA}, positive input range is 0 to V_{REF}, not $–V_{REF}$ to V_{REF}. This is because the signal cannot go below V_{SSA}. Only 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.

19.3.1.3 Result Data Format

Result data format is configurable from two aspects:

- Singed/unsigned
- Left/right alignment

When the result is considered signed, the most significant bit of the conversion is used for sign extension to 16 bits with MSB. For an unsigned conversion, the result is zero extended to 16-bits. It can be configured by SAR_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 bits of the result register. By default, data is right-aligned in data[11:0], with sign extension to 16 bits, if required. A lower resolution combined with left-alignment will cause lower significant bits 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 shown as follows.

Table 19-1. 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	–	–	–	–	–	–	–	–

19.3.1.4 Negative Input Selection

The negative input connection choice affects the voltage range, SNR, and effective resolution (see Table 19-2). In single-ended mode, negative input of the SAR ADC can be connected to V_{SSA}, V_{REF}, or P2.1/P2.3/P2.5/P2.7.

Table 19-2. 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 4 pin. Because of this, the global configuration bit SINGLE_ENDED_SIGNED (SAR_SAMPLE_CTRL[2]) will be ignored and the result is always (0x000-0x7FF).

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 single-ended conversions with Vminus connected to P2.1, P2.3, P2.5, or P2.7 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 measured signal value (SARMUX.vplus) cannot go below ground.

19.3.1.5 Resolution

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. Lower resolution results in higher conversion rate.

19.3.1.6 Acquisition Time

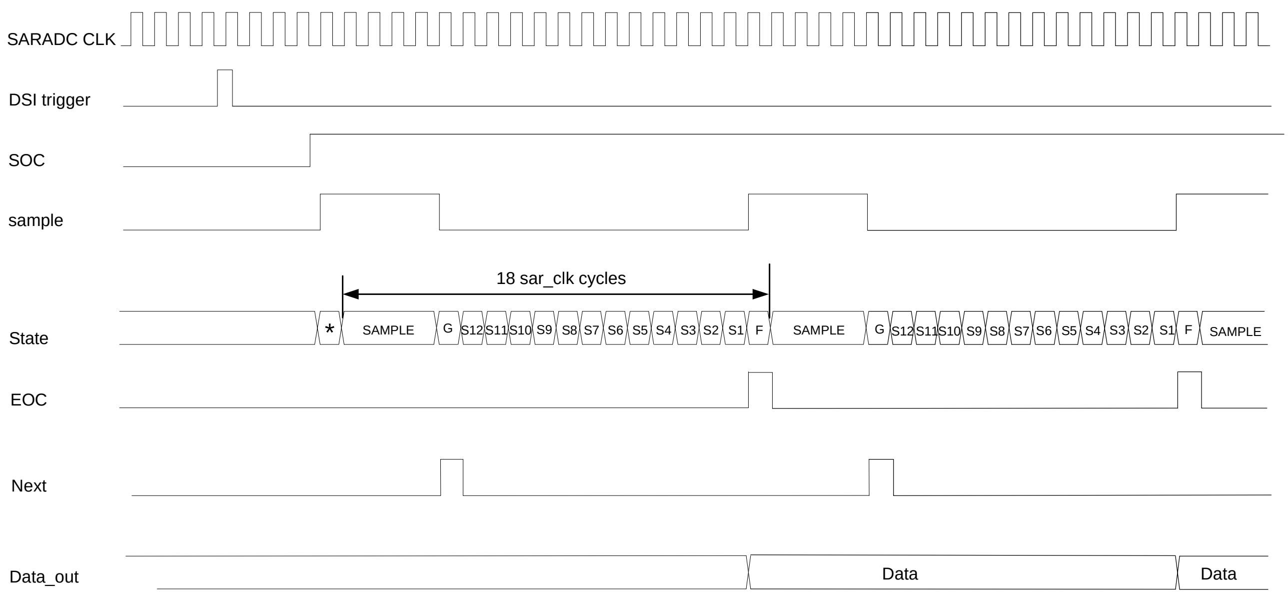
Acquisition time is the time taken by sample and hold (S/H) circuit inside SAR ADC to settle. After acquisition time, the input signal source is disconnected from the SARADC core, and the output of the S/H circuit will be used for conversion. Each channel can select one from four acquisition time options, from 4 to 1023 SAR clock cycles defined in global configuration registers SAR_SAMPLE_TIME01 and SAR_SAMPLE_TIME23.

19.3.1.7 SAR ADC Clock

SAR ADC clock frequency must be between 1 MHz and 18 MHz for PSoC 4200 and 1 MHz to 14.5 MHz for PSoC 4100, which comes from the IMO via a clock divider. Note that a fractional divider is not supported for SAR ADC. To get a 1-Msps sample rate, an 18-MHz SAR ADC clock is required. To achieve this, the system clock (IMO) must be set to 36 MHz rather than 48 MHz. To get a 806-ksps sample rate for the PSoC 4100 device, IMO must be set to 29 MHz. A 12-bit ADC conversion with the default acquisition time of four clocks requires 18 clocks in which to complete. A 10-bit and 8-bit conversion requires 16 and 14 clocks respectively.

19.3.1.8 SAR ADC Timing

Figure 19-2. SAR ADC Timing



As the timing graph shows, there is a sar_clk delay before raising start-of-conversion (SOC). 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 (or dsi_sample_done), the SARMUX can route to other pin and signal, it will be done automatically with sequencer control (see SARSEQ on page 205 for details).

19.3.2 SARMUX

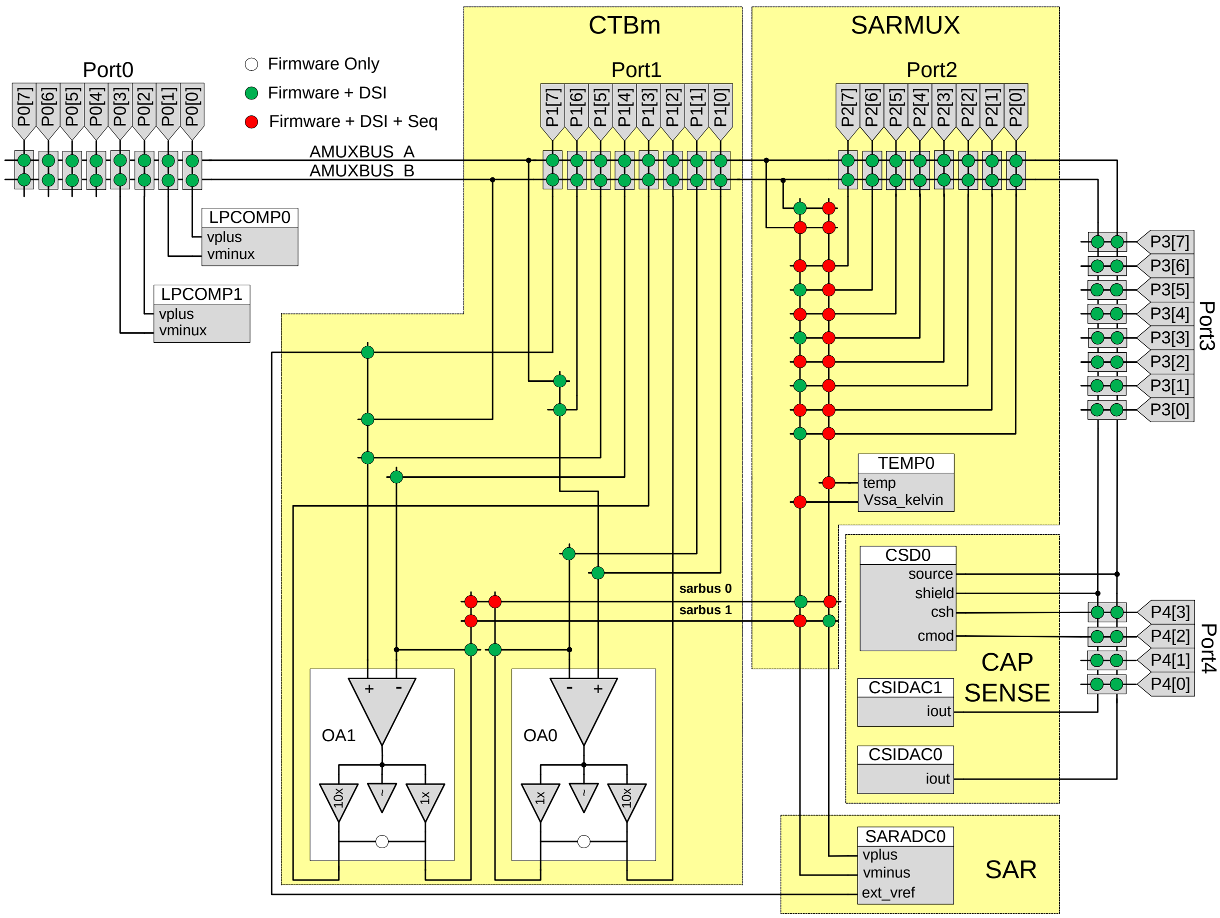
SARMUX is an analog dedicated programmable multiplexer. The main features of SARMUX are:

- Switch on resistance: 600 Ω (maximum)
- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ), UDBs, or firmware.
- Charge pump inside:
  - If V_{DDA} < 4.0 V, charge pump should be turned on to reduce switch resistance
  - If V_{DDA} >= 4.0 V, charge pump is turned off and delivers V_{DDA} as its output
- Multiple inputs:
  - Analog signals from pins (port 2)
  - 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)

19.3.2.1 Analog Routing

SARMUX has many switches that may be controlled by SARSEQ block (sequencer controller), firmware, or the DSI. Sequencer and DSI are the hardware control method, which can be masked by the hardware control bit in the register, SAR_MUX_SWITCH_HW_CTRL. Different control methods have different control capability on the switches. See Figure 19-3.

Figure 19-3. SARMUX Switches and Control Capability



Sequencer control: The switches are controlled by the sequencer in 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 19-3. The corresponding registers are: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_CTRL, and SAR_MUX_SWITCH_HW_CTRL. The detailed configuration is available in register mode; see Set SARMUX Analog Routing on page 216.

Firmware control: Programmable registers directly define the VPLUS/VMINUS connection. It can control every switch in SARMUX; see Figure 19-3. 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: SAR_MUX_SWITCH0, SAR_MUX_SWITCH_HW_CTRL. and SAR_CTRL. The detailed configuration is available in register mode; see Set SARMUX Analog Routing on page 216.

DSI control: Switches are controlled by DSI signals from the UDB, which can act as a secondary sequencer with a customized logic design. DSI can control most switches, except some design for test (DFT) switches. Thus, it can do a differential measurement between any two pins and signals and firmware control. The detailed configuration is available in DSI mode; see Set SARMUX Analog Routing on page 213.

19.3.2.2 *Analog Interconnection*

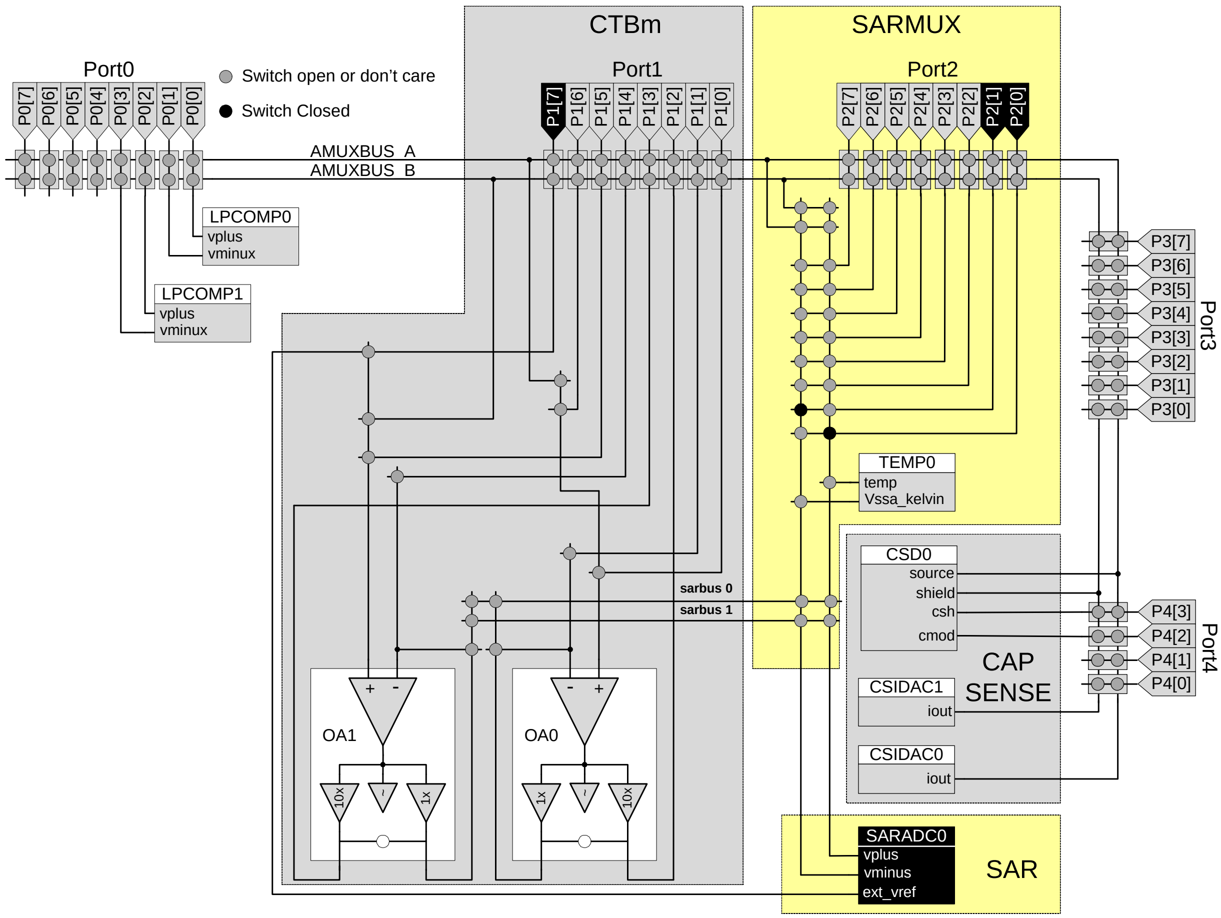
PSoC 4 analog interconnection is very flexible. SAR ADC can be connected to multiple inputs via SARMUX, including both external pins (port 2) and internal signals. For example, it can connect to a neighboring block such as CTBm through a pair of wires, sarbus0 and sarbus1. It can also connect to other 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 analog interconnection.

Input from External Pins

Figure 19-4 shows how P2.0 and P2.1 are connected to SAR ADC as a differential pair (Vpuls/Vminus) via switches. These two switches can be controlled by sequencer, firmware, or DSI. However, if P2.1 and P2.2 need to be used as differential pair, sequencer does not work; use firmware or DSI.

Figure 19-4. Input from External Pins

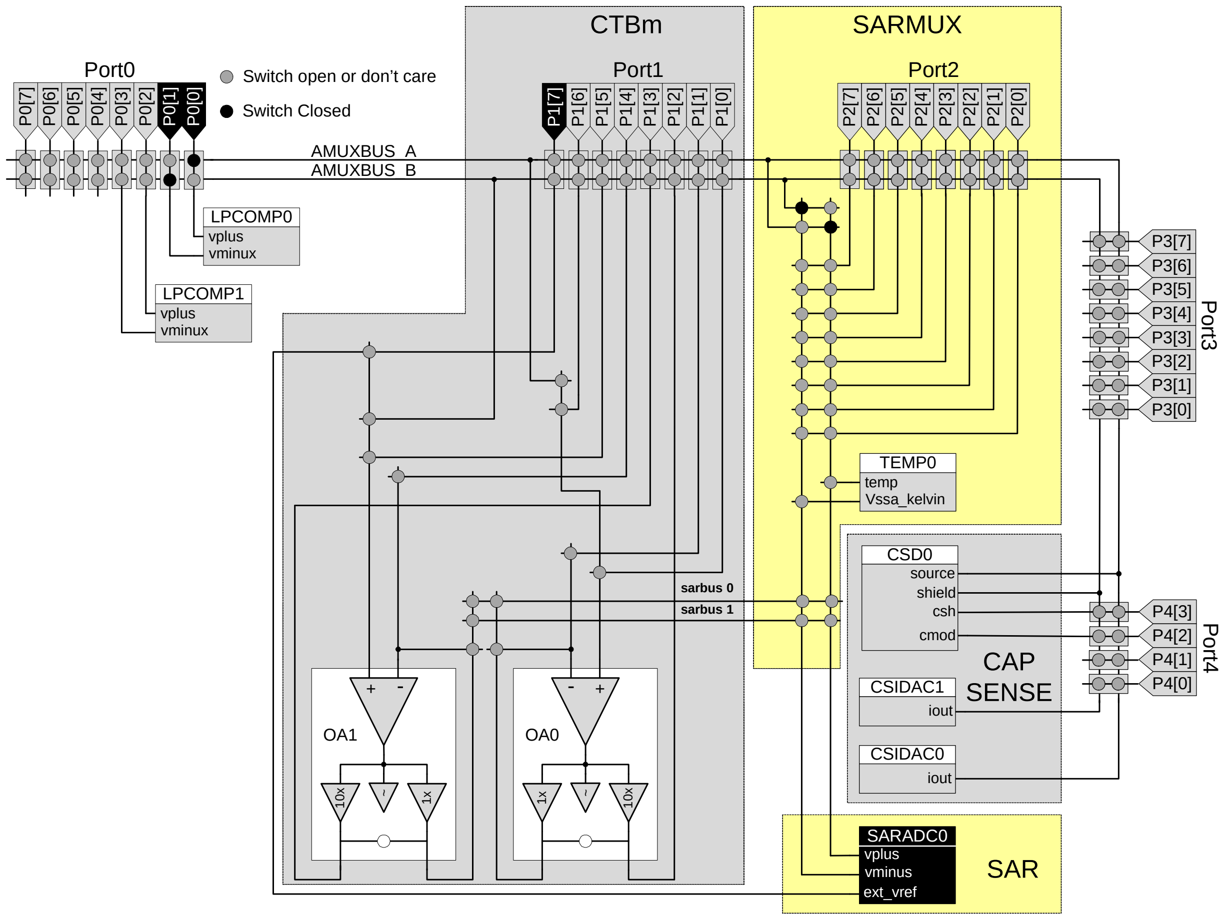


Input from Analog Bus (AMUXBUS_A/_B)

Figure 19-5 shows how P0.0, P0.1 are connected to ADC as differential pair. Additional switches must connect P0.0 and P0.1 to two analog buses: 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; use port 2, if possible.

Figure 19-5. Input from Analog Bus



Input from CTBm Output via sarbus

SAR ADC can be connected to CTBm output via sarbus 0/1. Figure 19-6 shows how to connect an opamp (configured as a follower) output to a single-ended SAR ADC. Negative terminal is connected to V_{REF}. Figure 19-7 shows how to connect two opamp outputs to SAR ADC as a differential pair. It must connect opamp output to sarbus 0/1, then connect SAR ADC input to sarbus 0/1. Because there are also additional switches, it is not fast enough to sample at 1 Msps. However, two on-chip opamps add value for many applications.

Figure 19-6. Input from CTBm Output via sarbus

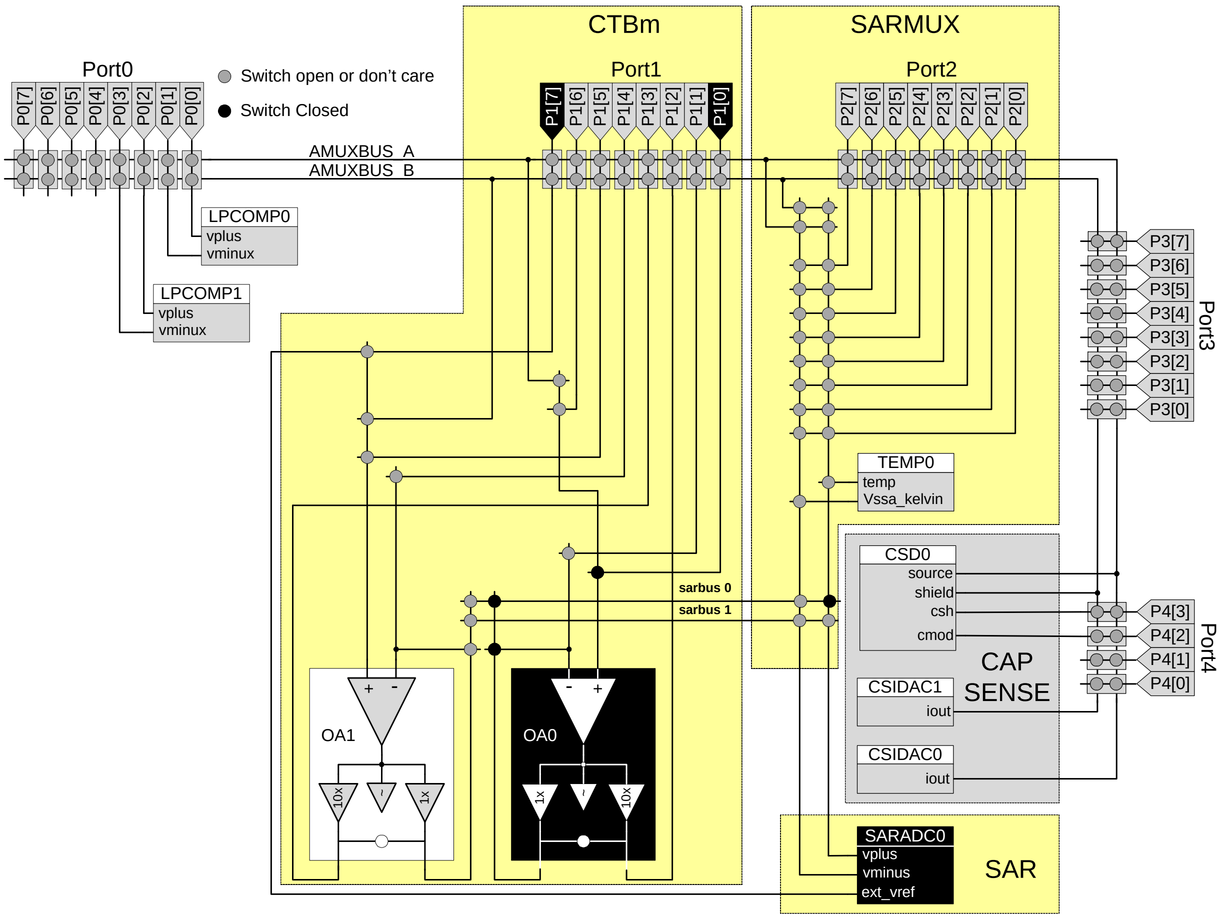
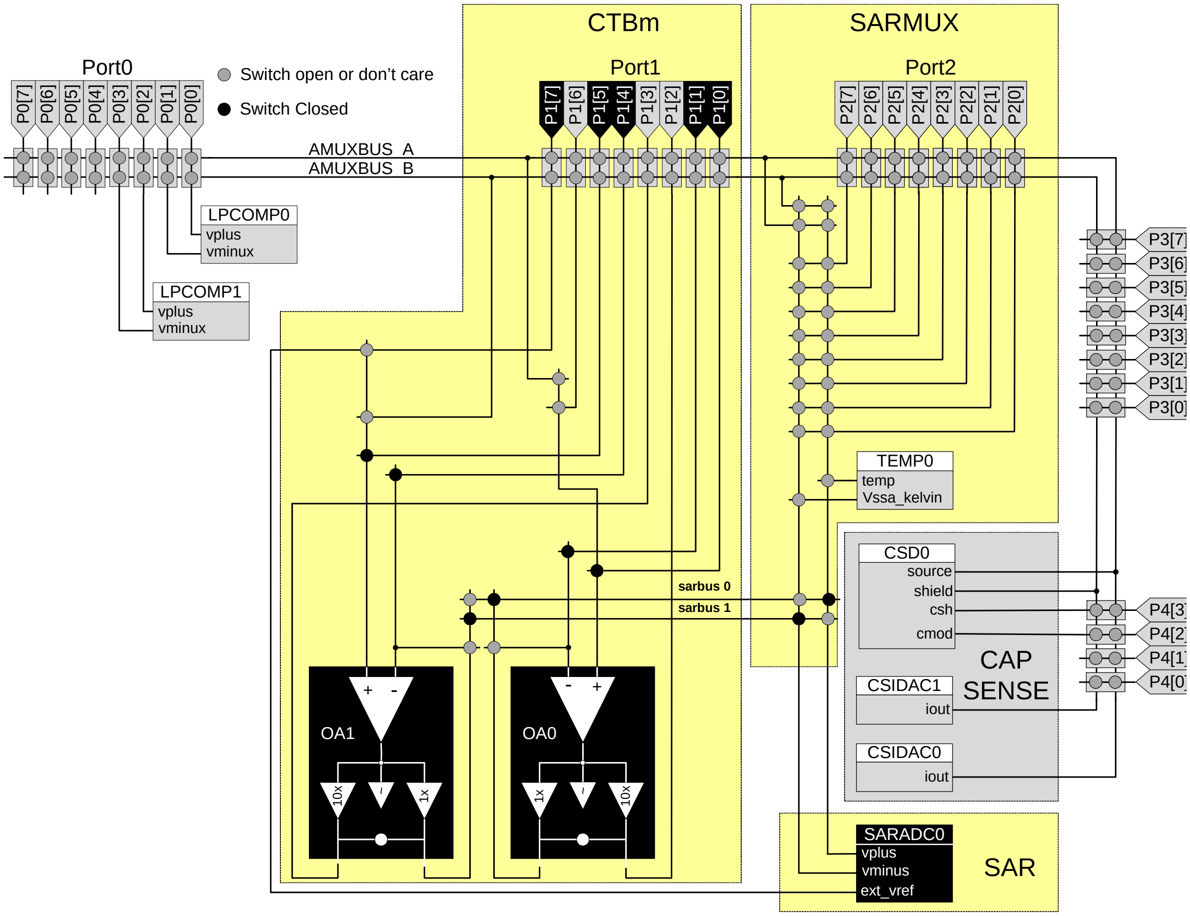


Figure 19-7. Inputs from CTBm Output via sarbus0 and sarbus1

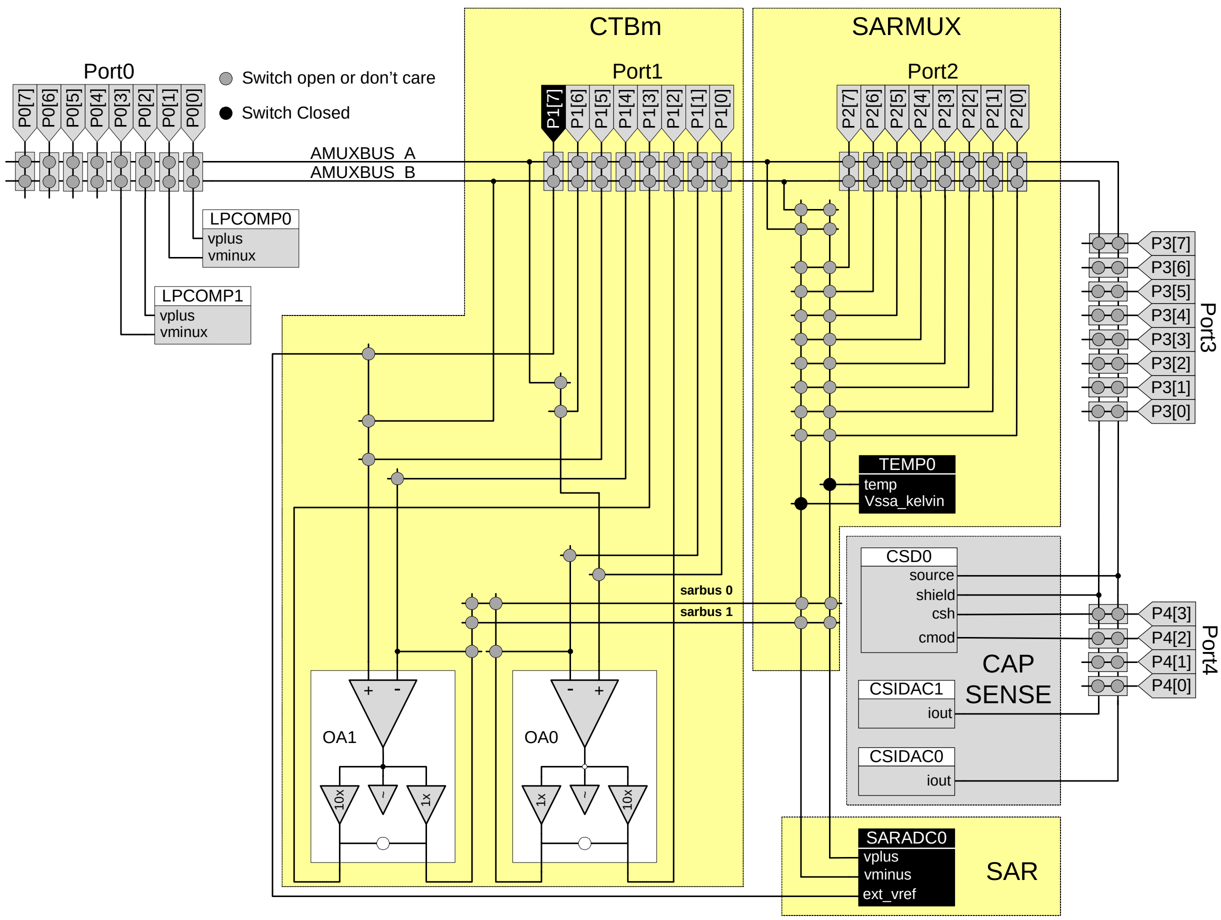


Input from Temperature Sensor

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Note for temperature sensor, differential conversions are not available (conversion result is undefined), thus always use it in singled-ended mode. Reference is from internal 1.024 V.

As Figure 19-8 shows, temperature sensor can be routed to positive input of SAR ADC via switch, which can be controlled by sequencer, firmware, or DSI. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit will disable temperature sensor by cutting its bias current.

Figure 19-8. Inputs from Temperature Sensor

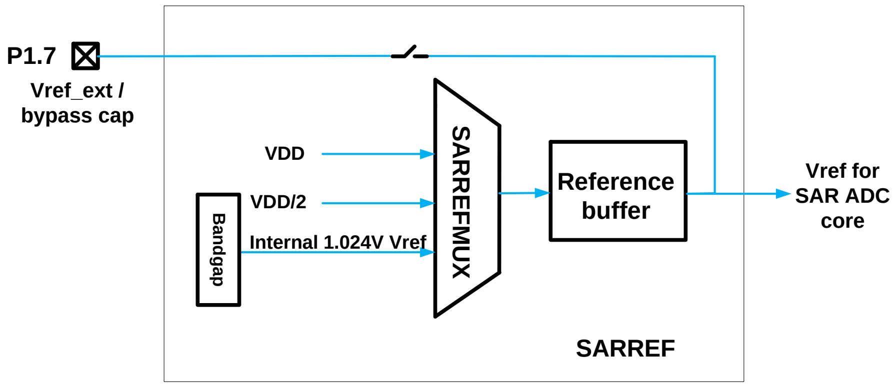


19.3.3 SARREF

The main features of SARREF are:

- Reference options: V_{DDA}, $V_{DDA}/2$, 1.024-V bandgap (±1 percent), external reference
- Reference buffer + bypass cap to enhance internal reference drive capability

Figure 19-9. SARMUX Block Diagram



19.3.3.1 Reference Options

The 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 1.024-V internal reference from a bandgap or an external V_{REF} connected to a GPIO pin, P1.7. The control for the reference mux in SARREF is in the global configuration register SAR_CTRL [6:4].

19.3.3.2 Bypass Capacitors

The internal references, 1.024 V from bandgap, $V_{DDA}/2$, or V_{DDA} are buffered with the reference buffer. This reference may be routed to P1.7 where an external capacitor can be used to filter internal noise that may exist on the reference signal.

The SAR ADC sample rate cannot exceed 166 ksps without an external reference bypass capacitor. For example, without a bypass capacitor and with 1.024-V internal V_{REF}, the maximum SAR ADC clock frequency is 3 MHz. When using an external reference, it is recommended that an external capacitor is used. Bypass capacitors can be enabled by setting SAR_CTRL [7].

Table 19-3 lists different reference modes and its maximum frequency/sample rate for 12-bit continuous mode operation.

Table 19-3. Reference Modes

Reference Mode	Reference SAR_CTRL [6:4]	Bypass Cap SAR_CTRL[7]	Buffer	Max Frequency	Max Sample Rate
1.024 V internal V_{REF} without bypass cap	4	0	Yes	3 MHz	166 ksps
1.024 V internal V_{REF} with bypass cap	4	1	Yes	18 MHz	1 Msps
External V_{REF}	5	X	No	18 MHz	1 Msps
$V_{DDA}/2$ without bypass cap	6	0	Yes	3 MHz	166 ksps
$V_{DDA}/2$ with bypass cap	6	1	Yes	18 MHz	1 Msps
V_{DDA}	7	X	Yes	18 MHz	1 Msps

1.024-V internal V_{REF} startup time varies with the different bypass capacitor size, Table 19-4 lists two common values for the bypass capacitor and its startup time specification. If reference selection is changed between scans, make sure the 1.024-V internal V_{REF} is settled when SAR ADC starts sampling.

Table 19-4. Bypass Capacitor Values

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

19.3.3.3 Input Range versus Reference

All inputs should be in the range of V_{SSA} to V_{DDA}. Input voltage range is limited by V_{REF} selection. If negative input is Vn, ADC reference is V_{REF} and the positive input range 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_{SS} to V_{DD}.

19.3.4 SARSEQ

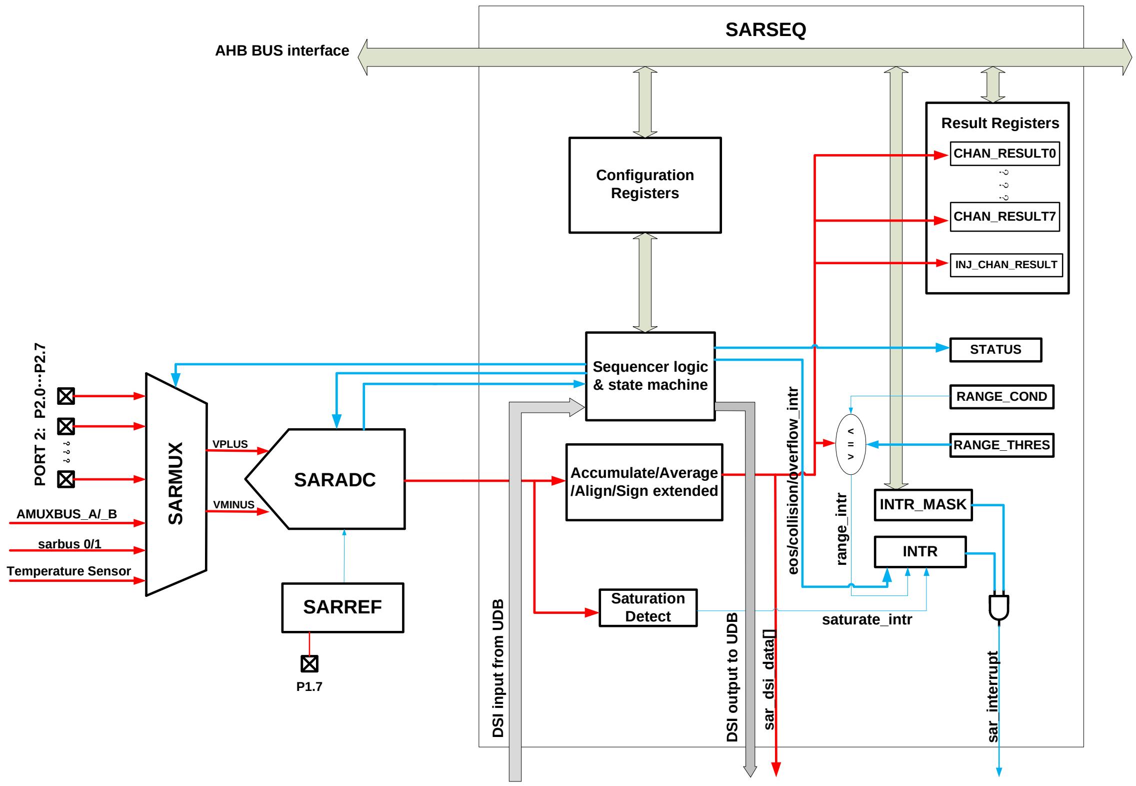
SARSEQ is a dedicated sequencer controller that automatically sequences the input mux from one channel to the next while placing the result in an array of registers, one per channel.

- Control SARMUX analog routing automatically without CPU intervention
- Control SAR ADC core (such as resolution, acquisition time, and reference)
- Receive data from SAR ADC and pre-process (average, range detect)
- Results are double-buffered so the CPU can safely read the results of the last scan while the next scan is in progress.

The features of SARSEQ are:

- Eight channels can be individually enabled as an automatic scan without CPU intervention
- A ninth channel (injection channel) for infrequent signal to insert in an automatic scan
- Per channel selectable
  - Input from external pin or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Up to four programmable acquisition time
  - Default 12-bit resolution, selectable alternate resolution: either 8-bit or 10-bit
  - Single-ended or differential mode
  - Result averaging
- Scan triggering
  - One shot, periodic, or continuous mode
  - Triggered by any digital signal or input from GPIO pin
  - Triggered by internal UDB of fixed-function block
  - Software triggered
- 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
  - Per control mode, each channel saturation detect
  - Per channel over range (configurable) detect
  - 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

Figure 19-10. SARSEQ Block Diagram



19.3.4.1 Averaging

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

In register control mode, channel configuration SAR_CHAN_CONFIG register has an enable bit (AVG_EN) to enable averaging. In DSI control mode, average is enabled by dsi_cfg_average signal.

In global configuration, AVG_CNT (SAR_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 (SAR_SMAPLE_CTRL [6:4]) = 3, then N = 16.

AVG_SHIFT bit (SAR_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. Because the conversion result is 12-bit and the maximum value of N is 256 (left shift 8 bits), the 20-bit accumulator will never overflow.

If AVG_SHIFT in SAR_SAMPLE_CTRL register is set, the accumulated result is shifted right AVG_CNT + 1 bits to get averaged. If it is not, the result is forced to shift right to ensure 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 in this case, the average result is bigger than expected; it is recommended to set AVG_SHIFT.

After shifting, the result is stored in the 16-bit result register after sign extended for sign conversion. Averaging always uses the maximum resolution 12-bit and right-alignment – the RESOLUTION and LEFT_ALIGN bits of the channel are ignored.

19.3.4.2 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 SAR_RANGE_THRES register. The RANGE_LOW field (SAR_RANGE_THRES [15:0]) value defines the lower threshold and RANGE_HIGH field (SAR_RANGE_THRES [31:16]) defines the upper threshold of the range.

The SAR_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 209 for details.

19.3.4.3 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's application. Allow sufficient time for the firmware to read the previous scan before the present scan is completed. Failing to do so may result in corrupted data. All input channels are double buffered with 16 registers, except the injection channel. The injection channel is not required to be doubled buffered because it is not normally part of a normal channel scan.

19.3.4.4 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 SAR is operating in continuous mode, enabling the injection channel will change the sample rate.

The injection channel can only be controlled by the firmware with a firmware trigger (one-shot). This means the injection channel does not support continuous or DSI trigger. It also does not support output of its data or interrupt to the DSI bus. Because 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 SAR_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 end-of-conversion interrupt INJ_EOC_INTR
- Maskable range detect interrupt INJ_RANGE_INTR
- Maskable saturation detect interrupt INJ_SATURATE_INTR
- Maskable collision interrupt INJ_COLLISION_INTR

SAR_INTR, SAR_INTR_MASK, SAR_INTR_MASKED, and SAR_INTR_SET are the corresponding registers.

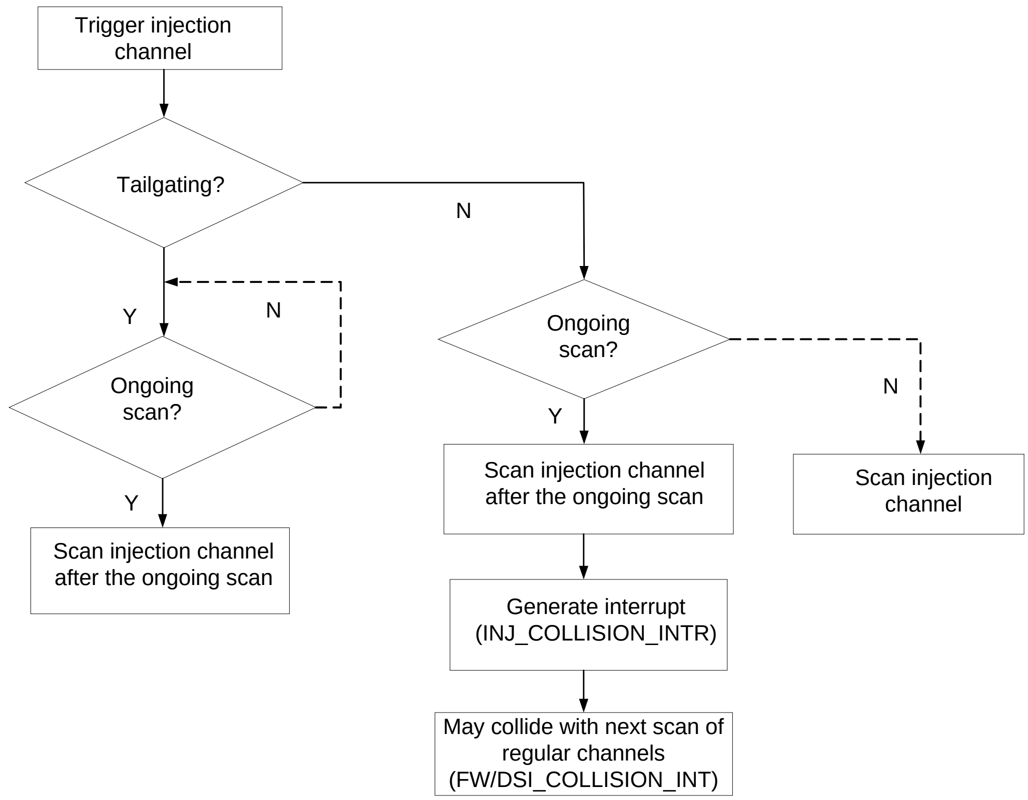
These features are described in detail in Set Global SARSEQ Configuration on page 213, Set Channel Configurations on page 214, and Interrupt on page 209.

Tailgating

The injection channel conversion can be triggered by setting the start or enable bit INJ_START_EN (SAR_INJ_CHAN_CONFIG [31]). If there is an ongoing scan, it is recommended to select tailgating by setting INJ_TAILGATING=1 (SAR_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. In this case, tailgating is not necessary.

If tailgating is not selected, the injection channel will also be scanned at the end of the ongoing scan of regular channels, 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/DSI_COLLISION_INTR is raised). The regular scan is postponed until the injection scan is finished, thus causing jitter on a regular scan. Note that continuous trigger and DSI trigger level mode will never trigger a collision interrupt.

Figure 19-11. Injection Channel Flow Chart



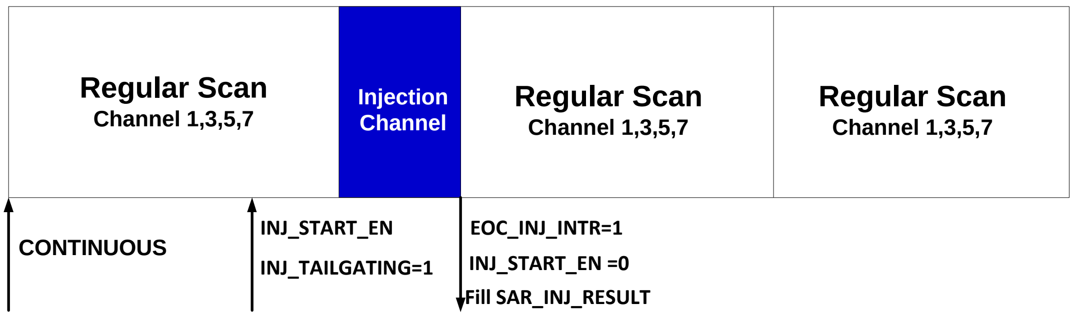
The disadvantage of tailgating is that it may be a long time before the next trigger occurs. If there is no risk of colliding or causing jitter on the regular channels, the injection channel can be used safely without tailgating.

After completing the conversion for the injection channel, the end-of conversion interrupt (INJ_EOC_INTR) is set and the INJ_START_EN bit is cleared. The conversion data of the injection is put in the SAR_INJ_RESULT register. Similar to the SAR_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 19-12 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 19-12. Injection Channel Enabled with Tailgating



19.3.5 Interrupt

Each of the interrupts described in this section has an interrupt mask in the SAR_INTR_MASK register. By making the interrupt mask low, the corresponding interrupt source is ignored. The SAR interrupt is generated if the interrupt flag is high and the corresponding interrupt source is pending.

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

The SAR_INTR_MASKED register is the logical AND between the interrupts sources and the interrupt mask. This 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) is used to trigger each interrupt. This allows the firmware to generate an interrupt without the actual event occurring.

19.3.5.1 End-of-Scan Interrupt (EOS_INTR)

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

Optionally, the EOS_INTR can also be sent out on the DSI bus by setting the EOS_DSI_OUT_EN bit in SAR_SAMPLE_CTRL [31]. The EOS_INTR signal is maintained on the DSI bus 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 SAR_INTR_MASK register. EOS_MASKED bit of the SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a ‘1’ to EOS_SET bit in SAR_INTR_SET register can set the EOS_INTR, which is intended for debug and verification.

19.3.5.2 Overflow Interrupt

If a new scan completes and the hardware tries to set the EOS_INTR and EOS_INTR as 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. The old data will be overwritten.

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

19.3.5.3 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 DSI trigger (DSI_COLLISION_INTR), and for the injection channel (INJ_COLLISION_INTR). This allows the firmware to identify which trigger collided with an ongoing scan.

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

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

19.3.5.4 Injection End-of-Conversion Interrupt (INJ_EOC_INTR)

After completing a conversion for the injection channel, the injection end-of-conversion 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, the 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 SAR_INTR_MASK register. The INJ_EOC_MASKED bit of SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a ‘1’ to the INJ_EOC_SET bit in SAR_INTR_SET register can set INJ_EOC_INTR, which is intended for debug and verification.

19.3.5.5 Range Detection Interrupts

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.

Range detection interrupt for a specified channel can be masked by setting the SAR_RANGE_INTR_MASK register specified bit to ‘0’. Register SAR_RANGE_INTR_MASKED reflects a bitwise AND between the interrupt request and mask registers. If the value is not zero, then the SAR interrupt signal to the NVIC is high.

SAR_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).

19.3.5.6 Saturate Detection Interrupts

The saturation detection is always applied to every conversion. This feature detects if a sample value is equal to the minimum or the maximum value for the specific resolution. If it is, a maskable interrupt flag is set for the corresponding channel. This allows the firmware to take action, such as discarding the result, when the SAR ADC saturates. The sample value is tested right 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.

Saturation interrupt flag is set immediately to enable a fast response to saturation, before the full scan and averaging. Saturation detection interrupt for specified channel can be masked by setting the SAR_SATURATE_INTR_MASK register specified bit to ‘0’. SAR_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.

SAR_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.

19.3.5.7 Interrupt Cause Overview

INTR_CAUSE register contains an overview of all the pending SAR interrupts. It allows the ISR to determine the interrupt cause by reading this register. The register consists of a mirror copy of SAR_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).

19.3.6 Trigger

The three possible ways to trigger a scan are:

- A firmware or one-shot trigger is generated when the firmware writes to the FW_TRIGGER bit of the SAR_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.
- A periodic trigger comes in over the DSI connections (dsi_trigger). This trigger is connected to the output of a TCPWM; however, it can also be connected to any GPIO pin or a UDB. The UDB can implement a state machine looking for a certain sequence of events.
- A continuous trigger is activated by setting the CONTINUOUS bit in SAR_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 a DSI trigger coincides with a firmware trigger, the DSI trigger is handled first and a separate scan is done for the firmware trigger (and a collision interrupt is set). When a DSI trigger coincides with a continuous trigger, both triggers are effectively handled at the same time (a collision interrupt may be set for the DSI trigger).

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

19.3.6.1 DSI Trigger Configuration

- DSI Synchronization

The DSI interface of SARSEQ runs at the system clock frequency (clk_sys); see Clocking System chapter on page 61 for details. If the incoming DSI trigger signal is not synchronous to the AHB clock, the signal needs to be synchronized by double flopping it (default). However, if the DSI trigger signal is already synchronized with the AHB clock, then these two flops can be bypassed. The configuration bit DSI_SYNC_TRIGGER controls the double flop bypass. DSI_SYNC_TRIGGER affects the trigger width (TW) and trigger interval (TI) requirement of the DSI pulse trigger signal.

- DSI Trigger Level

The DSI trigger can either be a pulse or a level; this is indicated by the configuration bit DSI_TRIGGER_LEVEL. If it is a level, then the SAR starts new scans for as long as the DSI trigger signal remains high. When the DSI trigger signal is a pulse input, a positive edge detected on the DSI 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 configuration, the transmission time is different; Table 19-5 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 DSI trigger is ignored.

Table 19-5. DSI Trigger Maximum Time

Maximum DSI_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 19-6. Trigger Signal Requirement

Trigger Spec	Requirement
Trigger Width (TW)	TW should be greater 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 of the DSI pulse trigger signal should be longer than the transmission time (as specified in Table 19-5); otherwise, the second trigger pulse will be ignored.

19.3.7 SAR ADC Status

The current SAR status can be observed through the BUSY and CUR_CHAN fields in the SAR_STATUS register. The BUSY bit is high whenever the SAR is busy sampling or converting a channel; the CUR_CHAN bits indicates the current channel. SW_VREF_NEG bit indicates the current switch status, including DSI and register controls, of the switch in the SAR ADC that shorts NEG with V_{REF} input.

CHAN_WORK_VALID register indicates the channel that is sampled during the current scan. CHAN_RESULT_VALID register indicates the channel that is sampled during the last scan. When CHAN_RESULT_VALID is set, the corresponding CHAN_WORK_VALID bit is cleared. The CUR_AVG_ACCU and CUR_AVG_CNT fields in the SAR_AVG_STAT register indicate the current averaging accumulator contents and the current sample counter value for averaging (counts down).

SAR_MUX_SWITCH_STATUS register gives the current switch status of MUX_SWITCH0 register.

These status registers help to debug SAR behavior.

19.3.8 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 operation. The easiest way is to reduce the trigger frequency; that is, reduce the number of conversions per second.

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 19-7. 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

The V_{REF} buffer (if in use) can be set to one of four power levels. It limits the maximum clock frequency if there is no bypass capacitor for internal reference. If there is an external bypass capacitor or the reference is external, the maximum clock frequency is 18 MHz.

Table 19-8. V_{REF} Buffer

PWR_CTRL_VREF [1:0]	Need Bypass Capacitor	Relative Power [%]	Maximum Frequency [MHz]	Minimum Sample Time [cycles]	Maximum Sample Speed (at 12-bit) [ksps]
0	N	100	3	2	187.5
1	N	50	1.5	1	100
2	N	33	1	1	66.6
3	Y	25	18	4	1000

Finally, to reduce power, use a lower resolution on channels that do not need high accuracy. This shortens the conversion by up to four out of 18 cycles (for 8-bit resolution and minimum sample time).

19.3.9 System Operation

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

1. Set SAR ADC control mode: 19.3.10 Register Mode or 19.3.11 DSI Mode
2. Set SARMUX analog routing (pin/signal selection) via sequencer/firmware/DSI
3. Set the global SARSEQ conversion configurations
4. Configure each channel source (such as pin address)
5. Enable the channels
6. Set the trigger type
7. Set interrupt masks

8. Start the trigger source
9. Retrieve data after each end of conversion interrupt
10. Do injection conversions if needed

Register mode means using registers to control the SARMUX and SAR ADC conversion; DSI mode means using DSI from UDB to control. The major difference between these two control modes is shown in Table 19-9. DSI mode can be enabled by setting DSI_MODE bit (SAR_CTRL [29]).

Table 19-9. Difference between Control Modes

Control Mode	Register	DSI
DSI_MODE	0	1
SARMUX control	Sequencer control registers: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL SAR_CTRL Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL	DSI signal control signals: dsi_out, dsi_oe,dsi_swctrl, dsi_sw_negvref Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL
Global configuration	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND
Channel configuration	Channel configure registers: CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG	By DSI signal: dsi_cfg_st_sel, dsi_cfg_average, dsi_cfg_resolution, dsi_cfg_differential (CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG are ignored)
Trigger	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])
Interrupt	All Apply	All Apply (only EOS_INTR, RANGE_INTR, SATURATE_INTR output on DSI signal)
DSI output	Support	Support
Result data	8 channel result registers 1 injection channel result register	Only channel0 result register is available
Injection	Support	Not supported
Average	Support average on one PIN/signal	Support average on different PIN/signal

19.3.10 Register Mode

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

19.3.10.1 Set SARMUX Analog Routing

In register mode, 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 MUX_SWITCH_HW_CTRL register and the firmware control bits in 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 SAR_CHANx_CONFIG [6:4] and PIN_ADDR bits are SAR_CHANx_CONFIG [2:0]. Table 19-10 shows the PORT_ADDR and PIN_ADDR setup with corresponding SARMUX selection. The unused port/pins are reserved for other products in the PSoC 4 series.

Table 19-10. PORT_ADDR and PIN_ADDR

PORT_ADDR	PIN_ADDR	Description
0	0..7	8 dedicated pins of the SARMUX (P2.0-P2.7)
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 227 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. P2.0/P2.1, P2.2/P2.3, P2.4/P2.5, P2.6/P2.7 are valid differential pairs for sequencer control. More flexible analog can be implemented by firmware or DSI.

For single-ended conversions, NEG_SEL (SAR_CTRL [11:9]) is intended to decide which signal is connected to negative input. In differential mode, these bits are ignored. Negative input choice affects the input voltage range and effective resolution. See Negative Input Selection on page 196 for details. The options include: V_{SSA}, V_{REF}, or P2.1, P2.3, P2.5, and P2.7. To connect negative input to V_{REF}, an additional bit, SAR_HW_CTRL_NEGVREF (SAR_CTRL[13]) must be set, because the MUX_SWITCH_HW_CTRL register does not have that hardware control bit.

Firmware Control

By default, the SARMUX operates in firmware control. VPLUS (positive) and VMINUS (negative) inputs of SAR ADC can be controlled separately by setting the appropriate bits in SAR_MUX_SWITCH0 [29:0]. Clear appropriate bits in the hardware switch control register (SAR_MUX_SWITCH_HW_CTR[n]=0). Otherwise, hardware control method (sequencer/DSI) will control the SARMUX analog routing.

SAR_CTRL register bit SWITCH_DISABLE is used to disable 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 (SAR_CTRL [11:9]) decides which signal is connected to the negative terminal (vminus) of 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 SAR_MUX_SWITCH0 should be set to control the negative input. A special case is when SAR_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 196 for details.

19.3.10.2 Set 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. Global configuration is applied to both register control and DSI control mode.

SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, and SAR_RANGE_COND are all 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 19-11. Global Configuration Registers

Configurations	Control Registers	Detailed Reference
Reference selection	SAR_CTRL[6:4]	19.3.3.1 Reference Options
Signed/unsigned selection	SAR_SAMPLE_CTRL [3:2]	19.3.1.3 Result Data Format
Data left/right alignment	SAR_SAMPLE_CTRL [1]	19.3.1.3 Result Data Format
Negative input selection in single-ended mode	SAR_CTRL[11:9]	19.3.1.4 Negative Input Selection
Resolution	SAR_SAMPLE_CTRL[0]	19.3.1.5 Resolution
Acquisition time	SAR_SAMPLE_TIME01 [25:0] SAR_SAMPLE_TIME32 [25:0]	19.3.1.6 Acquisition Time
Averaging count	SAR_SAMPLE_CTRL[7:4]	19.3.4.1 Averaging
Range detection	SAR_RANGE_THRES [31:0] SAR_RANGE_COND [31:30]	19.3.4.2 Range Detection

19.3.10.3 Set 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, the channel configurations should only be updated between scans (same as global configurations). However, if a channel is not enabled for the ongoing scan, then 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 19-12. Channel Configuration Registers

Configurations	Registers	Detailed Reference
Single-ended/differential	SAR_CHANx_CONFIG [8]	19.3.1.1 Single-ended and Differential Mode
Acquisition time selection	SAR_CHANx_CONFIG [13:12]	19.3.1.6 Acquisition Time
Resolution selection	SAR_CHANx_CONFIG [9]	19.3.1.5 Resolution
Average enable	SAR_CHANx_CONFIG [10]	19.3.4.1 Averaging
DSI output enable	SAR_CHANx_CONFIG [30]	DSI Output Enable

SUB_RESOLUTION (SAR_SAMPLE_CTRL[0]) can choose which alternate resolution will be used, either 8-bit or 10 bit. Resolution (SAR_CHANx_CONFIG [9]) can determine whether 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 19-13. Resolution

Average	SUB_RESOLUTION	Register Mode Resolution	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

Set 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 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.

19.3.10.4 Set Interrupt Masks

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

- End-of-scan interrupt
- Overflow interrupt
- Collision interrupt
- Injection end-of-conversion 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 re-quest 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 19.3.5 Interrupt for details.

19.3.10.5 Trigger

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

- Firmware trigger: SAR_START_CTRL [0]
- DSI trigger: dsi_trigger
- Continuous trigger: SAR_SAMPLE_CTRL [16]

See 19.3.6 Trigger for details.

19.3.10.6 Retrieve Data after Each Interrupt

Make sure 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 do 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 SAR_CHAN_WORK register is the mirror bit of the corresponding bit in SAR_CHAN_WORK_VALID register. Bit[29], bit [30],and bit[31] in SAR_CHAN_RESULT are the mirror bits of the corresponding bit in SAR_SATURATE_INTR, SAR_RANGE_INTR, and SAR_CHAN_RESULT_VALID registers. Note that the interrupt bits mirrored here are the raw (unmasked) interrupt bits. It helps firmware to check if the data is valid by just reading the data register.

If DSI output is enabled, it allows the SARSEQ result data to be processed by the UDBs and the channel number allows the possibility of applying different processing to data of different channels. See DSI Output Enable for detailed description.

19.3.10.7 Injection Conversions

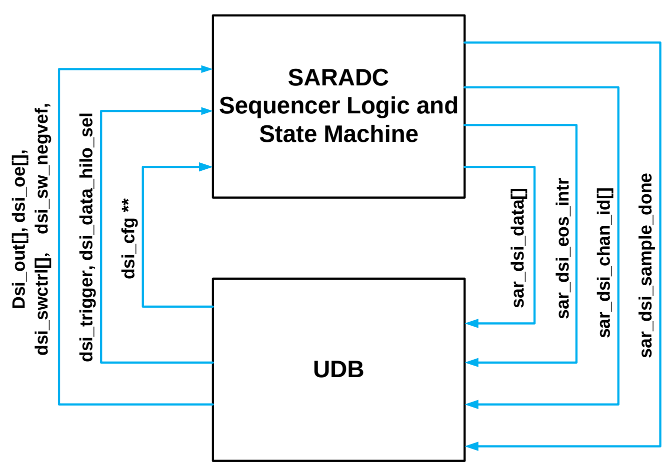
Injection channel can be triggered by setting the start bit INJ_START_EN (INJ_CHAN_CONFIG [31]). To prevent the collision of regular automatic scan, it is recommended to enable tailgating by setting INJ_CHAN_CONFIG [30]. When it is enabled, INJ_START_EN will enable the injection channel to be scanned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

See 19.3.4.4 Injection Channel for details.

19.3.11 DSI Mode

In DSI control mode, all of SAR ADC configuration can be done by DSI signals from UDB except the global configuration, such as interrupt masks, range detect settings, and triggers. The major difference between DSI mode and register mode is that the DSI mode allows hardware to dynamically control the ADC configuration. Figure 19-13 is a subset of the SAR ADC block diagram (Figure 19-1), which specifies the DSI input and output signals.

Figure 19-13. DSI Control Mode Block Diagram



The DSI control mode is selected by setting the DSI_MODE bit in the SAR_CTRL register. In this mode, the SARSEQ ignores all channel configurations in CHAN_EN, CHAN_CONFIG, and INJ_CHAN_CONFIG. Instead, it uses the configuration coming in via the DSI signal.

The following DSI signals are used.

Table 19-14. DSI Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging, if available) for one channel; the internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0]. If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>.
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_out	8	dsi_out[0]=1, P2.0 connected to ADC dsi_out[1]=1, P2.1 connected to ADC … dsi_out[7]=1, P2.7 connected to ADC **Note** MUX_SWITCH0 configuration determines whether the pin is connected to vplus or vminus.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, opamp0 output connected to ADC dsi_oe[3]=1, opamp1 output connected to ADC **Note** MUX_SWITCH0 configuration determines whether the signal is connected to vplus or vminus.
dsi_swctrl[0]	1	SARMUX analog switch control, connect vssa_kelvin to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temp_sens to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input
dsi_cfg_st_sel	2	Configuration control for DSI control mode: select 1 of 4 global sample times
dsi_cfg_average	1	Configuration control for DSI control mode: enable averaging
dsi_cfg_resolution	1	Configuration control for DSI control mode: 0=12-bit resolution 1=use globally configure resolution (8 or 10 bit)
dsi_cfg_differential	1	Configuration control for DSI control mode: 0= single-ended, 1=differential
dsi_trigger	1	Trigger to start SARSEQ scanning all enabled channels
dsi_data_hilo_sel	1	Selects between high and low byte output for sar_dsi_data[7:0]. This signal is fully asynchronous (affects sar_dsi_data without any clock involved).

19.3.11.1 Set SARMUX Analog Routing

In DSI mode, analog routing can be implemented by DSI signals and firmware. Firmware control is always available regardless of the register configuration and it is the same as in register mode. See 19.3.11.1 Set SARMUX Analog Routing for firmware control details.

DSI Control

DSI signals from UDB block are used to control SARMUX switches. In DSI control mode, the SARSEQ does not output any switch enables from the sequencer. Figure 19-3 shows that DSI can control every switch, except the DFT (design for test) switch. Thus, negative and positive input of SAR ADC can be connected to any switches in DSI mode.

Besides the DSI signals, appropriate hardware and firmware control bits in registers should be set. These registers and signals include SAR_MUX_SWITCH0 [n] = 1 and SAR_MUX_SWITCH_HW_CTRL[n] = 1. When V_{REF} is connected to the negative input, set SAR_CTRL [11:9] = 7 (firmware control field) and SAR_CTRL [13] = 1 (hardware control bit) except DSI signals.

DSI signals have control over the negative terminal of SAR ADC through dsi_swctrl[0] and dsi_sw_neg v_{REF} for single-ended mode. If NEG_SEL (SAR_CTRL[11:9]) is set, only NEG_SEL=7 is useful; the other value is ignored.

Table 19-15 shows the DSI signals.

Table 19-15. DSI Signal

Signal	Width	Description
dsi_out	8	dsi_out[0]=1, P2.0 connected to ADC dsi_out[1]=1, P2.1 connected to ADC … dsi_out[7]=1, P2.7 connected to ADC **Note** Whether the pin is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, sarbus0 output connected to ADC dsi_oe[3]=1, sarbus1 output connected to ADC **Note** Whether the signal is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_swctrl[0]	1	SARMUX analog switch control, connect V_{SSA} to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temperature sensor to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input

19.3.11.2 Set Global SARSEQ Configuration

Global configuration applies to both register mode and DSI control mode. See 19.3.10.2 Set Global SARSEQ Configuration for details.

19.3.11.3 Channel Configuration

For DSI control mode, only channel 0 is available. The channel 0 configuration can be done with DSI signals, as shown in Table 19-16. CHAN_EN and channel configurations in CHAN_CONFIG and INJ_CHAN_CONFIG are ignored.

The dsi_cfg_* signals can optionally be synchronized to the SAR clock domain (actually clk_hf) by setting DSI_SYNC_CONFIG. Bypassing synchronization may be required when running the SAR at a low frequency.

Table 19-16. Channel Configuration

Signal	Width	Config	Description
dsi_cfg_st_sel	2	Acquisition time	Configuration control for DSI control mode: select 1 of 4 global sample times
dsi_cfg_average	1	Average enable	Configuration control for DSI control mode: enable averaging
dsi_cfg_resolution	1	Resolution	Configuration control for DSI control mode: 0: 12-bit resolution 1: use globally configure resolution bit SUB_RESOLUTION (8 or 10 bit)
dsi_cfg_differential	1	Differential/single-ended	Configuration control for DSI control mode: 0: single-ended 1: differential

19.3.11.4 Interrupt

For an introduction to the SAR ADC interrupt, see Set Interrupt Masks on page 214. All interrupt masks work normally in register control mode. Not all interrupts are sent on DSI; SATURATE_INTR, RANGE_INTR, and EOS_INTR are sent via the DSI signal.

- Along with the data, SATURATE_INTR is output on dsi_chan_id[0]; SATURATE_INTR[0] is set in DSI control mode because only channel 0 is valid in DSI mode.
- Along with the data, RANGE_INTR is output on dsi_chan_id[1]; RANGE _INTR[0] is set in DSI control mode because only channel 0 is valid in DSI mode.
- Channel enables are ignored; this means only one conversion is done per trigger. An EOS_INTR is generated for each conversion.
- EOS_INTR is always sent via the DSI signal sar_dsi_eos_intr (a copy of dsi_data_valid).

Table 19-17 lists the interrupts that are sent via DSI signals.

Table 19-17. DSI Signal Interrupts

Signal	Width	Description
sar_dsi_chan_id	4	Register mode: Channel ID (ID of the channel that is currently being converted) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_eos_intr	1	End-of-scan interrupt to indicate that the SARSEQ has finished a scan of all enabled channels

19.3.11.5 Trigger

Typically, DSI control mode is used along with the DSI trigger. However, other trigger sources, such as firmware trigger and continuous trigger are also supported. The trigger configuration is the same as in the register control mode. See Trigger on page 210 for details.

For DSI trigger, the configuration settings (dsi_cfg_*) and switch settings should be stable no later than the cycle in which the dsi_trigger is sent. They should remain stable until the positive edge of the sar_dsi_sample_done.

19.3.11.6 Retrieve Data

The result data and channel number are sent out on sar_dsi_data. It is equivalent to dsi_out_en high in register control mode. See DSI Output Enable for details. After each conversion, the data is also written to both CHAN_WORK0 and CHAN_RESULT0 registers.

DSI Output Enable

If the DSI_OUT_EN bit (SAR_CHANx_CONFIG[31]) is set, the result data and channel number are also sent out on the DSI bus (sar_dsi_data, sar_dsi_chan_id), next to being stored in the regular result register. This allows for the SARSEQ result data to be processed by the UDBs and the channel number allows for the possibility to apply different processing to data of different channels.

The data sent out on the DSI bus is formatted in the same way it is stored in the result register. However, by default only the 12 LSBs are sent out; it is not recommended to use left alignment unless more than 12 bits are required. To get the upper eight LSBs, the dsi_data_hilo_sel input needs to be set to ‘1’. To get the full 16-bit data from result register, first set dsi_data_hilo_sel = 0 to get the lower 12-bit data and then set dsi_data_hilo_sel = 1 to get the upper 8-bit data. Additional data process is needed to deal with the data overlap.

The channel number (sar_dsi_chan_id) will be sent out earlier, after the SAR ADC has completed sampling that channel. The channel number by itself can trigger the UDBs to drive some GPIO pins, which in turn can power up (or down) some off-chip device. This drives an analog input pin that will be scanned by one of the subsequent channels in the same scan (a long sample time is useful here).

Note that the data is sent out one cycle after the conversion is completed. Channel numbers, data, and their respective valid signals are maintained for two system clock cycles on the DSI bus.

Table 19-18. DSI Output Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging if there is) for one channel. The internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0] If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_data_hilo_sel	1	Selects between high and low byte output for sar_dsi_data[7:0]. This signal is fully asynchronous (affects sar_dsi_data without any clock involved)

19.3.12 Analog Routing Configuration Example

Table 19-19 shows some examples of pin and signal selection for sequencer control, firmware control, and DSI control.

Table 19-19. Analog Routing Configuration Example

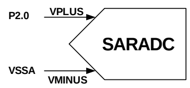
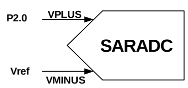
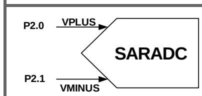
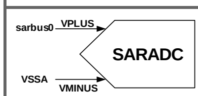
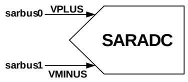
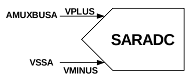
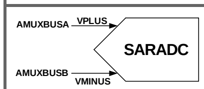
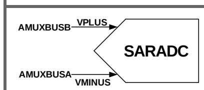
	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_out [0] =1 dsi_swctrl[0]=1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 7 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0]=1 HW_CTRL_NEGVREF =1 (CTRL[13])	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] =0 NEG_SEL = 7 (CTRL [11:9]) HW_CTRL_NEGVREF =0 (CTRL[13])	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 dsi_out [0] =1 dsi_sw_negvref =1 HW_CTRL_NEGVREF =1 (CTRL[13])
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 or PIN_ADDR = 1 (CHANx_CONFIG[2:0]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[1] = 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[1] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_out [0] =1 dsi_out [1] =1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH0 [9] = 1 MUX_SWITCH_HW_CTRL[1]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 1 (CHANx_CONFIG[6:4]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] =1 MUX_SWITCH_HW_CTRL[16] =1 **Note** Connecting sarbus1 to VPLUS is not supported for Port/Pin control	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [2] =1 dsi_swctrl[0]=1 MUX_SWITCH0 [16] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH_HW_CTRL[22] =1

Table 19-19. Analog Routing Configuration Example<Italic> (continued)

	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 1 **(**CHANx_CONFIG[6:4]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[23] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [2] = 1 dsi_oe [3] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** **NEG_SEL = 0** (CTRL [11:9]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[16]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [0] = 1 dsi_swctrl[0]=1 MUX_SWITCH0[18] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[19]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1
	Not supported. The differential pair is fixed for Port/Pin control	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =0 MUX_SWITCH_HW_CTRL[19] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =1 MUX_SWITCH_HW_CTRL[19] = 1

19.3.13 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 1.024 V.

A pin or signal can be routed to the SAR ADC in three ways. Table 19-20 lists the methods to route temperature sensors to SAR ADC. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit disables temperature sensor by cutting its bias current.

Table 19-20. Route Temperature to SAR ADC

Control Methods	Setup
Sequencer	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4]) PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0]) SWITCH_DISABLE = 0 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
Firmware	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) SWITCH_DISABLE = 1 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 0 SAR_MUX_SWITCH_HW_CTRL[17]= 0 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
DSI	SWITCH_DISABLE = 1 (SAR_CTRL[30]) VREF_SEL = 0 (SAR_CTRL[6:4]) Set DSI Signals: dsi_cfg_differential=1 dsi_swctrl[1]=1 dsi_swctrl[0]=1 SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]

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

19.4 Registers

Name	Offset	Qty.	Width	Description
SAR_CTRL	0x0000	1	32	Global configuration register Analog control register
SAR_SAMPLE_CTRL	0x0004	1	32	Global configuration register Sample control register
SAR_SAMPLE_TIME01	0x0010	1	32	Global configuration register Sample time specification ST0 and ST1
SAR_SAMPLE_TIME23	0x0014	1	32	Global configuration register Sample time specification ST2 and ST3
SAR_RANGE_THRES	0x0018	1	32	Global range detect threshold register
SAR_RANGE_COND	0x001C	1	32	Global range detect mode register
SAR_CHAN_EN	0x0020	1	32	Enable bits for the channels
SAR_START_CTRL	0x0024	1	32	Start control register (firmware trigger)
SAR_CHAN_CONFIG	0x0080	8	32	Channel configuration register
SAR_CHAN_WORK	0x0100	8	32	Channel working data register
SAR_CHAN_RESULT	0x0180	8	32	Channel result data register
SAR_CHAN_WORK_VALID	0x0200	1	32	Channel working data register valid bits
SAR_CHAN_RESULT_VALID	0x0204	1	32	Channel result data register valid bits
SAR_STATUS	0x0208	1	32	Current status of internal SAR registers (for debug)
SAR_AVG_STAT	0x020C	1	32	Current averaging status (for debug)
SAR_INTR	0x0210	1	32	Interrupt request register
SAR_INTR_SET	0x0214	1	32	Interrupt set request register
SAR_INTR_MASK	0x0218	1	32	Interrupt mask register
SAR_INTR_MASKED	0x021C	1	32	Interrupt masked request register: If the value is not zero, then the SAR interrupt signal to the NVIC is high. When read, this register reflects a bit-wise AND between the interrupt request and mask registers
SAR_SATURATE_INTR	0x0220	1	32	Saturate interrupt request register
SAR_SATURATE_INTR_SET	0x0224	1	32	Saturate interrupt set request register
SAR_SATURATE_INTR_MASK	0x0228	1	32	Saturate interrupt mask register
SAR_SATURATE_INTR_MASKED	0x022C	1	32	Saturate interrupt masked request register
SAR_RANGE_INTR	0x0230	1	32	Range detect interrupt request register
SAR_RANGE_INTR_SET	0x0234	1	32	Range detect interrupt set request register
SAR_RANGE_INTR_MASK	0x0238	1	32	Range detect interrupt mask register
SAR_RANGE_INTR_MASKED	0x023C	1	32	Range interrupt masked request register
SASR_INTR_CAUSE	0x0240	1	32	Interrupt cause register
SAR_INJ_CHAN_CONFIG	0x0280	1	32	Injection channel configuration register
SAR_INJ_RESULT	0x0290	1	32	Injection channel result register
SAR_MUX_SWITCH0	0x0300	1	32	SARMUX firmware switch controls
SAR_MUX_SWITCH_CLEAR0	0x0304	1	32	SARMUX firmware switch control clear
SAR_MUX_SWITCH_HW_CTRL	0x0340	1	32	SARMUX switch hardware control
SAR_MUX_SWITCH_STATUS	0x0348	1	32	SARMUX switch status
SAR_PUMP_CTRL	0x0380	1	32	Switch pump control

20. Low-Power Comparator

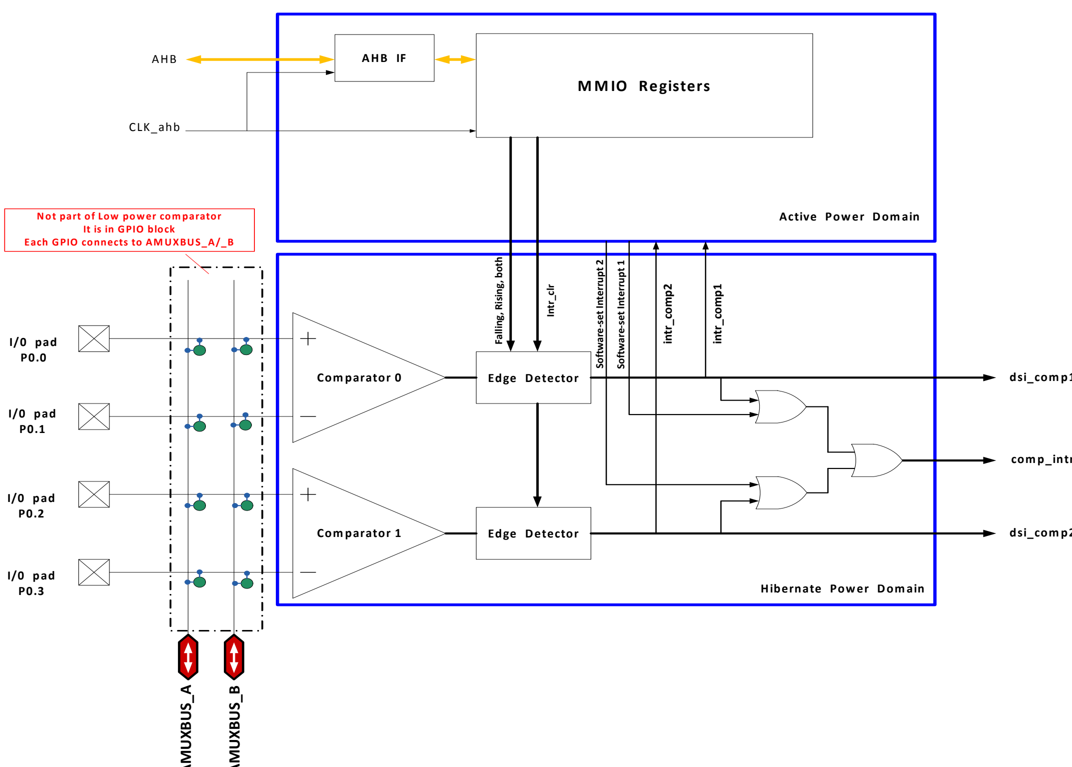
PSoC® 4 devices have two low-power comparators. These comparators are placed in the hibernate power domain, allowing fast analog signal comparison in all system power modes except the Stop mode. The positive and negative inputs can be connected to the dedicated GPIO pins or to AMUXBUS-A/AMUXBUS-B. The comparator output can be read by the CPU, used as an interrupt or wakeup source, or fed to the DSI.

20.1 Features

PSoC 4 comparators have the following features:

- Selectable input
- Programmable power and speed
- Low-power mode support
- Optional 10-mV input hysteresis
- Low-input offset voltage (<4 mV after trim)
- Sleep/hibernate wakeup with comparator output

20.2 Block Diagram



20.3 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 hibernate, comparator clock, and offset trim.

20.3.1 Input Configuration

Inputs to the comparators can be as follows:

- Two voltages on external pins
- A voltage from an external pin and an internally generated signal, both can be either on positive or negative input of the comparators. In this case, the internal signal is brought to the comparator using the AMUXBUS
- Two voltages from internally generated signals through AMUXBUS-A/AMUXBUS-B

As the block diagram shows, P0.0, P0.1, P0.2, and P0.3 are directly connected to the input of the low-power comparator. The use of the comparator connection to the AMUXBUSes consumes the input pins. See the I/O System chapter on page 51 for more details on connecting the GPIO to AMUXBUS A/B.

20.3.2 Power Mode and Speed Configuration

The two comparators can operate in three power modes: fast, slow, and ultra low-power. The power for Comparator 0 is configured in MODE1 bits [1:0] in the LPCOMP_CONFIG register. The power for Comparator 1 is configured in MODE2 bits [9:8] in the same register. Note that the output of the comparator may glitch when the power mode is changed.

Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ultra-low-power mode. Specifications for power consumption and response time are provided in the datasheet.

20.3.3 Output and Interrupt Configuration

The current output value of each comparator is stored in a separate OUT bit in the LPCOMP_CONFIG register. Comparator 0 output value is stored in LPCOMP_CONFIG [6], and comparator 1 output is stored in LPCOMP_CONFIG [14]. The comparator output is connected to an edge detector block. This block determines the edge (disable/rising/falling/both) that triggers the IRQ by configuring the INTTYPE bits in the LPCOMP_CONFIG register. Note that the direct result of the comparator is not available as a hardware signal. The output is normally connected to an interrupt. During compare events, the compare will output a pulse, which is cleared by a software interrupt. If the interrupt is not cleared, the next compare event cannot be detected.

Each comparator can generate an interrupt request. However, the LPCOMP block only has a single common interrupt to CPU NVIC, which is the logic OR of those two interrupt requests. The LPCOMP interrupt (comp1_intr/comp2_intr) is synchronous with clk_ahb. The LPCOMP DSI output dsi_comp1/dsi_comp2 is asynchronous. Clearing dsi_comp1/dis_comp2, and comp1_intr/comp2_intr are all synchronous. In active and sleep modes, dsi_comp1/2 can be routed to GPIO or other blocks through DSI routing in UDB with or without synchronization; there is an optional synchronizer on UDB DSI output. Note that in low-power modes (deep-sleep and hibernate), this routing is unavailable because the UDB is powered off. For example, when dsi_comp1/dis_comp2 is used as the kill signal of the PWM block, whether it is asynchronous or synchronous can be configured in a register, which will be specified in the UDB block. If the dsi_comp1/dsi_comp2 is routed to UDB for further processing, the timing depends on the user's algorithm and synchronizer choice. The LPCOMP_INTR register bits [1:0] show the interrupt request of comparator 0 and comparator 1. LPCOMP_INTR_SET register bits [1:0] can be used to assert an interrupt for software debugging.

In low-power mode, the wakeup interrupt controller (WIC) can be activated by a comparator switch event, which then wakes up the CPU. Thus, the LPCOMP still has the capability to monitor the specified signal in low-power mode.

20.3.4 Hysteresis

For applications that compare signals close to each other, hysteresis helps to avoid excessive toggling of the comparator output when the signals are noisy.

The 10-mV hysteresis level is enabled by setting the hysteresis enable (HYST) bit in the LPCOMP_CONFIG register, LPCOMP_CONFIG [2] for comparator 0 and LPCOMP_CONFIG [10] for comparator 1.

20.3.5 Wakeup from Low-Power Modes

The comparator can run in low-power mode, including sleep, deep-sleep, and hibernate modes. The comparator output interrupt can wake up the device from sleep, deep-sleep, and hibernate modes. No special setting is needed. In deep-sleep or hibernate power mode, the edge of both Comparator 0 and Comparator 1 output will generate an interrupt. This behavior is unrelated to the settings of INTTYPE bit in LPCOMP_CONFIG register.

20.3.6 Comparator Clock

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

20.3.7 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 operating range of 0.1 V to V_{DD}–0.1 V. For normal operation, further adjustment of trim values is not recommended.

If tighter trim is required at a specific input common mode voltage, trim the comparator at that voltage. The comparator offset trim is performed in the LPCOMP_TRIM1/2/3/4 registers. LPCOMP_TRIM1 and LPCOMP_TRIM2 are for comparator 0. LPCOMP_TRIM3 and LPCOMP_TRIM4 are for comparator 1. The bit fields that change the trim values are TRIMA in LPCOMP_TRIM1 and LPCOMP_TRIM3, and TRIMB in LPCOMP_TRIM2 and LPCOMP_TRIM4. If shorting of the inputs is required for offset calibration, the calibration enable field (cal_en) in the LPCOMP_DFT register helps to achieve it.

The trim procedure is as follows:

1. Short inputs using the calibration enable field (cal_en) in the LPCOMP_DFT register.
2. Set the two inputs 'inn' and 'inp' to the required value.
3. Change the trimA register settings:
   a. Depending on the polarity of the offset measured, set or clear trimA[4] bit.
   b. Increase the value of trimA[3:0] until offset measured is less than 1 mV.
4. If the polarity of the offset measured has changed, but the offset is still greater than 1 mV, use trimB[3:0] to fine tune the offset value. This is valid only for the slow mode of comparator operation.
5. If trimA[3:0] is 0Fh and the measured offset is still greater than 1 mV, set or clear trimB[3], depending on the polarity of offset. Increase the value of trimB[2:0] until the offset measured is less than 1 mV.

20.4 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_DFT	LPCOMP DFT 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

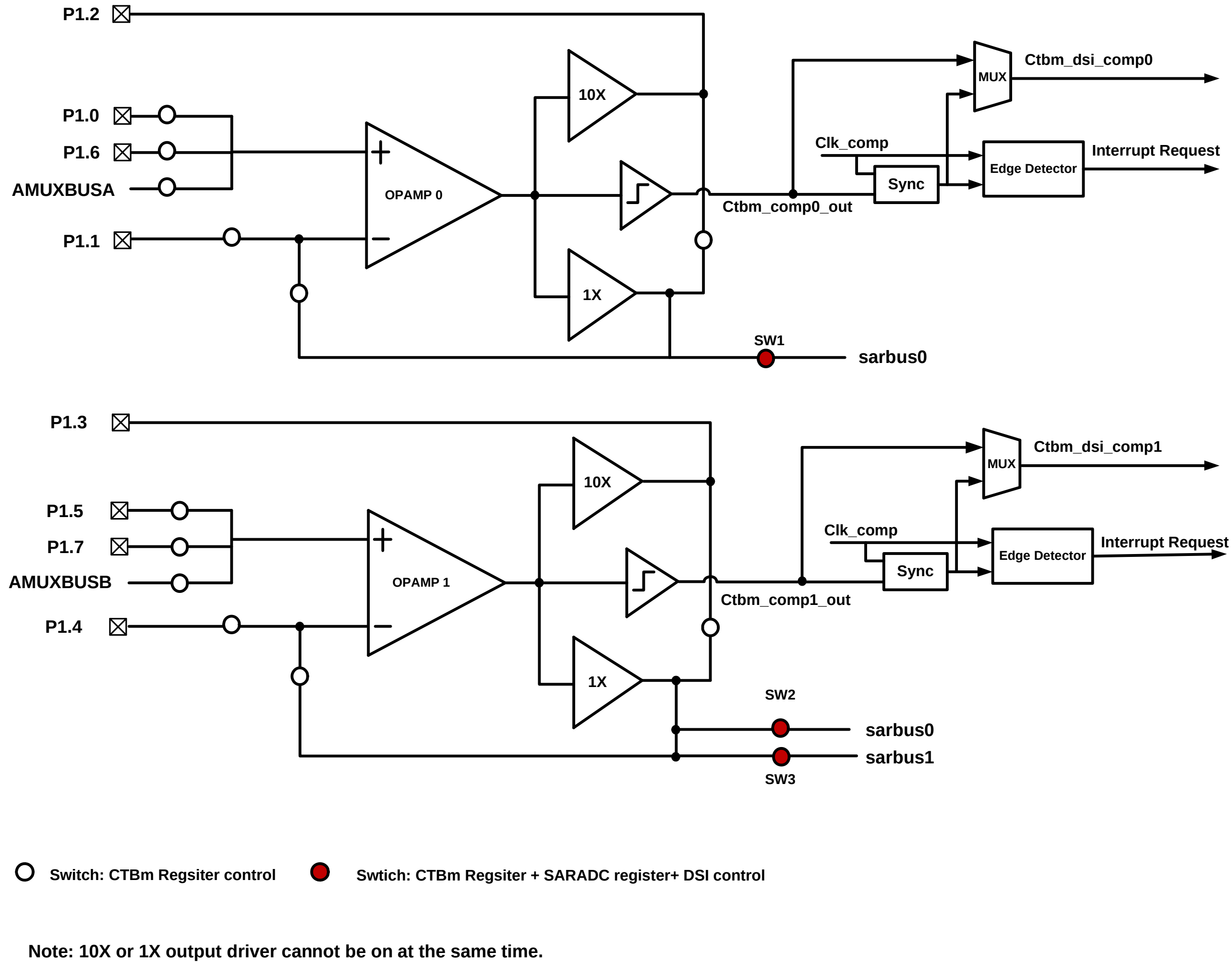
21. Continuous Time Block mini (CTBm)

The Continuous Time Block mini (CTBm) provides the continuous time functionality. It includes a switch matrix, two identical operational amplifiers (opamps), which are also configurable as two comparators, one charge pump inside each opamp, and a digital interface. Compared with the Continuous Time Block (CTB) found in other devices of the PSoC 4 family, the CTBm has no resistors and has fewer switches.

21.1 Features

- Highly configurable opamp: power and speed, output driver, compensation
- Each opamp can be configured as a follower with internal switch
- Each opamp can be configured as comparator with 10-mV hysteresis
- 10-mA output current drive capability
- 4-MHz gain bandwidth for 20-pF load
- Offset trimmed to less than 1 mV
- Rail-to-rail within 0.2 V of V_{SS} or V_{DDA} for 1-mA load
- Rail-to-rail within 0.5 V of V_{SS} or V_{DDA} for 10-mA load
- Slew rate 4 V/µs for 50-pF load

21.2 Block Diagram



21.3 How It Works

As the block diagram shows, CTBm is built up of two identical opamps and a switch routing matrix. Each opamp has one input and three output stages, which can be selected one at a time. The output stage consists of three drivers, which 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 this block by setting CTBm_CTRL [31]. 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 bit CTBm_OA_RES0_CTRL [11] for opamp0, and CTBm_OA_RES1_CTRL [11] for opamp1.

Then, follow these steps:

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
6. Configure comparator mode, if required

21.3.1 Power Mode Configuration

CTBm reduces power by reducing the reference currents coming into the opamp. The opamp can operate in three power modes – low, medium, and high. Power modes are configured using the PWR_MODE bits (CTBm_OA_RESx_CTRL[1:0]). The slew rate and gain bandwidth are maximum in high-power mode and minimum in low-power mode. Note that power mode configuration also impacts the maximum output drive capability (I_{OUT}) in 1X mode. See Table 21-1 for details. See the device datasheet for gain bandwidth, slew rate, and I_{OUT} specifications in various power modes.

21.3.2 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 on 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 21-1.

Table 21-1. 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 CTB_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 CTBm_OAx_SW [21] to '1'. However, Cypress does not guarantee performance in this case.

21.3.3 Compensation

Each opamp also has a programmable compensation capacitor block, which allows optimizing the opamp performance based on output load. The compensation of each opamp is controlled by the respective CTBm_OAx_COMP_TRIM register. Note that all the GBW, slew rate specifications in device datasheet are applied for all compensation trim.

Table 21-2. Opamp 0 or Opamp 1 Compensation

CTBm_OAx_COMP_TRIM[1:0]	Description
00	Minimum compensation, high speed, and low stability
01	Medium compensation, balanced speed and stability
11	Maximum compensation, low speed, and high stability

21.3.4 Switch Control

The CTBm has many switches to configure the opamp input and output. Most of them are controlled by configuring CTBm registers (CTBm_OA0_SW, CTBm_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, CTBm registers, and DSI signals.

Switches can be closed by setting the corresponding bit in register CTBm_OAx_SW; clearing them will cause the corresponding switches to open. Writing '1' to CTBm_OAx_SW_CLEAR can clear the corresponding bit in CTBm_OAx_SW.

21.3.4.1 Input Configuration

Positive and negative input 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, 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 Make sure only one switch is closed for both positive and negative input; otherwise, different input source may be short together.

- Positive input

  Both opamp0 and opamp1 have three positive input options through analog switches: two external pins and one AMUXBUS line. See Table 21-3 for details.

Table 21-3. Positive Input

	Positive Input	Switch Control Bit	Description
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

- Negative input

  Both opamp0 and opamp1 have two negative input options through analog switches: one external pin or output feedback, which is controlled by the CTBm_OAx_SW register. Table 21-4 shows the detailed control bits.

Table 21-4. Negative Input

	Negative Input	Switch Control Bit	Description
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

21.3.4.2 Output Configuration

The opamp 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). Opamp0 output can be connected to sarbus0 and opamp1 can be connected to sarbus0 or sarbus1, which is intended to connect opamp output to SAR ADC. These three switches are controlled by the CTBm register, SAR ADC register, and DSI signals together; the other switches can be controlled only by CTBm register.

The following truth tables show the control logic of the three switches. PORT_ADDR, PIN_ADDR, and DIFFERENTIAL_EN are from SAR_CHANx_CONFIG [6:4], SAR_CHANx_CONFIG [2:0], and SAR_CHANx_CONFIG [2:0], respectively. Either PORT_ADDR =0 or PIN_ADDR = 0 will set SW[n]=0. CTB_SW_HW_CTRL bit [2] or [3] should be set when using the SAR register or a DSI signal to control switches. CTB_OAx_SW[18]/[19] can mask the other control bits – if CTB_OAx_SW[18]/[19] = 0, SW[n] = 0.

Register CTBm_SW_STATUS [30:28] gives the current switch status of SW1/2/3.

Table 21-5. Truth Table of SW1 Control Logic

PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[2]	dsi_out[2]	CTB_OA0_SW[18]	SW1
X	X	X	X	0	0
X	0	1	0	1	0
0	X	1	0	1	0
X	X	X	1	1	1
X	X	0	X	1	1
1	2	X	X	1	1

Table 21-6. Truth Table of SW2 Control Logic

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[3]	dsi_out[3]	CTB_OA0_SW[18]	SW2
X	X	X	X	X	0	0
X	X	0	1	0	1	0
X	0	X	1	0	1	0
1	X	X	X	0	1	0
X	X	X	0	X	1	1
X	X	X	X	X	1	1
0	1	3	X	X	1	1

Table 21-7. Truth Table of SW3 control logic

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTB_SW_HW_CTRL[3]	dsi_out[3]	CTB_OA0_SW[18]	SW3
X	X	X	X	X	0	0
X	X	0	1	0	1	0
X	0	X	1	0	1	0
0	X	X	X	0	1	0
X	X	X	0	X	1	1
X	X	X	X	X	1	1
1	1	2	X	X	1	1

21.3.4.3 Comparator Mode

Each opamp can be configured as a comparator by setting the respective CTBm_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. When configured as comparators, they have the following features:

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

21.3.4.4 Comparator Configuration

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

The comparator output is routed to the DSI with optional synchronization. The synchronization with comparator clock (system AHB clock) can be configured in CTBm_OA_RESx_CTRL[6].

The output state of comparator0 and comparator1 are stored in CTBm_COMP_STAT[0] and CTBm_COMP_STAT[16], respectively.

21.3.4.5 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 CTBm_OA_RESx_CTRL[9:8] bits.

Each comparator has a separate IRQ. CTBm_INTR [0] is for comparator0 IRQ, CTBm_INTR [1] is for comparator1 IRQ.

Each of the interrupts has an interrupt mask bit in the CTBm_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 logic AND of the interrupt flags in CTBm_INTR registers and the corresponding interrupt masks in CTBm_INTR_MASK register is 1.

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

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

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

21.4 Register Summary

Table 21-8. Register Summary

Offset	Width	Name	Description
0x0000	32	CTBm_CTRL	Global CTBm block enable
0x0004	32	CTBm_OA_RES0_CTRL	Opamp0 control register
0x0008	32	CTBm_OA_RES1_CTRL	Opamp1 control register
0x000C	32	CTBm_COMP_STAT	Comparator status
0x0020	32	CTBm_INTR	Interrupt request register
0x0024	32	CTBm_INTR_SET	Interrupt request set register
0x0028	32	CTBm_INTR_MASK	Interrupt request mask
0x002C	32	CTBm_INTR_MASKED	Interrupt request masked
0x0030	32	CTBm_DFT_CTRL	Analog DFT controls
0x0080	32	CTBm_OA0_SW	Opamp0 switch control
0x0084	32	CTBm_OA0_SW_CLEAR	Opamp0 switch control clear
0x0088	32	CTBm_OA1_SW	Opamp1 switch control
0x008C	32	CTBm_OA1_SW_CLEAR	Opamp1 switch control clear
0x00C0	32	CTBm_SW_HW_CTRL	CTBm hardware control enable
0x00C4	32	CTBm_SW_STATUS	CTBm bus switch control status
0x0F00	32	CTBm_OA0_OFFSET_TRIM	Opamp0 trim control
0x0F04	32	CTBm_OA0_SLOPE_OFFSET_TRIM	Opamp0 trim control
0x0F08	32	CTBm_OA0_COMP_TRIM	Opamp0 trim control
0x0F0C	32	CTBm_OA1_OFFSET_TRIM	Opamp1 trim control
0x0F10	32	CTBm_OA1_SLOPE_OFFSET_TRIM	Opamp1 trim control
0x0F14	32	CTBm_OA1_COMP_TRIM	Opamp1 trim control

22. LCD Direct Drive

The PSoC® 4 Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PSoC device to directly drive STN and TN segment LCDs.

22.1 Features

The PSoC 4 LCD segment drive function has these features:

- Supports up to four commons (mux ration 1:4)
- Supports Type A (standard) and Type B (low-power) drive waveforms
- Any GPIO can be configured as a common or segment
- Supports three drive methods:
  - Digital correlation
  - PWM at 1/2nd bias
  - PWM at 1/3rd bias
- Ability to drive 3-V displays from 1.8 V V_{DD} in Digital Correlation mode
- Operates in active, sleep, and deep-sleep modes
- Digital contrast control

22.2 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_{LO}**: The voltage that the LCD driver can realize on segments that are intended to be off.
- **V_{HI}**: The voltage that the LCD driver can realize on segments that are intended to be on.
- **Discrimination Ratio (D)**: The ratio of V_{HI} and V_{LO} 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/Mth duty when it drives 'M' number of COM electrodes. Each COM electrode is effectively driven 1/Mth of the time. PSoC 4 supports 1/2nd, 1/3rd, and 1/4th duties.
- **Bias**: A driver is said to use 1/Bth 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 1/2nd and 1/3rd 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 different types of drive waveforms in all drive modes. These are:

- **Type-A Waveform**: In this type of waveform, the driver structures a frame into M sub-frames. 'M' is the number of COM electrodes. Each COM is addressed only once during a frame. For example, COM[i] is addressed in sub-frame i.
- **Type-B Waveform**: The driver structures a frame into 2M sub-frames. The two sub-frames are inverses of each other. Each COM is addressed twice during a frame. For example, COM[i] is addressed in sub-frames i and M+i. Type-B waveforms are slightly more power efficient because it contains fewer transitions.

22.2.1 Drive Modes

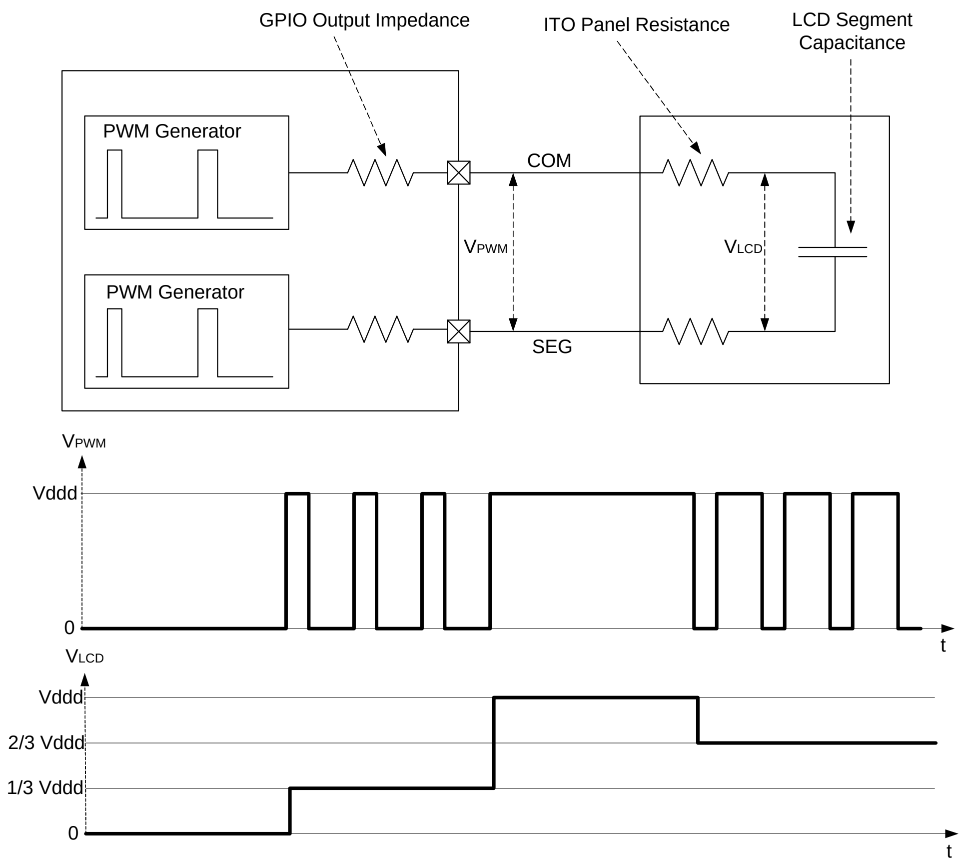
PSoC 4 supports the following drive modes.

- PWM Drive at 1/2nd bias
- PWM Drive at 1/3rd bias
- Digital correlation

22.2.1.1 PWM Drive

In PWM drive mode, multi-voltage drive signals are generated using a PWM output signal together with the intrinsic resistance and capacitance of the LCD. Figure 22-1 illustrates this.

Figure 22-1. PWM Drive (at 1/3rd Bias)

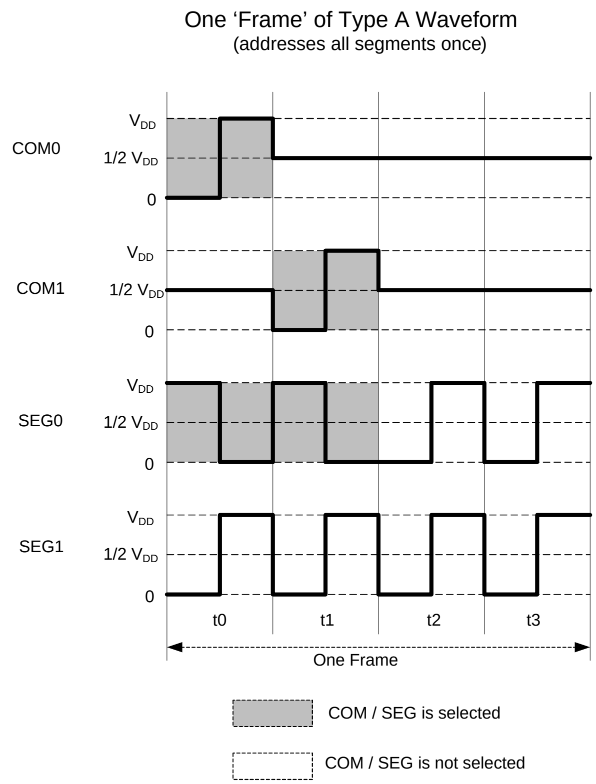


The output waveform of the drive electronics is a PWM waveform. With the Indium Tin Oxide (ITO) panel resistance and the segment capacitance to filter the PWM, the voltage across the LCD segment is an analog voltage, as shown in Figure 22-1. This figure illustrates the generation of a 1/3rd bias waveform (four commons and voltage steps of $V_{DD}/3$).

The PWM frequency is derived from either ILO (32 kHz) or IMO. The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 22-2 and Figure 22-3 illustrate the generated analog waveforms for COM and SEG electrodes for 1/2nd bias and 1/4th duty. Only COM0/COM1 and SEG0/SEG1 are drawn. Similarly, Figure 22-4 and Figure 22-5 illustrate the analog waveforms for COM and SEG electrodes for 1/3rd bias and 1/4th duty.

Figure 22-2. PWM1/2nd Type-A Waveform Example



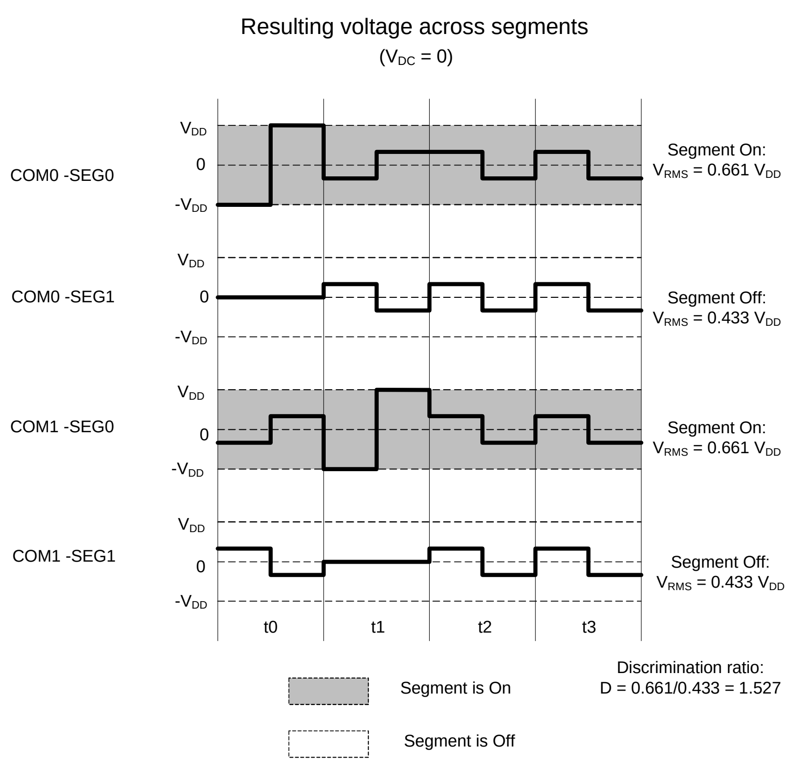


Figure 22-3. PWM1/2nd Type-B Waveform Example

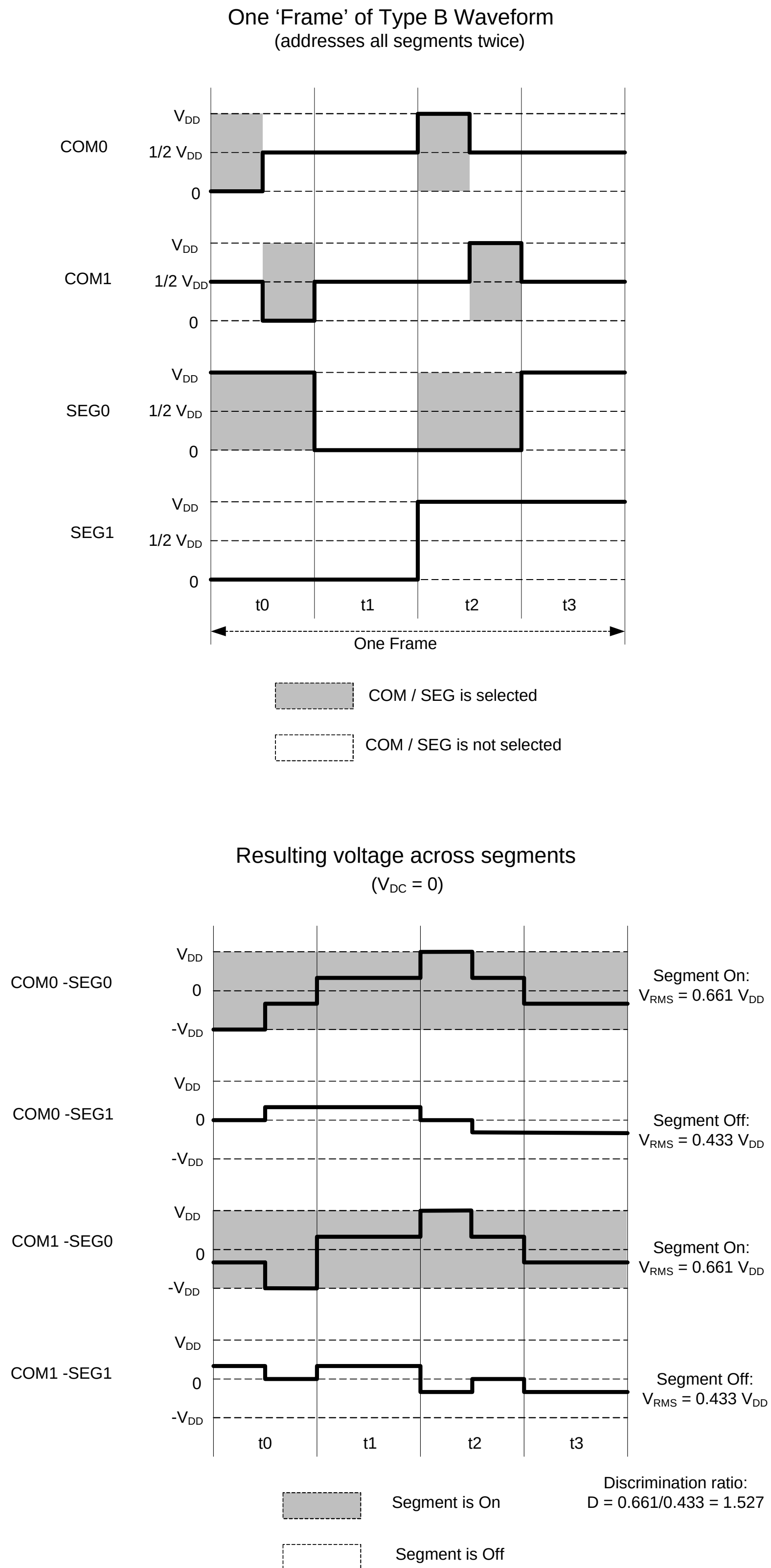
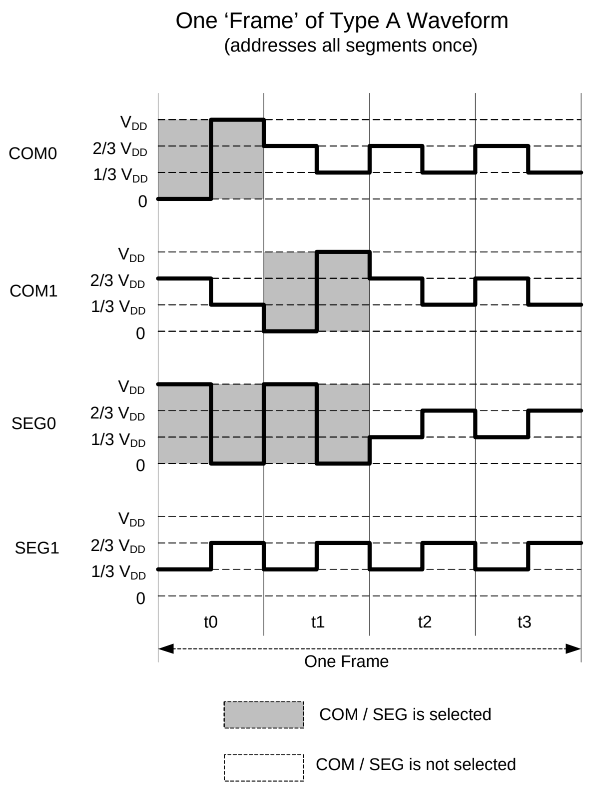


Figure 22-4. PWM1/3rd Type-A Waveform Example



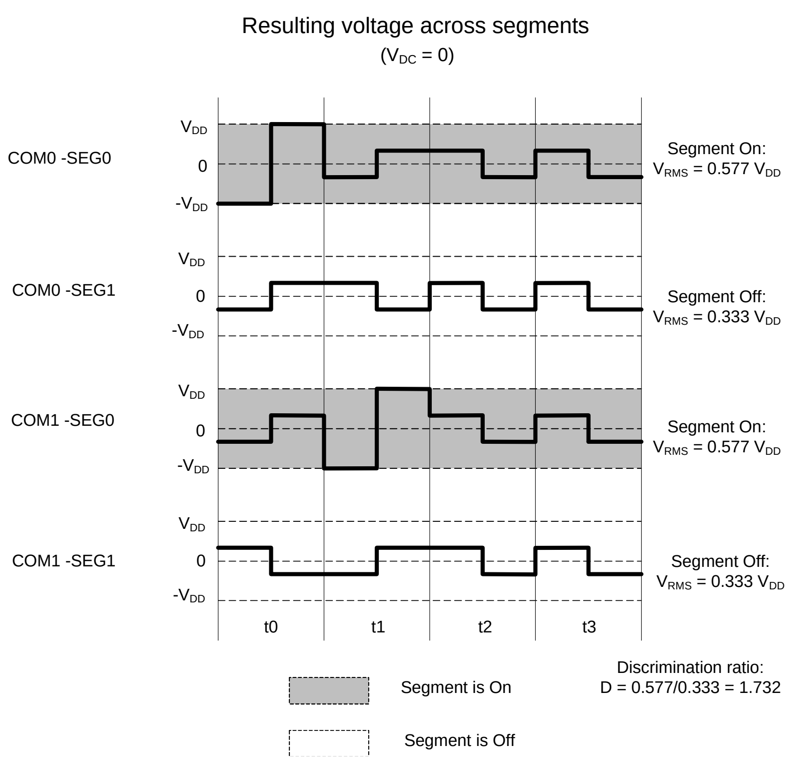
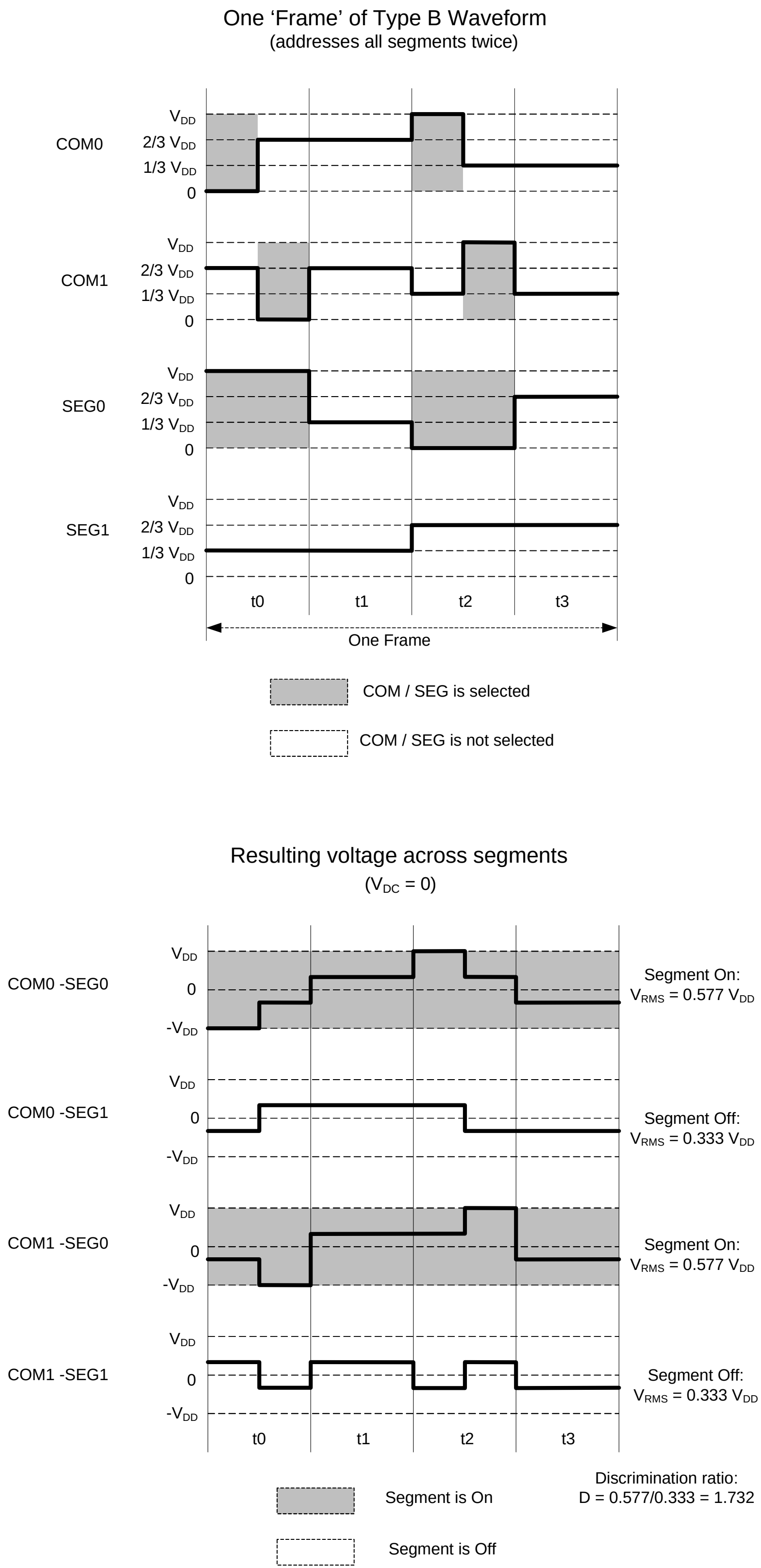


Figure 22-5. PWM1/3rd Type-B Waveform Example



The effective RMS voltage for ON and OFF segments can be calculated easily using these equations:

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

Equation 22-1

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

Equation 22-2

Where B is the bias and M is the duty (number of COMs).

For example, if the number of COMs is 4, the resulting discrimination ratios (D) for 1/2nd and 1/3rd biases are 1.528 and 1.732, respectively. 1/3rd bias offers better discrimination ratio in 2 and 3 COM drives also. Therefore, 1/3rd bias offers better contrast than 1/2nd bias and is recommended for most applications.

When the low-speed operation of LCD is used, the PWM signal is derived from the 32-kHz ILO. To drive a low-capacitance display with acceptable ripple and rise/fall times using a 32-kHz PWM, additional external series resistances of 100k-1MΩ should be used. External resistors are not required for PWM frequencies greater than ~1 MHz. The ideal PWM frequency depends on the capacitance of the display and the internal ITO resistance of the ITO routing traces.

The 1/2nd 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 22-2 and Figure 22-3.

22.2.1.2 Digital Correlation

The digital correlation mode, instead of generating bias voltages between the rails, takes advantage of the characteristic of LCDs that the contrast of LCD segments is determined by the RMS voltage across the segments. In this approach, the correlation coefficient between any given pair of COM and SEG signals determines whether the corresponding LCD segment is on or off. Thus, by doubling the base drive frequency of the COM signals in their inactive sub-frame intervals, the phase relationship of the COM and SEG drive signals can be varied to turn segments on and off. This is different from varying the DC levels of the signals as in the PWM drive approach. Figure 22-8 and Figure 22-9 are example waveforms that illustrate the principles of operation.

Figure 22-6. Digital Correlation Type-A Waveform

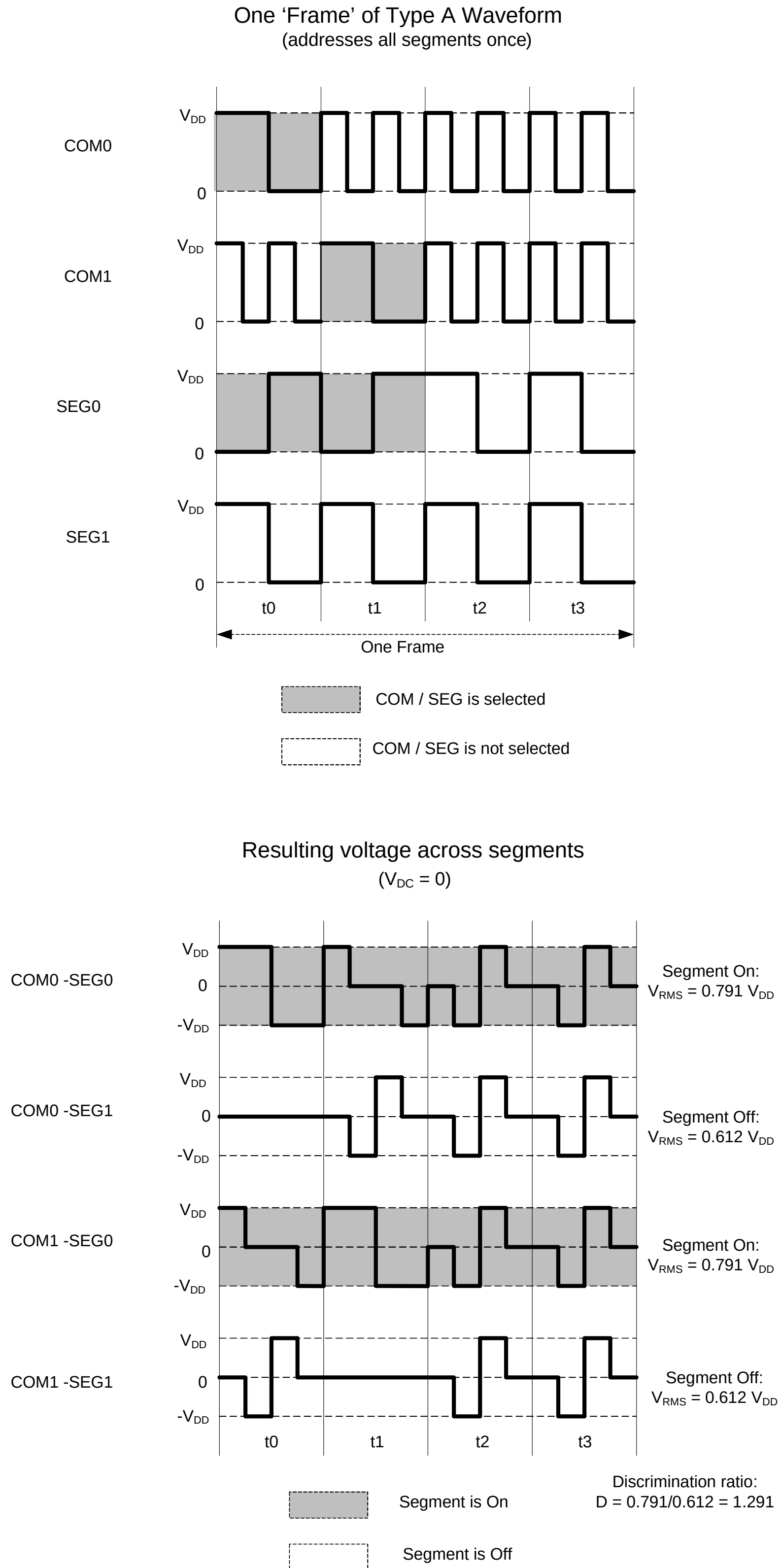
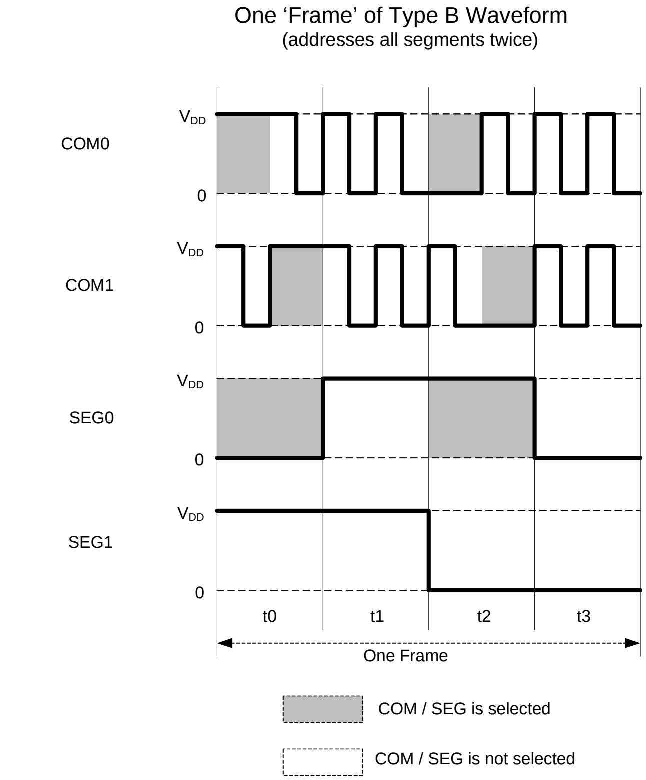
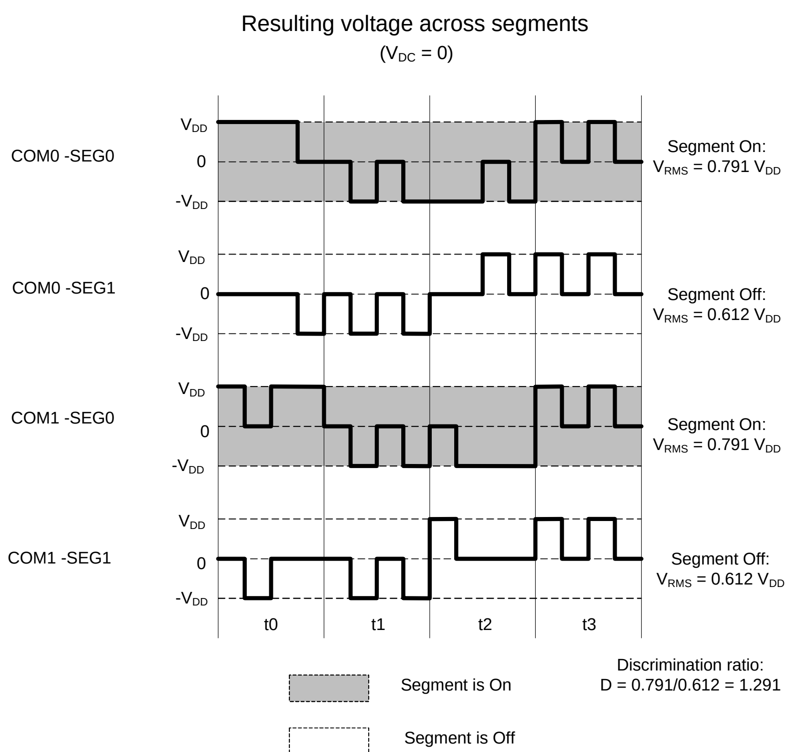


Figure 22-7. Digital Correlation Type-B Waveform





The RMS voltage applied to on and off segments can be calculated as follows:

$$V_{RMS(OFF)} = \sqrt{\frac{(M-1)}{2M}} x (V_{DD})$$

$$V_{RMS(ON)} = \sqrt{\frac{2+(M-1)}{2M}} x (V_{DD})$$

Where B is the bias and M is the duty (number of COMs). This leads to a discrimination ratio (D) of 1.291 for four COMs.

Digital correlation mode also has the ability to drive 3-V displays from 1.8 V V_{DD}.

22.2.2 Recommended Usage of Drive Modes

The PWM drive mode has higher discrimination ratios compared to the digital correlation mode, as explained in 22.2.1.1 PWM Drive and 22.2.1.2 Digital Correlation. Therefore, the contrast in digital correlation method is lower than PWM method but digital correlation has lower power consumption because its waveforms toggle at low frequencies.

The digital correlation mode creates reduced, but acceptable contrast on TN displays, but no noticeable difference in contrast or viewing angle on higher contrast STN displays.

Because each mode has strengths and weaknesses, recommended usage is as follows.

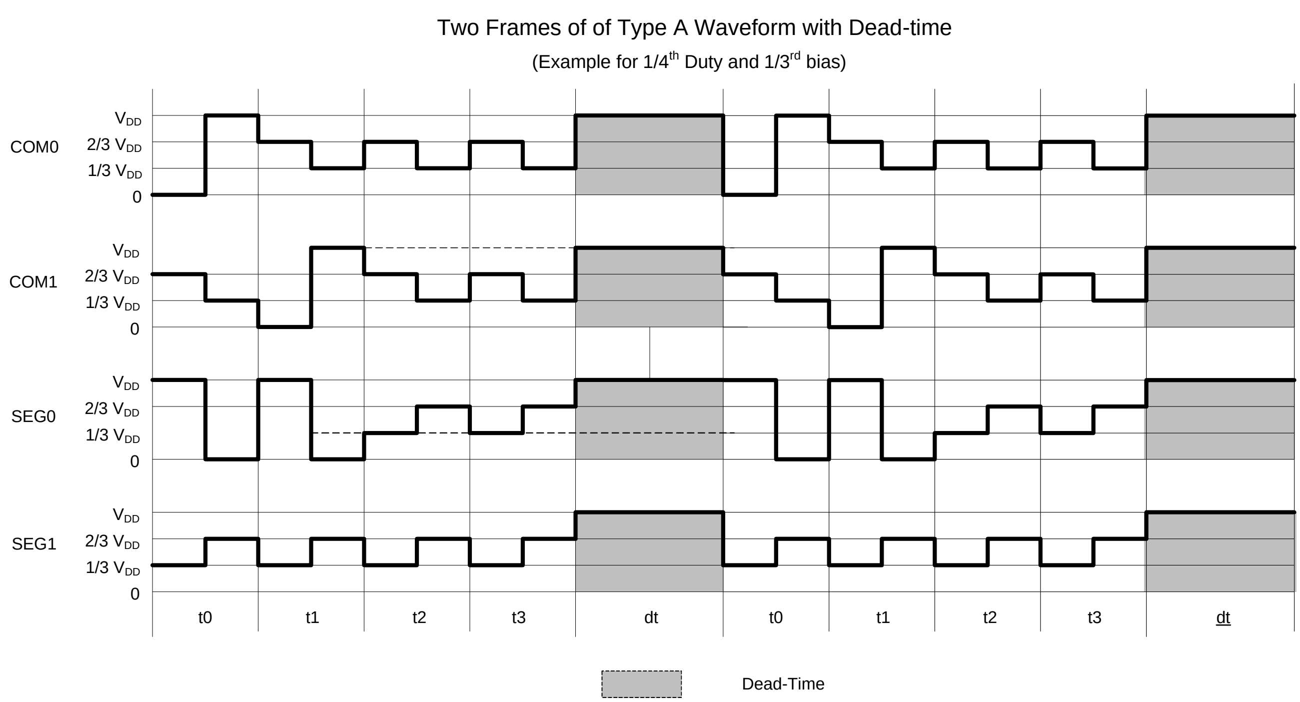
Table 22-1. Recommended Usage of Drive Modes

Display Type	Deep-Sleep Mode	Sleep/Active Mode	Notes
TN Glass	Digital Correlation	PWM 1/3 Bias	Firmware must switch between LCD drive modes before going to deep sleep or waking up.
STN Glass	Digital Correlation		No contrast advantage for PWM drive with STN glass.

22.2.3 Digital Contrast Control

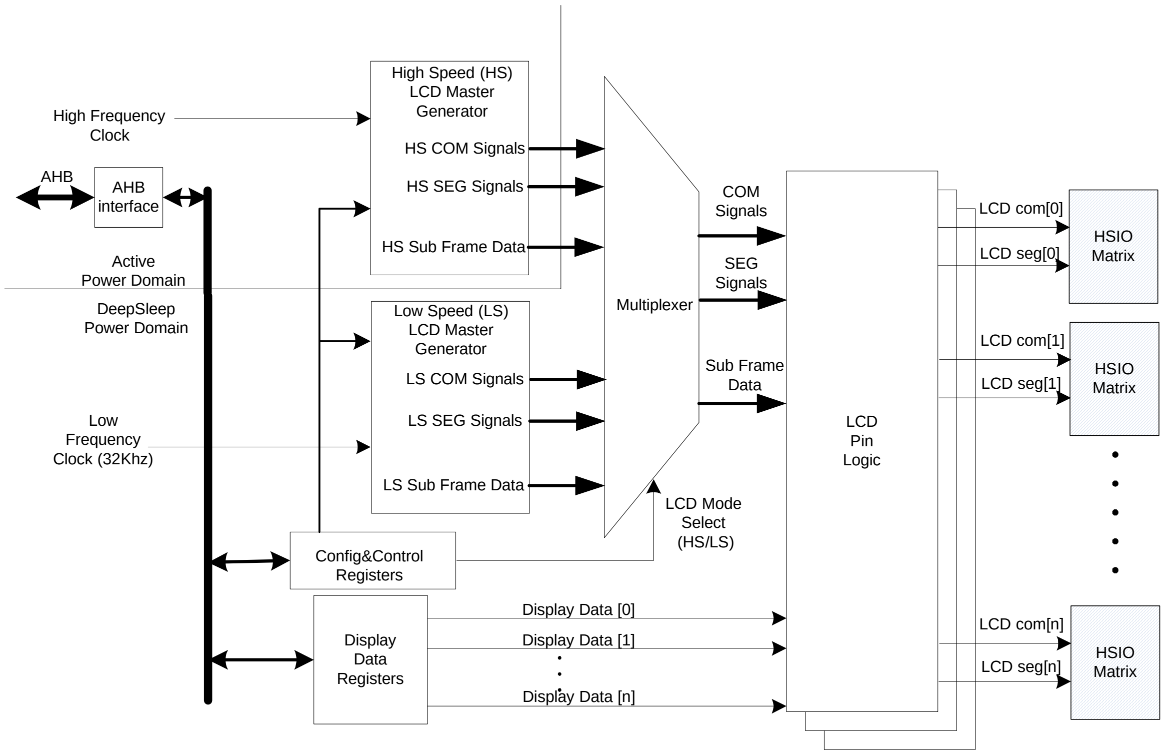
In all drive modes, digital contrast control can be used to change the contrast level of the segments. This method reduces contrast by reducing the driving time of the segments. This is done by inserting a 'Dead-Time' interval after each frame. During dead time, all COM and SEG signals are driven to a logic 1 state. The dead time can be controlled in fine resolution. Figure 22-8 illustrates the dead-time contrast control method for 1/3 bias and 1/4 duty implementation.

Figure 22-8. Dead-Time' Contrast Control



22.3 Block Diagram

Figure 22-9. Block Diagram of LCD Direct Drive System



22.3.1 How it Works

The LCD controller block contains two generators; one with a high-speed clock source HFCLK and the other with a low-speed clock source (32 kHz) derived from the ILO. These are called high-speed LCD master generator and low-speed LCD master generator, respectively. Both the generators support PWM and digital correlation drive modes. PWM drive mode with low-speed generator requires external resistors, as explained in PWM Drive on page 234.

The multiplexer selects one of these two generator outputs to drive LCD, as configured by the firmware. The LCD pin logic block routes the COM and SEG outputs from the generators to the corresponding I/O matrices. Any GPIO can be used as either COM or SEG. This configurable pin assignment for COM or SEG is implemented in GPIO and I/O matrix; see High-Speed I/O Matrix on page 54. These two generators share the same configuration registers. These memory mapped I/O registers are connected to the system bus (AHB) using an AHB interface.

The LCD controller works in three device power modes: active, sleep, and deep-sleep. High-speed operation is supported in active and sleep modes. Low-speed operation is supported in active, sleep, and deep-sleep modes. The LCD controller is unpowered in hibernate and stop modes.

22.3.2 High-Speed and Low-Speed Master Generators

The high-speed and low-speed master generators are similar to each other. The only exception is that the high-speed version has larger frequency dividers to generate the frame and sub-frame periods. This is because the clock of the high-speed block (HFCLK) is derived from the IMO, which is typically at 30 to 100 times the frequency of the ILO (32 kHz) clock fed to the low-speed block. The high-speed generator is in the active power domain and the low-speed generator is in the deep-sleep power domain. A single set of configuration registers is provided to control both high-speed and low-speed blocks. Each master generator has the following features and characteristics:

- Register bit configuring the block for either Type A or Type B drive waveforms (LCD_MODE bit in LCD_CONTROL register).
- Register bits to select the number of COMs (COM_NUM field in LCD_CONTROL register). The available values are 2, 3, and 4.
- Operating mode configuration bits enabled to select one of the following:
  - Digital correlation
  - PWM 1/2 bias
  - PWM 1/3 bias

- Off/disabled. Typically, one of the two generators will be configured to be Off

  OP_MODE and BIAS fields in LCD_CONTROL bits select the drive mode.

- A counter to generate the sub-frame timing. The SUBFR_DIV field in the LCD_DIVIDER register determines the duration of each sub-frame. If the divide value written into this counter is C, the sub-frame period is 4 × (C+1). The low-speed generator has an 8-bit counter. This generates a maximum half sub-frame period of 8 ms from the 32-kHz ILO clock. The high-speed generator has a 16-bit counter.
- A counter to generate the dead time period. These counters have the same number of bits as the sub-frame period counters and use the same clocks. DEAD_DIV field in the LCD_DIVIDER register controls the dead time period.

22.3.3 Multiplexer and LCD Pin Logic

The multiplexer selects the output signals of either high-speed or low-speed master generator blocks and feeds it to the LCD pin logic. This selection is controlled by the configuration and control register. The LCD pin logic uses the sub-frame signal from the multiplexer to choose the display data. This pin logic will be replicated for each LCD pin.

22.3.4 Display Data Registers

Each LCD pin has its own display data register (LCD_DATA0 to LCD_DATA3). If the pin is configured as COM, the display data for COM pins must be configured as follows, where the first listed value corresponds to the data output in the first subframe:

COM 0 – 1, 0, 0, 0

COM 1 – 0, 1, 0, 0

COM 2 – 0, 0, 1, 0

COM 3 – 0, 0, 0, 1

If the pin is configured as SEG, the display data register is programmed according to the display data of each sub-frame. The display data registers are Memory Mapped I/O (MMIO) and accessed through the AHB slave interface.

22.4 Register List

Table 22-2. LCD Direct Drive Register List

Register Name	Description
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_DATA0	LCD pin data register
LCD_DATA1	LCD pin data register
LCD_DATA2	LCD pin data register
LCD_DATA3	LCD pin data register

23. CapSense

PSoC® 4 uses a capacitive touch sensing method known as CapSense® Sigma Delta (CSD). The CapSense Sigma Delta touch sensing method provides the industry's best in class signal to noise ratio. CSD is a combination of hardware and firmware techniques. This chapter explains how the CSD hardware is implemented in PSoC 4. See the PSoC 4 CapSense Design Guide for more details on CSD operation, CapSense design tools, the PSoC Creator™ component, performance tuning, and design considerations.

23.1 Features

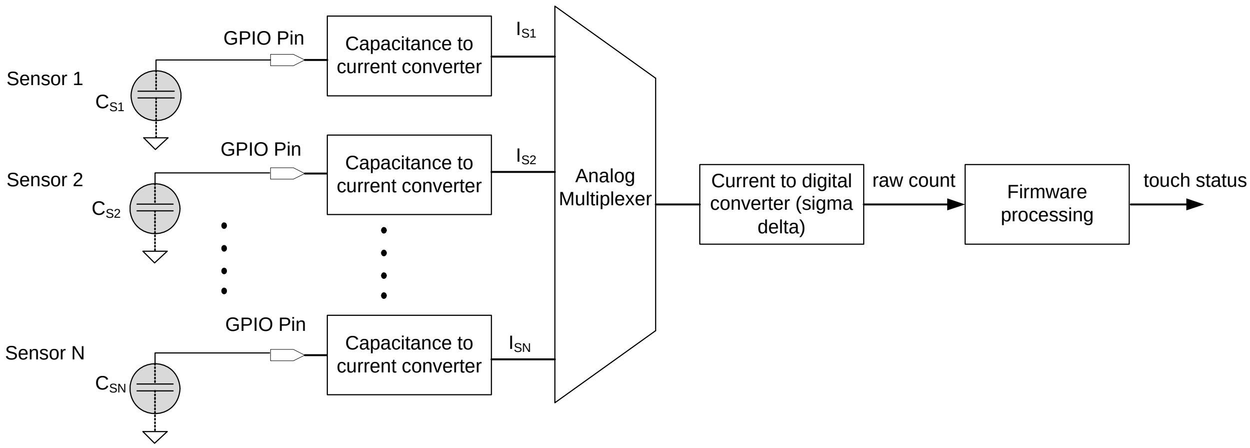
PSoC 4 CapSense has the following features:

- Robust sensing technology
- CapSense Sigma Delta (CSD) operation provides best in class signal-to-noise ratio (SNR)
- High-performance sensing across a variety of overlay materials and thicknesses
- SmartSense™ auto-tuning technology
- Supports as many as 35 sensors
- High range proximity sensing
- Water tolerant operation
- Low power consumption
- Two IDAC operation to increase scan speed and SNR
- Any GPIO pin can be used for sensing or shielding
- Pseudo random sequence (PRS) clock source for lower electromagnetic interference (EMI)
- GPIO precharge (supported on two dedicated pins) quickly initializes external tank capacitors

23.2 Block Diagram

Figure 23-1 shows the CapSense Sigma Delta (CSD) system block diagram.

Figure 23-1. CapSense Module Block Diagram



23.3 How It Works

With CSD, each GPIO has a switched capacitance circuit that converts the sensor capacitance into an equivalent current. An analog multiplexer then selects one of the currents and feeds it into the current to digital converter. The current to digital converter is similar to a Delta Sigma ADC.

The output count of the current to digital converter, known as raw count, is a digital value that is proportional to the capacitance of the sensor C_S:

$$\text{rawcount} = G_C C_S$$

Equation 23-1

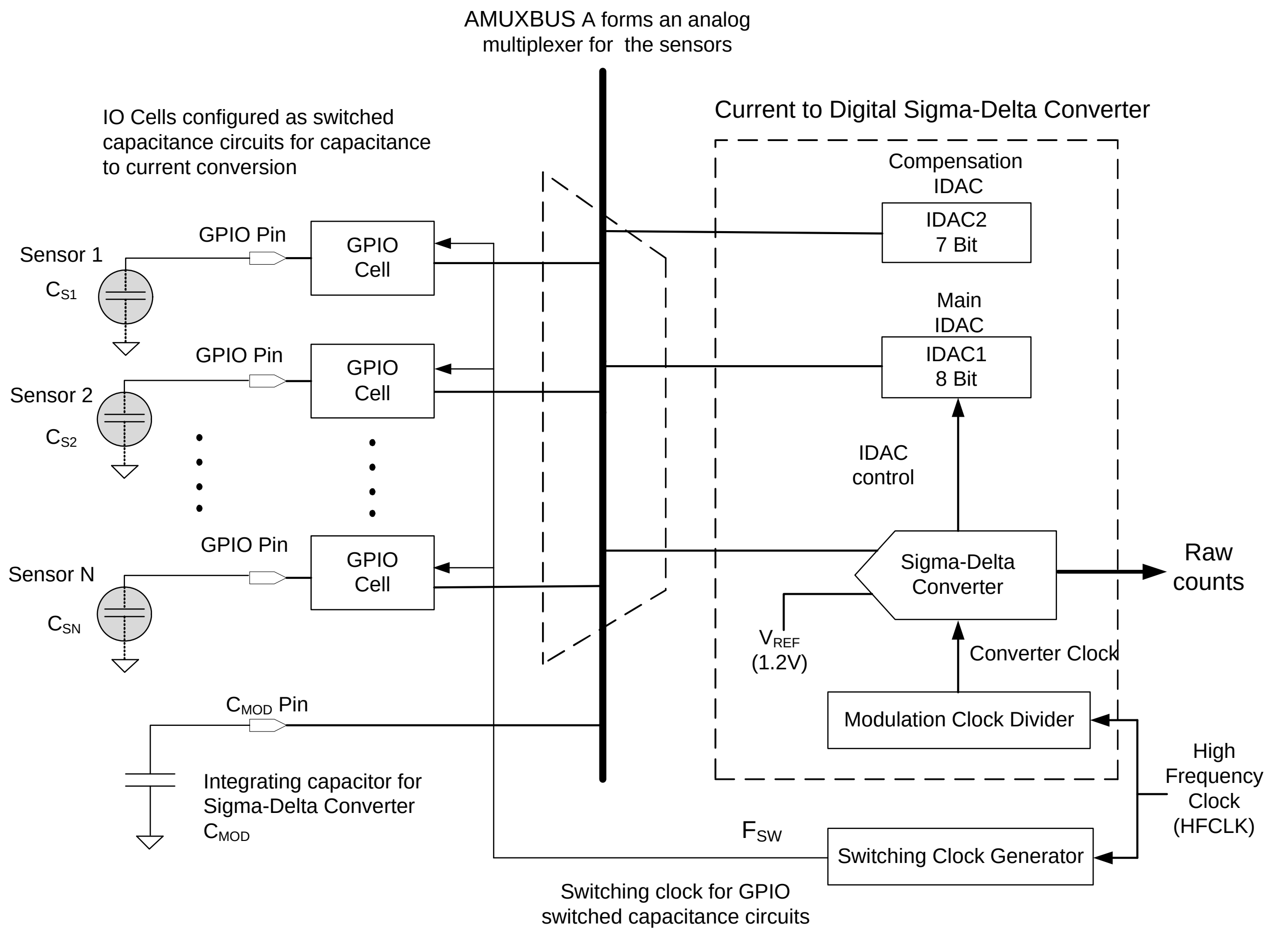
Where G_C is the capacitance to digital conversion gain of CapSense.

When a finger touches the sensor, the sensor capacitance increases; the raw count also increases proportionally. By comparing the change in raw count to a predetermined threshold, logic in firmware decides whether the sensor is active (finger is present).

23.3.1 CapSense CSD Sensing

Figure 23-2 shows the block diagram of the PSoC 4 CapSense block, which scans the CapSense sensors.

Figure 23-2. PSoC 4 CapSense CSD Sensing



23.3.1.1 GPIO Cell Capacitance to Current Converter

In the CapSense CSD system, the GPIO cells are configured as switched capacitance circuits that convert the sensor capacitances to equivalent currents. Figure 23-3 shows a simplified diagram of the PSoC 4 GPIO cell structure.

PSoC 4 has two analog multiplexer buses: AMUXBUS A is used for CSD sensing and AMUXBUS B is used for CSD shielding. The GPIO switched capacitance circuit has two possible configurations: source current to AMUXBUS A or sink current from AMUXBUS A. Figure 23-4 shows the switched capacitance configuration for sourcing current to AMUXBUS A.

Figure 23-3. PSoC 4 GPIO Cell

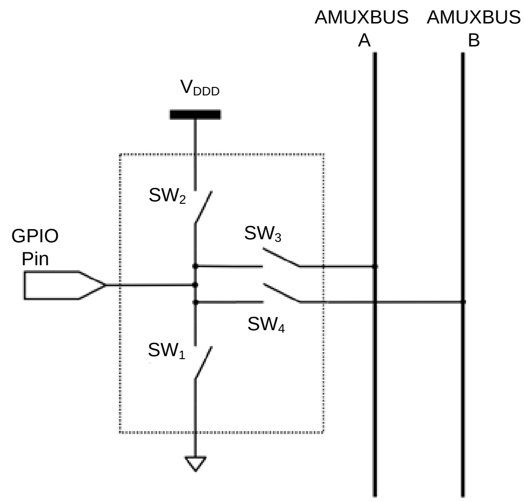
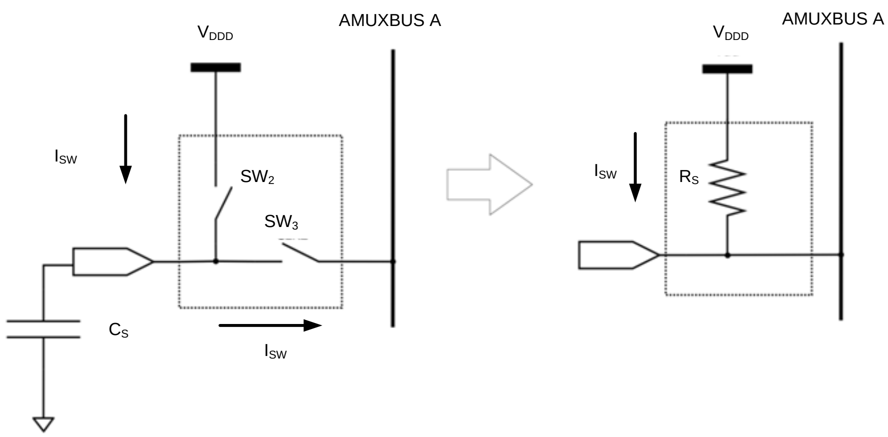


Figure 23-4. Sourcing Current to AMUXBUS A



Two non-overlapping, out of phase clocks of frequency F_{SW} (see Figure 23-1) control the switches SW_2 and SW_3. The continuous switching of SW_2 and SW_3 forms an equivalent resistance R_S, as Figure 23-3 shows. The value of the equivalent resistance R_S is:

$$R_S = \frac{1}{C_S F_{SW}}$$

Equation 23-2

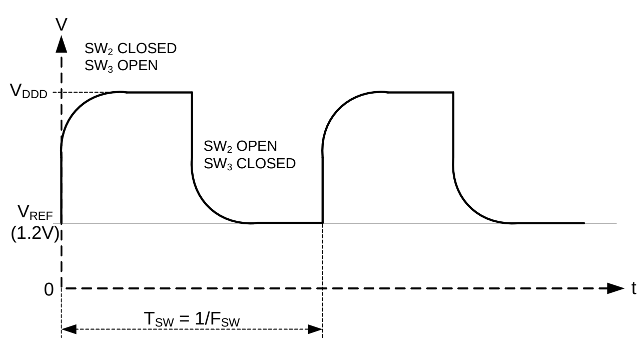
Where:

C_S = Sensor capacitance

F_{SW} = Frequency of the switching clock

The Sigma Delta converter maintains the voltage of AMUXBUS A at a constant V_{REF} (this process is explained in Sigma Delta Converter). Figure 23-5 shows the voltage waveform across the sensor capacitance.

Figure 23-5. Voltage Across Sensor Capacitance



Equation 23-3 gives the value of average current supplied to AMUXBUS A.

$$I_S = C_S F_{SW}(V_{DDD} - V_{REF})$$

Equation 23-3

Figure 23-6 shows the switched capacitance configuration for sinking current from AMUXBUS A. Figure 23-7 shows the resulting voltage waveform across C_S.

Figure 23-6. Sinking Current from AMUXBUS A

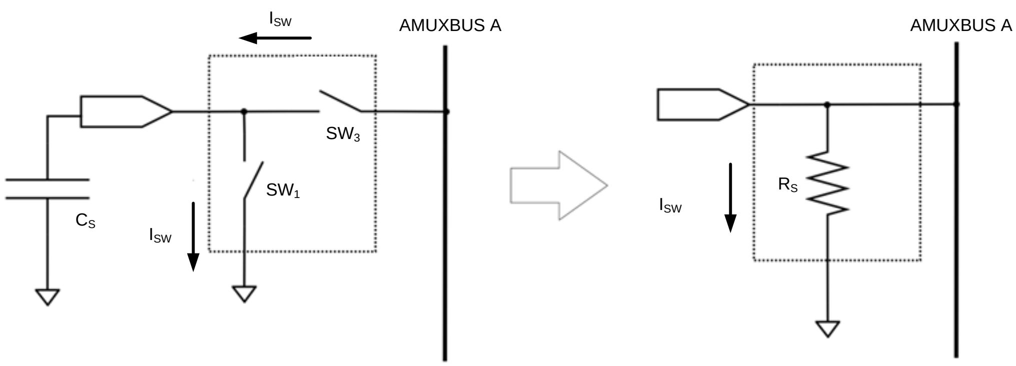
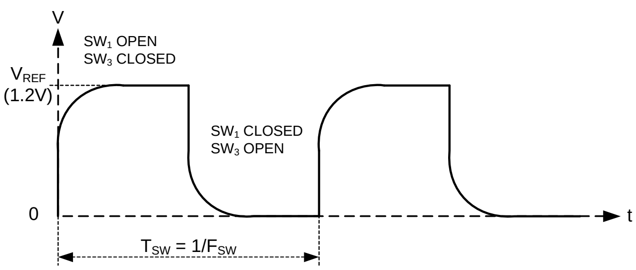


Figure 23-7. Voltage Across Sensor Capacitance



Equation 23-4 gives the value of average current taken from AMUXBUS A.

$$I_S = C_S F_{SW} V_{REF}$$

Equation 23-4

23.3.1.2 Switching Clock Generator

This block generates the switching clock F_{SW} from the high-frequency clock (HFCLK), as Figure 23-1 shows. The switching clock is required for the GPIO cell switched capacitance circuits. The switching clock generator output has three options: direct, 8-bit pseudo random sequence (PRS), and 12-bit PRS. You can set the desired switching frequency by selecting a clock divider parameter of the switching clock generator. This clock divider parameter is known as the analog switch divider. If the "direct" output is selected, the value of the generated switching clock frequency F_{SW} is

$$F_{SW} = \frac{HFCLK}{2 AnalogSwitchDivider}$$

Equation 23-5

You can also select one of the PRS outputs to lower the Electro Magnetic Interference (EMI) effect - it averages the switching frequency over a wide range. If PRS output is selected, Equation 23-5 gives the average value of the average value of F_{SW}. Equations 23-6 and 23-7 give the maximum and minimum frequencies.

$$F_{SW}(\text{maximum}) = \frac{HFCLK}{AnalogSwitchDivider}$$

Equation 23-6

$$F_{SW}(\text{minimum}) = \frac{HFCLK}{mAnalogSwitchDivider}$$

Equation 23-7

Where m is the resolution of the PRS (8 or 12 bits).

23.3.1.3 Sigma Delta Converter

The Sigma Delta converter converts the input current to a corresponding digital count. It consists of a Sigma Delta converter, a clock generator, known as a modulation switch divider, and two current sourcing/sinking digital-to-analog converters (IDACs), as Figure 23-1 shows. The 8-bit IDAC1 is known as the main IDAC and the 7-bit IDAC2 is known as the compensation IDAC. IDAC2 is not required for basic CSD operation; it is used for improving CSD performance. The Sigma Delta converter also requires an external integrating capacitor C_{MOD}, as Figure 23-1 shows. The recommended value of C_{MOD} is 2.2 nF.

The Sigma Delta modulator maintains the voltage across C_{MOD} at V_{REF}. It works in one of the following modes:

- IDAC sourcing mode: If the switched capacitor circuit sinks current from the AMUXBUS A, the IDACs then source current to AMUXBUS A to balance its voltage. The IDAC1 current is switched ON and OFF corresponding to the small voltage variations across C_{MOD} to maintain this voltage at V_{REF}.

- IDAC sinking mode: In this mode, the IDACs sink current from C_{MOD}, and the switched capacitor circuit sources current to C_{MOD}. The IDAC1 current is switched ON and OFF corresponding to the small voltage variations across C_{MOD} to maintain this voltage at V_{REF}.

The delta-sigma modulator has a maximum resolution of 16 bits. This is achieved by feeding the comparator output into a counter, which counts the number of clock cycles the input is above comparator threshold. The modulator uses a 16-bit counter. To achieve lower resolutions, the period value of the counter is adjusted.

If IDAC2 is not used, the raw count is proportional to the sensor capacitance. If 'N' is the resolution of the Sigma Delta converter and IDAC1 is the value of IDAC1 current, the approximate value of raw count in IDAC sourcing mode is given by Equation 23-8.

$$\text{Rawcount} = 2^N \frac{V_{REF} F_{SW}}{I_{DAC1}} C_S$$

Equation 23-8

Similarly, the approximate value of raw count in IDAC sinking mode is:

$$\text{Rawcount} = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{DAC1}} C_S$$

Equation 23-9

In both cases, the raw count is proportional to sensor capacitance C_S. The raw count is then processed by the CapSense firmware to detect touches.

You can use IDAC2 to increase the performance of CapSense. When IDAC2 is used, the equation for the raw count in IDAC sourcing mode is:

$$\text{Rawcount} = 2^N \frac{V_{REF} F_{SW}}{I_{DAC1}} C_S - 2^N \frac{I_{DAC2}}{I_{DAC1}}$$

Equation 23-10

Where IDAC2 is the value of IDAC2 current, raw count in IDAC sinking mode is given by equation 23-11.

$$\text{Rawcount} = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{DAC1}} C_S - 2^N \frac{I_{DAC2}}{I_{DAC1}}$$

Equation 23-11

23.3.1.4 Analog Multiplexer

The Sigma Delta converter scans one sensor at a time. An analog multiplexer selects one of the GPIO cells and connects it to the input of the Sigma Delta converter, as Figure 23-1 shows. The AMUXBUS A and the GPIO cell switches (see SW_3 in Figure 23-3) form this analog multiplexer. AMUXBUS A connects to all GPIOs, so you can use any PSoC 4 GPIO for CSD sensing. AMUXBUS A also connects the integrating capacitor C_{MOD} to the Sigma Delta converter circuit.

23.3.2 CapSense CSD Shielding

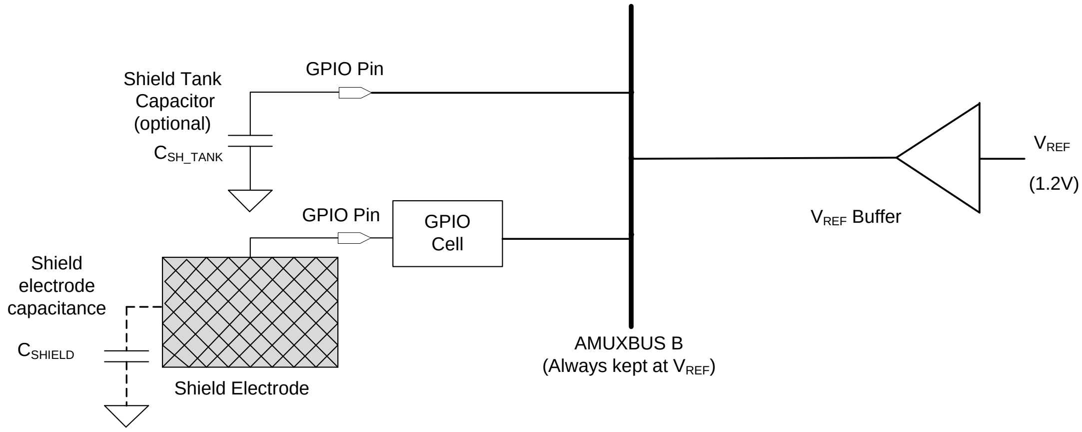
PSoC 4 CapSense supports shield electrodes for waterproofing and proximity sensing. For waterproofing, the shield electrode is always kept at the same potential as the sensors. PSoC 4 CapSense has a shielding circuit that drives the shield electrode with a replica of the sensor switching signal (see GPIO Cell Capacitance to Current Converter) to nullify the potential difference between sensors and shield electrode.

In the sensing circuit, the Sigma Delta converter keeps the AMUXBUS A at V_{REF} (see Sigma Delta Converter). The GPIO cells generate the sensor waveforms by switching the sensor between AMUXBUS A and a supply rail (either V_{DD} or ground, depending on the configuration). The shielding circuit works in a similar way; AMUXBUS B is always kept at V_{REF}. The GPIO cell switches the shield between AMUXBUS B and a supply rail (either V_{DDD} or ground, same configuration as the sensor). This process generates a replica of the sensor switching waveform on the shield electrode.

Depending on how AMUXBUS B is kept at V_{REF}, two different configurations are possible.

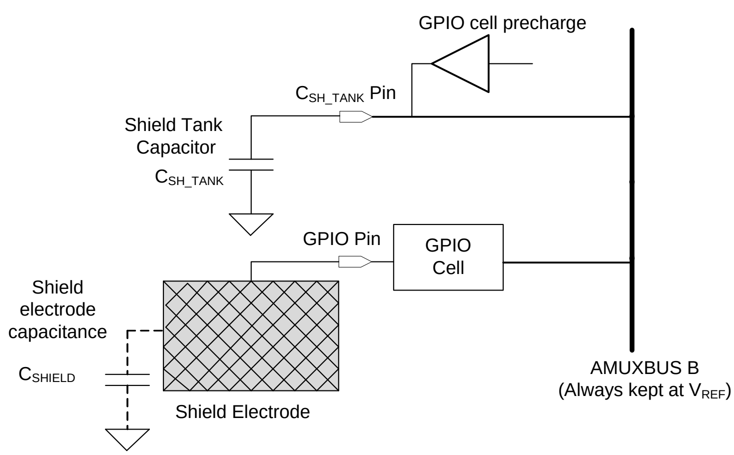
Shield driving using V_{REF} buffer: In this configuration, a voltage buffer is used to drive AMUXBUS B to V_{REF}, as Figure 23-8 shows. An external C_{SH_TANK} capacitor is recommended to reduce switching transients. See Chapter 3 in the PSoC 4 CapSense Design Guide for details.

Figure 23-8. Shield Driving Using V_{REF} Buffer



Shield driving using GPIO cell precharge: This configuration requires an external C_{SH_TANK} capacitor, as Figure 23-9 shows. A special GPIO cell charges the C_{SH_TANK} capacitor and hence the AMUXBUS B to V_{REF}.

Figure 23-9. Shield Driving Using GPIO Precharge



This GPIO cell precharge capability is available only on a fixed C_{SH_TANK} pin. See the device pinout in the PSoC 4 datasheet for details.

23.3.2.1 C_{MOD} Precharge

When the CapSense hardware is enabled for the first time, the voltage across C_{MOD} starts at zero. Then the Sigma Delta converter slowly charges the C_{MOD} to V_{REF}. The charging current is supplied by the IDACs in the IDAC sourcing mode and it is supplied by the sensor switched capacitance circuit in IDAC sinking mode. However, this is a slow process because C_{MOD} is a relatively large capacitor.

Precharging of C_{MOD} is the process of quickly initializing the voltage across C_{MOD} to V_{REF}. Precharging is used to reduce the time required for the Sigma Delta converter to start its operation. There are two options for precharging C_{MOD}.

- Precharge using V_{REF} buffer: When the shield is enabled, the V_{REF} buffer output is always connected to AMUXBUS B (Figure 23-8). To precharge using the V_{REF} buffer, C_{MOD} is initially connected to AMUXBUS B. After the precharging process, C_{MOD} is connected to AMUXBUS A for normal Sigma Delta operation. When the shield is disabled, the V_{REF} buffer output is always connected to AMUXBUS A for precharging, and disconnected afterwards.
- Precharge using GPIO cell: In this configuration, a special GPIO cell charges the C_{MOD} capacitor to V_{REF}. This GPIO cell precharge capability is available only on a fixed C_{MOD} pin. See the device pinout in the PSoC 4 datasheet for details. Precharge using a GPIO cell is faster than using the V_{REF} buffer. Therefore, GPIO precharge is the recommended precharge configuration. However, if you do not need a fast initialization of CapSense, use V_{REF} buffer precharge. In this mode, you can connect C_{MOD} to any GPIO. When precharging using a GPIO cell, C_{MOD} must be charged through AMUXBUS-B.

24. Temperature Sensor

PSoC® 4 has an on-chip temperature sensor that is used to measure the internal die temperature. The temperature sensor is a transistor connected in diode configuration. The temperature dependence of the base-to-emitter voltage (Vbe) is the basis for temperature measurement.

24.1 Features

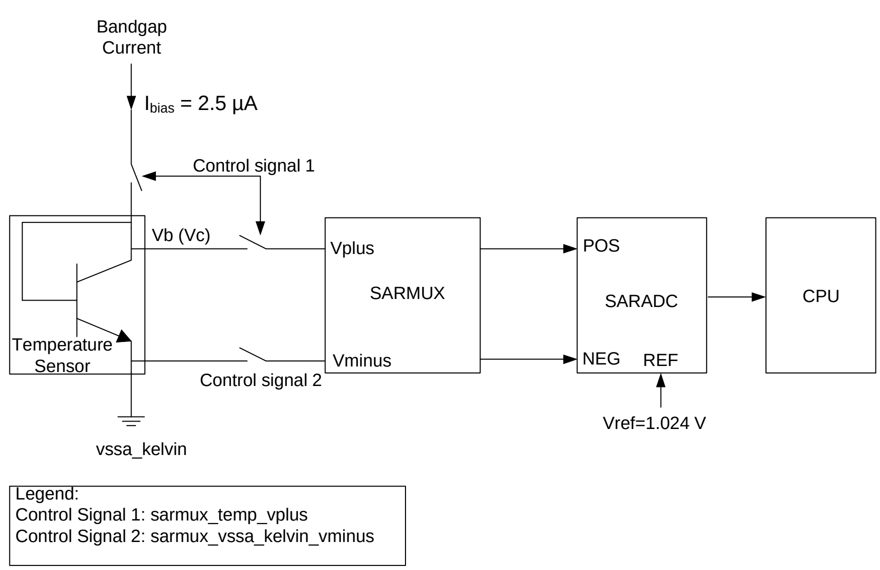
The temperature sensor has these features:

- ± 5° Celsius accuracy over temperature range –40 °C to +100 °C
- 0.5° Celsius/LSB resolution
- 10 µs setting (sampling) time

24.2 How it Works

The base-to-emitter voltage of a bipolar junction transistor (BJT) device has a strong dependence on temperature at a constant collector current and zero collector-base voltage. The PSoC uses this property to calculate the die temperature by measuring the base-emitter voltage (Vbe) using SARMUX channel and SAR ADC in 12-bit mode, single-ended, and unsigned configuration, as shown in Figure 24-1.

Figure 24-1. Temperature Sensing Mechanism



The digital output of SAR ADC is calibrated in firmware using the linear equation:

$$\text{Temp} = A \times \text{Vbe} + B$$

Equation 24-1

Note A and B are 16-bit constants stored in flash during factory calibration. You will not be able to alter these values.

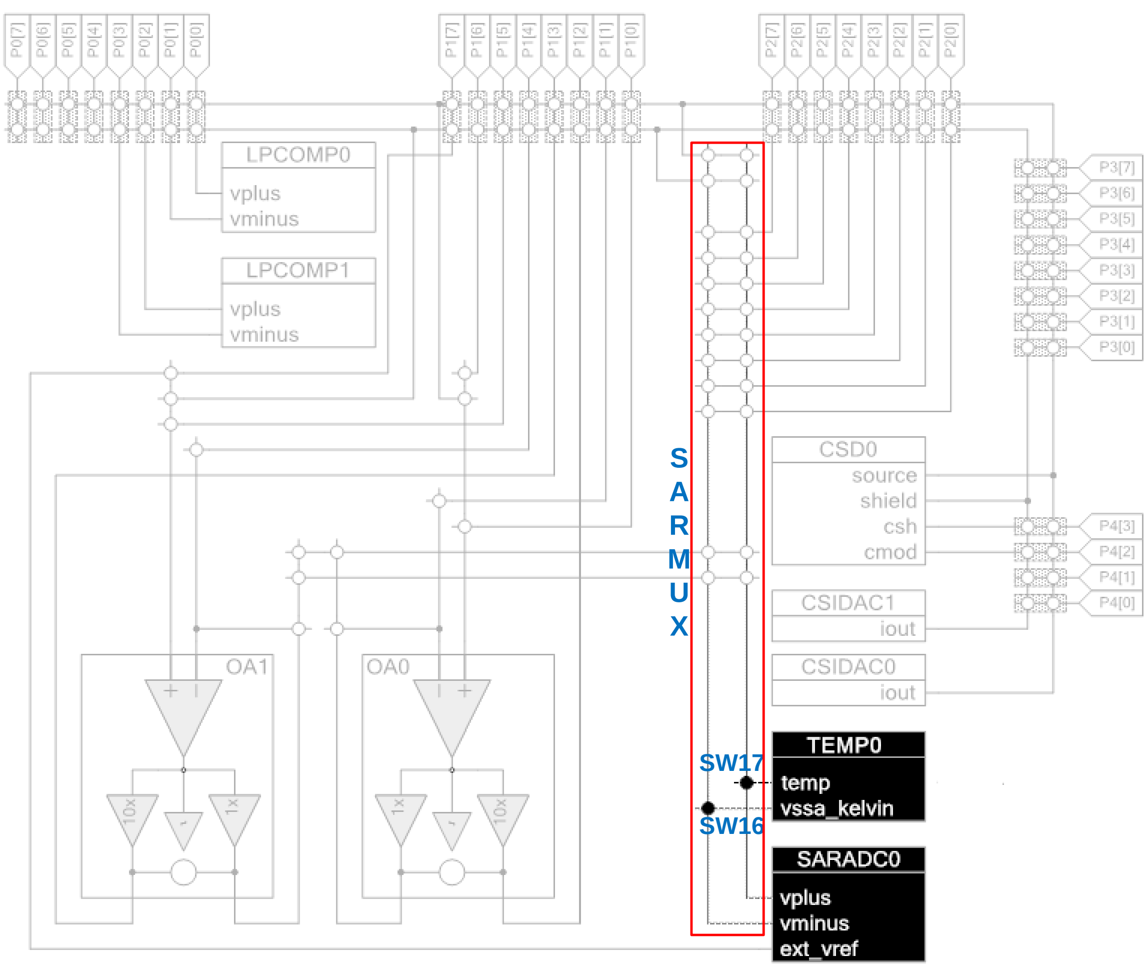
- "A" is the 16-bit multiplier constant. The value of A is determined by the PSoC 4 family characterization data, and is a constant value for all die. It is stored in a PSoC Creator defined resistor CYREG_SFLASH_SAR_TEMP_MULTIPLIER at the location 0x0FFFF164. A and 16-bit Vbe are multiplied and the 32-bit product is stored in 16.16 fixed point format.
- "B" is the 16-bit offset constant. The value of B is determined on a per die basis by taking care of all the process variations and the actual bias current (Ibias) present in the chip. It is stored in a PSoC Creator defined resistor CYREG_SFLASH_SAR_TEMP_OFFSET at the location 0x0FFFF166. B is multiplied by 1024 to get the 32-bit product in 16.16 fixed point format.
- "Temp" is the die temperature expressed in 16.16 fixed point format. The upper 16 bits represent the integer part of temperature and the lower 16 bits represent the decimal part of temperature. The 32-bit Temp value is right shifted by 16-bit positions to get the integer part of temperature in degree Celsius. For example, 0x1E5DFC = to 30.36713 °C; 0xFFFD09D2 = –2.96164 °C.

24.3 Temperature Sensor Configuration

In Figure 24-2, the temperature sensor output is routed to the positive input of SAR ADC via dedicated switches, which can be controlled by sequencer, firmware or Digital System Interconnect (DSI). The control signal for switch-17 (sarmux_temp_vplus) enables the temperature sensor by passing bias current from bandgap and by routing the sensor output to the positive input of SAR ADC. The control signal for switch-16 (sarmux_vssa_kelvin_vminus) connects the negative input of SAR ADC to V_{SSA}.

See Temperature Sensor Configuration on page 221 to know how the sequencer, DSI, or firmware routes the temperature sensor output to SAR ADC.

Figure 24-2. Routing Temperature Sensor Output to SAR ADC



Note that for temperature sensor, the differential conversions are not available (conversion result is undefined). Therefore, always use it in singled-ended mode. Reference is from internal 1.024 V.

24.4 Algorithm

1. Enable the SARMUX and SAR ADC.
2. Configure SAR ADC in single-ended mode with $V_{NEG} = V_{SS}$, V_{REF} = 1.024 V, and 12-bit resolution.
3. Enable the temperature sensor.
4. Get the digital output from the SAR ADC.
5. Fetch A value from CYREG_SFLASH_SAR_TEMP_MULTIPLIER and B from CYREG_SFLASH_SAR_TEMP_OFFSET.
6. Calculate the die temperature using the linear equation Temp = A × Vbe+ B

   For example, let A = 0xBC4B and B = 0x65B4. Assume that the output of SAR ADC (Vbe) is 0x595 at a given temperature.

   Firmware does the following calculations:

   a. Multiply A and Vbe: 0xBC4B × 0x595 = $(-17333)_{10} \times (1429)_{10} = (-24768857)_{10}$
   b. Multiply B and 1024: 0x65B4 × 0x400 = $(26036)_{10} \times (1024)_{10} = (26660864)_{10}$
   c. Add the result of step 1 and 2: $(-24768857)_{10} + (26660864)_{10} = (1892007)_{10}$ = 0x1CDEA7
   d. The integer part of temperature is the upper 16 bits = 0x001C = $(28)_{10}$
   e. The decimal part of temperature is the lower 16 bits = 0xDEA7 = $(0.86974)_{10}$
   f. Combining the result of step 4 and 5, Temp = 28.869740 °C

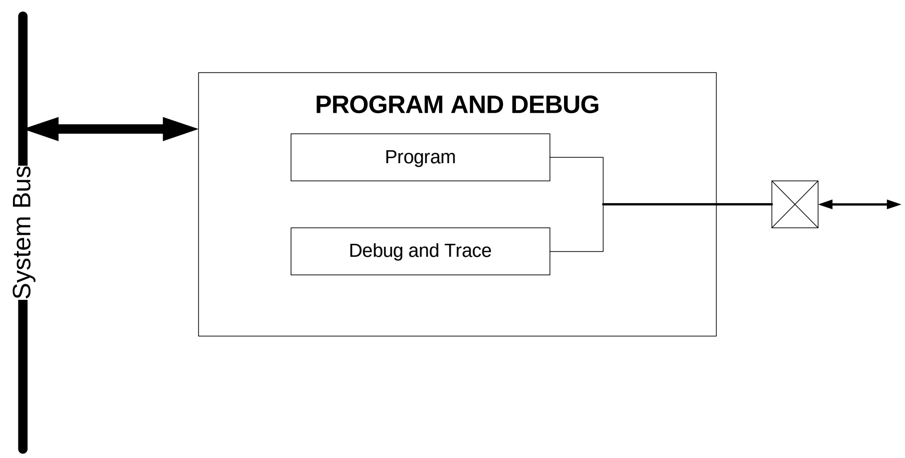
Section G: Program and Debug

This section encompasses the following chapters:

-
-

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 PSoC 4 programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and PSoC 4.

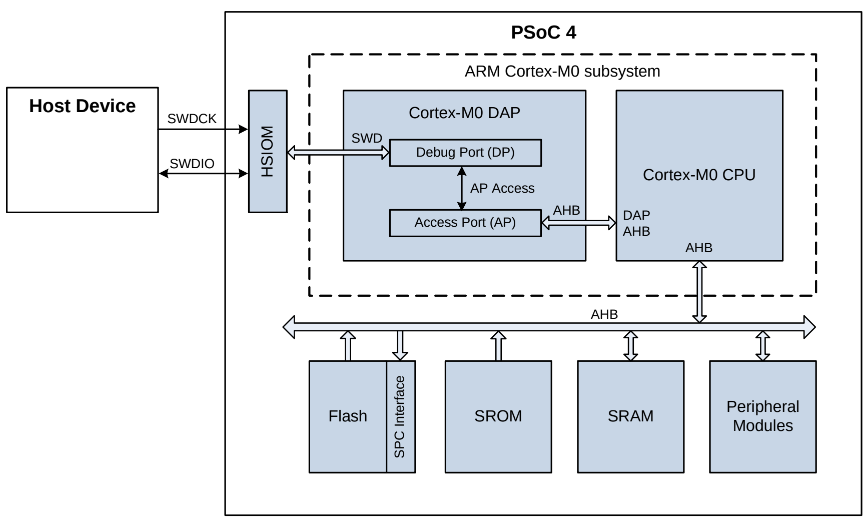
25.1 Features

- 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 Cortex-M0 register bank when the core is running or halted

25.2 Functional Description

Figure 25-1 shows the block diagram of the program and debug interface in PSoC 4. The Cortex-M0 debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the "host", communicates with the DAP of the PSoC 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 high-speed IO matrix (HSIOM). The HSIOM provides a flexible mapping between the GPIO pins and the different on-chip peripherals that connect to the GPIO pins such as the LCD driver, DAP, and SAR ADC.

Figure 25-1. PSoC 4 Program and Debug Interface



The DAP communicates with the Cortex-M0 CPU using the ARM-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside PSoC 4, which facilitates memory and peripheral register access by the AHB master. PSoC 4 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

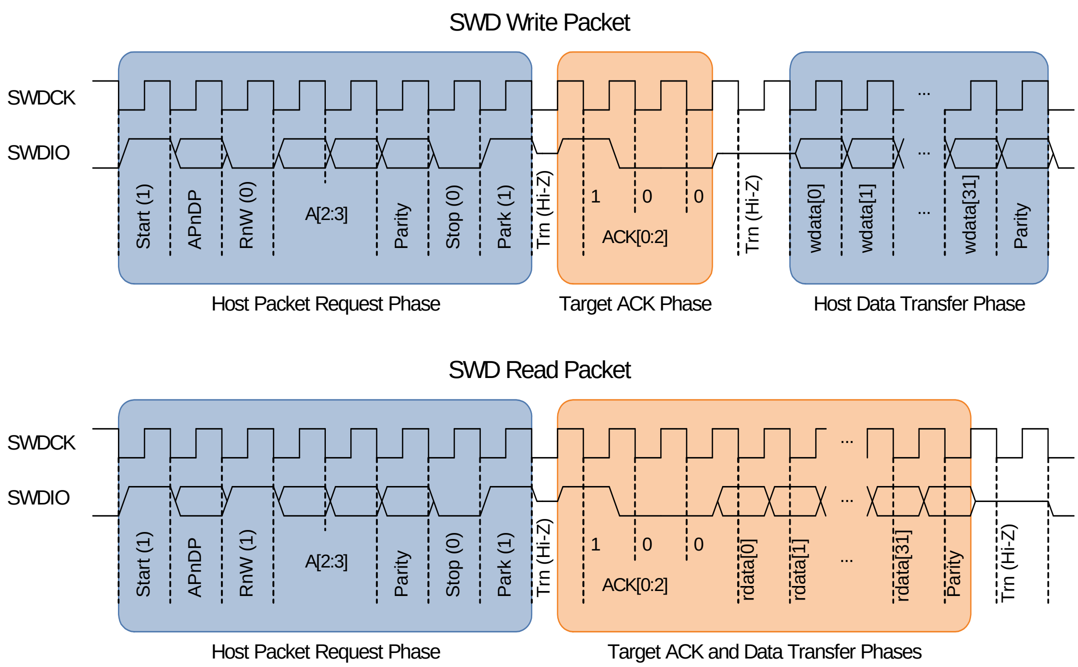
PSoC 4's Cortex-M0 supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete data transfer (one SWD packet) requires 46 clocks and consists of three phases:

- **Host Packet Request Phase** – The host issues a request to the PSoC 4 target.
- **Target Acknowledge Response Phase** – The PSoC 4 target sends an acknowledgement to the host.
- **Data Transfer Phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

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

Figure 25-2. PSoC 4 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 a logical 1.
   b. The "AP not DP" (APnDP) bit determines whether the transfer is an AP access – 1b1 or a DP access – 1b0.
   c. The "Read not Write" bit (RnW) controls which direction the data transfer is in. 1b1 represents a 'read from' the target, or 1b0 for a 'write to' the target.
   d. The Address bits (A[3:2]) are register select bits for AP or DP, depending on the APnDP bit value. See Table 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 the PSoC 4. But, the host must still send the data to be written to complete the packet. The parity bit corresponding to the data should also be sent by the host.
- For a WAIT response, it means that the PSoC 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 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 Cortex-M0 Debug and Access Port (DAP)

The Cortex-M0 program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex-M0 DAP-AHB interface. Typically, the DP registers are used for a one time configuration or for error detection purposes, and the AP registers are used to perform the programming and debugging operations. Complete architecture details of the DAP is available in the *ARM® Debug Interface v5 Architecture Specification*.

25.4.1 Debug Port (DP) Registers

Table 25-3 shows the Cortex-M0 DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the *ARM® Debug Interface v5 Architecture Specification* for details on all of the DP registers.

Table 25-3. Main Debug Port (DP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex-M0 CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In PSoC 4, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

25.4.2 Access Port (AP) Registers

Table 25-4 lists the main Cortex-M0 AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 25-4. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the PSoC 4 Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register

25.5 Programming the PSoC 4 Device

PSoC 4 is programmed using the following sequence. Refer to the *PSoC 4 Device Programming Specifications* for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in 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 Primary and Secondary SWD Pin Pairs

The first step in device programming is to acquire the SWD port in PSoC 4. PSoC 4 devices support the SWD interface in two different physical pin locations – the primary pin pair of P3[2] and P3[3], and the secondary pin pair of P3[6] and P3[7]. Within those pairs, P3[2] and P3[6] are used for SWDIO, and P3[3] and P3[7] are used for SWDCK. The primary SWD pin pair is supported on all device packages while the secondary SWD pin pair is present only in the higher pin count packages. Refer to the PSoC 4 device datasheet for information on which device packages support the secondary SWD pin pair.

The PSoC 4 SWD_CONFIG register in the supervisory flash region is used to select between one of the two SWD pin pairs that can be used for programming and debugging. Note that only one of the SWD pin pairs can be used during any programming or debugging session. The default selection for devices coming from the factory is the primary SWD pin pair. To select the secondary SWD pin pair, it is necessary to program the device using the primary pair with the hex file that enables the secondary pin pair configuration. Afterwards, the secondary SWD pin pair may be used.

25.5.1.2 SWD Port Acquire Sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After deasserting the XRES signal, the host must send an SWD connect sequence 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 Pseudocode

```
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 < 1.5 ms);  //retry connection until OK ACK or timeout

if (time_elapsed >= 1.5 ms) return FAIL; //check for acquire time out

if (ID != CM0_ID) return FAIL; //confirm SWD ID of Cortex-M0 CPU. (0x0BB11477)
```

In this pseudocode, SWD_LineReset() is the standard ARM command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (1.5 ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex-M0 DAP.

25.5.2 SWD Programming Mode Entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudocode for entering the programming mode are shown in the device programming specification.

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 265. The exact sequence of calling the programming routines is given in the device programming specifications document.

25.6 PSoC 4 SWD Debug Interface

Cortex-M0 DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex-M0 Debug.

See the *ARMv6-M Architecture Reference Manual* for complete details on the debug architecture.

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 (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 (DFSR) – This register describes the reason a debug event has occurred. This includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (DCRSR) – This register is used to select the general-purpose register in the Cortex-M0 CPU to which a read or write operation must be performed by the external debugger.

- Debug Core Register Data Register (DCRDR) – This register is used to store the data to write to or read from the register selected in the DCRSR register.
- Debug Exception and Monitor Control Register (DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

25.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0 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 Cortex-M0. The BPU has two types of registers.

- The breakpoint control register 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 PSoC 4).
- Each hardware breakpoint has a Breakpoint Compare Register (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 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. Trace is not supported by the Cortex-M0 in PSoC 4. 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 (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 (DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event has occurred.
- The watchpoint comparator PC sample register (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 device by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0 DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PSoC 4 device is to acquire the SWD port. The acquire sequence consists of a 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. The PORT_SEL3 register contains the bits 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 repurposed. If only programming support is needed, the SWD pins can be repurposed SWD programming and SWD debugging differ in that after SWD programming, the pins can be used for any purpose. But during SWD debugging, the pins cannot be used for another purpose.

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 87. Only the OPEN protected mode supports device debugging. Also, the external debugger loses connection to the target device when the device enters either Hibernate or Stop modes. The connection must be re-established after the device enters the Active mode again. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

26. 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. These functions can be used by Cypress supplied programmers and other third-party programmers to program the PSoC 4 device with the data in an application hex file. They can also be used to perform bootload operations where a portion of flash memory will be updated by the CPU.

26.1 Features

- Supports programming through the debug and access port (DAP) and Cortex-M0 CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex-M0 CPU

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 SPC input registers, and then requesting the SROM to execute the function. Based on the function opcode, the SPC will execute the corresponding system call from SROM and update the SPC 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 will interact with the SPC interface 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 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 Debug and Access Port (DAP). The programming sequence for the PSoC 4 device with an external programmer is given in the PSoC 4 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 System Performance Controller (SPC) interface.

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 85) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. Also, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

26.3 System Call Implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: There are two 8-bit parameters that are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows.

  key1 = 0xB6

  key2 = 0xD3 + Opcode

The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, there may be additional parameters 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 are those that 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 system call is initiated by the CPU. The DAP will only use system calls during programming mode and the CPU is halted during this process.

The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

Because the CPU cannot execute code from flash while doing an erase or program operation on the flash, the non-blocking system calls can only be called from a code executing out of SRAM. If the non-blocking functions are called from flash memory, the result is undefined and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

The System Performance Controller (SPC) is the block that generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called from SRAM, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure the subsequent steps in the system call are completed. Because the CPU can execute code only from SRAM when a non-blocking write or program operation is being done, the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. The Resume Non-Blocking function call done in the SPC ISR is called once in a non-blocking program operation and thrice in a non-blocking write operation.

The pseudo code for using a non-blocking write system call and executing user code out of SRAM is given later in this chapter.

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 initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits are not cleared after 1 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 SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.
4. Check 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 87 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

Table 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 128 bytes	✔			✔
Write Row	Erases and then programs a row of flash with data in the page latch buffer	✔			✔
Program Row	Programs a row of flash with data in the page latch buffer	✔			✔
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	✔			
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔		✔
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	✔	✔		
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access				✔
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access				✔
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access				✔

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	See the device datasheet for Silicon ID values for different part numbers
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision Id	See the device programming specification 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 0x093 for PSoC 4
Bits [15:12]	Chip Protection	Refer Device Security chapter
Bits [31:16]	0xXXXX	Not used

26.5.2 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. 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 0x80 – Byte 128 of latch buffer
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 (Refer Device Memory chapter for 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
.	.	.
.	.	.
Byte (Load size –1)	Data Byte [Load size –1]	Last data byte to be loaded
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]	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)

26.5.3 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 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.

Note that this system call disables the 36-MHz IMO output before performing the flash write operation. The 36-MHz IMO output can be used to source the analog switch pump or CTBm pump. If the 36-MHz IMO output is used, it must be manually re-enabled after the system call completes. Specifically, the CLK_IMO_CONFIG EN_CLK36 and FLASHPUMP_SEL must be reset.

Refer to the CLK_IMO_CONFIG register in the PSoC 4 Registers TRM for more information.

Parameters

Address	Value to be Written	Description
SRAM Address: 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [31:16]	Row ID	Row number to write. 0x0000 – Row 0
CPUSS_SYSARG register		
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)

26.5.4 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. This clears the page latch buffer contents after the row is programmed.

Usage Requirements: 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)

26.5.5 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in supervisory flash row 0 of each flash macro.

Usage Requirements: This API can be called only from the DAP in programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation.

Parameters

Address	Value to be Written	Description
SRAM Address: 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDD	Key2
Bits [31:16]	0xXXXX	Don’t care
CPUSS_SYSARG register		
Bits [31:0]	32’hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x000A	Erase All opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

26.5.6 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDE	Key2
Bits [31:16]	Row ID	Selects the flash row number on which the checksum operation is done. Row number – 16 bit flash row number or 0x8000 – Checksum is performed on entire flash memory
CPUSS_SYSREQ register		
Bits [15:0]	0x000B	Checksum opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	Checksum	24-bit checksum value of the selected flash region

26.5.7 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by 8. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

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	

26.5.8 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from SRAM while the write operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 266.

The non-blocking write row system call has three phases: Pre-program, Erase, Program. Pre-program is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all of the bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code from SRAM. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After the completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

Usage Requirements: Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. Also, the non-blocking write row function can be called only from 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 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

Address	Value to be Written	Description
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)

26.5.9 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 266. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation. Unlike the Non-Blocking Write Row system call, the Program system call only has a single phase. So 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 Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. Also, the non-blocking program row function can be called only from SRAM. This is because the CM0 CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDB	Key2
Bits [31:16]	Row ID	Row number to write. 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

26.5.10 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions in explained in Blocking and Non-Blocking System Calls on page 266.

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 API's 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.

Table 26-2. System Call Status Codes

Status Code (32-bit value in CPUSS_SYSARG register)	Description
F000000Ah	Checksum Zero Failed – The calculated checksum was not zero
F000000Bh	Invalid Opcode – The opcode is not a valid API opcode
F000000Ch	Key Opcode Mismatch – The opcode provided does not match key1 and key2.
F000000Eh	Invalid start address – The start address is greater than the end address provided.

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_REGREG( 0x40100000 )
#define CPUSS_SYSREQ_REG REG( 0x40100004 )
#define CPUSS_SYSARG_REG REG( 0x40100008 )
#define ROW_SIZE              128

//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)
{
   /* Call Resume API */
   // 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;
   iStatusInt ++; // Number of times the ISR has triggered
}
 __sram void NonBlockingWriteRow(void)
 {
    int iter;
    /*Load the Flash page latch with data to write*/
    //Write key1, key2, byte address,
```

```
        //and macro sel parameters to SRAM
        REG( 0x20000000 ) = 0x0000D7B6;

        //Write load size param (128 bytes) to SRAM
        REG( 0x20000004 ) = 0x0000007F;

        for(i = 0; i < ROW_SIZE/4; i += 1);
        {
            REG( 0x20000008 + i*4 ) = 0xDADADADA;
        }

        //Write the address of the key1 param to CPUSS_SYSARG reg
        CPUSS_SYSARG_REG = 0x20000000;

        //Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000004;

        /*Perform Non-Blocking Write Row on Row 200 as an example */

        //Write key1, key2, row id to SRAM
        //row id = 0xC8 -> which is row 200
        REG( 0x20000000 ) = 0x00C8DAB6;

        //Write the address of the key1 param to CPUSS_SYSARG reg
        CPUSS_SYSARG_REG = 0x20000000;

        //Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000007;

        //Execute user code until iStatusInt equals 3 to signify
        //3 SPC interrupts have happened. This should be 1 in case
        // of non-blocking program System Call
        while( iStatusInt != 0x03 )
        {
            DoOtherUserStuff();
        }

        //Get the success or failure status of System Call
        syscall_status = CPUSS_SYSARG_REG;
    }
```

In the code, the CM0 exception table is configured to be in SRAM by writing 0x01 to the CPUSS_CONFIG register. The SRAM exception table should have the vector address of the SPC interrupt as the address of the *SpcIntHandler()* function, which is also defined to be in SRAM. See the Interrupts chapter on page 37 for details on configuring the CM0 exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PSoC device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.

AND See *Boolean Algebra*.

API (Application Programming Interface) A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications.

array An array, also known as a vector or list, is one of the simplest data structures in computer programming. Arrays hold a fixed number of equally-sized data elements, generally of the same data type. Individual elements are accessed by index using a consecutive range of integers, as opposed to an associative array. Most high level programming languages have arrays as a built-in data type. Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers; for example, by a group of two integers. One- and two-dimensional arrays are the most common. Also, an array can be a group of capacitors or resistors connected in some common form.

assembly A symbolic representation of the machine language of a specific processor. Assembly language is converted to machine code by an assembler. Usually, each line of assembly code produces one machine instruction, though the use of macros is common. Assembly languages are considered low level languages; where as C is considered a high level language.

asynchronous A signal whose data is acknowledged or acted upon immediately, irrespective of any clock signal.

attenuation The decrease in intensity of a signal as a result of absorption of energy and of scattering out of the path to the detector, but not including the reduction due to geometric spreading. Attenuation is usually expressed in dB.

B

bandgap reference A stable voltage reference design that matches the positive temperature coefficient of V_T with the negative temperature coefficient of V_{BE}, to produce a zero temperature coefficient (ideally) reference.

bandwidth

1. The frequency range of a message or information processing system measured in hertz.
2. The width of the spectral region over which an amplifier (or absorber) has substantial gain (or loss); it is sometimes represented more specifically as, for example, full width at half maximum.

bias

1. A systematic deviation of a value from a reference value.
2. The amount by which the average of a set of values departs from a reference value.
3. The electrical, mechanical, magnetic, or other force (field) applied to a device to establish a reference level to operate the device.

bias current The constant low level DC current that is used to produce a stable operation in amplifiers. This current can sometimes be changed to alter the bandwidth of an amplifier.

binary The name for the base 2 numbering system. The most common numbering system is the base 10 numbering system. The base of a numbering system indicates the number of values that may exist for a particular positioning within a number for that system. For example, in base 2, binary, each position may have one of two values (0 or 1). In the base 10, decimal, numbering system, each position may have one of ten values (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

bit A single digit of a binary number. Therefore, a bit may only have a value of ‘0’ or ‘1’. A group of 8 bits is called a byte. Because the PSoC's M8CP is an 8-bit microcontroller, the PSoC devices's native data chunk size is a byte.

bit rate (BR) The number of bits occurring per unit of time in a bit stream, usually expressed in bits per second (bps).

block

1. A functional unit that performs a single function, such as an oscillator.
2. A functional unit that may be configured to perform one of several functions, such as a digital PSoC block or an analog PSoC block.

Boolean Algebra In mathematics and computer science, Boolean algebras or Boolean lattices, are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the set theoretic operations union, intersection, and complement. Boolean algebra also defines a set of theorems that describe how Boolean equations can be manipulated. For example, these theorems are used to simplify Boolean equations, which will reduce the number of logic elements needed to implement the equation.

The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR, and NOT. In describing circuits, NAND (NOT AND), NOR (NOT OR), XNOR (exclusive NOT OR), and XOR (exclusive OR) may also be used. Mathematicians often use + (for example, A+B) for OR and • for AND (for example, A*B) (in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated (for example, ~A, A_, !A).

break-before-make The elements involved go through a disconnected state entering (‘break”) before the new connected state (“make”).

broadcast net A signal that is routed throughout the microcontroller and is accessible by many blocks or systems.

buffer

1. A storage area for data that is used to compensate for a speed difference, when transferring data from one device to another. Usually refers to an area reserved for I/O operations, into which data is read, or from which data is written.
2. A portion of memory set aside to store data, often before it is sent to an external device or as it is received from an external device.
3. An amplifier used to lower the output impedance of a system.

bus

1. A named connection of nets. Bundling nets together in a bus makes it easier to route nets with similar routing patterns.
2. A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, address[7:0].
3. One or more conductors that serve as a common connection for a group of related devices.

byte A digital storage unit consisting of 8 bits.

C

C	A high level programming language.
capacitance	A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.
capture	To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file.
chaining	Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another.
checksum	The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a pre-determined value.
clear	To force a bit/register to a value of logic ‘0’.
clock	The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks.
clock generator	A circuit that is used to generate a clock signal.
CMOS	The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor.
comparator	An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements.
compiler	A program that translates a high level language, such as C, into machine language.
configuration	In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance.
configuration space	In PSoC devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’.
crowbar	A type of over-voltage protection that rapidly places a low resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value.
crystal oscillator	An oscillator in which the frequency is controlled by a piezoelectric crystal. Typically a piezoelectric crystal is less sensitive to ambient temperature than other circuit components.
cyclic redundancy check (CRC)	A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression.

D

data bus	A bi-directional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions.
data stream	A sequence of digitally encoded signals used to represent information in transmission.
data transmission	The sending of data from one place to another by means of signals over a channel.
debugger	A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory.
dead band	A period of time when neither of two or more signals are in their active state or in transition.
decimal	A base-10 numbering system, which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 (called digits) together with the decimal point and the sign symbols + (plus) and - (minus) to represent numbers.
default value	Pertaining to the pre-defined initial, original, or specific setting, condition, value, or action a system will assume, use, or take in the absence of instructions from the user.
device	The device referred to in this manual is the PSoC device, unless otherwise specified.
die	An non-packaged integrated circuit (IC), normally cut from a wafer.
digital	A signal or function, the amplitude of which is characterized by one of two discrete values: ‘0’ or ‘1’.
digital blocks	The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
digital logic	A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system.
digital-to-analog (DAC)	A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation.
direct access	The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data.
duty cycle	The relationship of a clock period *high time* to its *low time*, expressed as a percent.

E

External Reset (XRES_N) An active high signal that is driven into the PSoC device. It causes all operation of the CPU and blocks to stop and return to a pre-defined state.

F

falling edge A transition from a logic 1 to a logic 0. Also known as a negative edge.

feedback The return of a portion of the output, or processed portion of the output, of a (usually active) device to the input.

filter A device or process by which certain frequency components of a signal are attenuated.

firmware The software that is embedded in a hardware device and executed by the CPU. The software may be executed by the end user, but it may not be modified.

flag Any of various types of indicators used for identification of a condition or event (for example, a character that signals the termination of a transmission).

Flash An electrically programmable and erasable, *volatile* technology that provides users with the programmability and data storage of EPROMs, plus in-system erasability. Nonvolatile means that the data is retained when power is off.

Flash bank A group of flash ROM blocks where flash block numbers always begin with ‘0’ in an individual flash bank. A flash bank also has its own block level protection information.

Flash block The smallest amount of flash ROM space that may be programmed at one time and the smallest amount of flash space that may be protected. A flash block holds 64 bytes.

flip-flop A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until it is made to change to the other state by application of the corresponding signal.

frequency The number of cycles or events per unit of time, for a periodic function.

G

gain The ratio of output current, voltage, or power to input current, voltage, or power, respectively. Gain is usually expressed in dB.

gate

1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients.
2. One of many types of combinational logic elements having at least two inputs (for example, AND, OR, NAND, and NOR (also see *Boolean Algebra*)).

ground

1. The electrical neutral line having the same potential as the surrounding earth.
2. The negative side of DC power supply.
3. The reference point for an electrical system.
4. The conducting paths between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

hardware

A comprehensive term for all of the physical parts of a computer or embedded system, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.

hardware reset

A reset that is caused by a circuit, such as a POR, watchdog reset, or external reset. A hardware reset restores the state of the device as it was when it was first powered up. Therefore, all registers are set to the POR value as indicated in register tables throughout this document.

hexadecimal

A base 16 numeral system (often abbreviated and called hex), usually written using the symbols 0-9 and A-F. It is a useful system in computers because there is an easy mapping from four bits to a single hex digit. Thus, one can represent every byte as two consecutive hexadecimal digits. Compare the binary, hex, and decimal representations:

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

So the decimal numeral 79 whose binary representation is 0100 1111b can be written as 4Fh in hexadecimal (0x4F).

high time

The amount of time the signal has a value of ‘1’ in one period, for a periodic digital signal.

I

I^2C A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). I^2C is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. I^2C uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.

idle state A condition that exists whenever user messages are not being transmitted, but the service is immediately available for use.

impedance

1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.
2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.

input A point that accepts data, in a device, process, or channel.

input/output (I/O) A device that introduces data into or extracts data from a system.

instruction An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.

instruction mnemonics A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.

integrated circuit (IC) A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.

interface The means by which two systems or devices are connected and interact with each other.

interrupt A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

interrupt service routine (ISR) A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter

1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams.
2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

L

latency The time or delay that it takes for a signal to pass through a given circuit or network.

least significant bit (LSb) The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.

least significant byte (LSB) The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

Linear Feedback Shift Register (LFSR) A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.

load The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).

logic function A mathematical function that performs a digital operation on digital data and returns a digital value.

lookup table (LUT) A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.

low time The amount of time the signal has a value of ‘0’ in one period, for a periodic digital signal.

low voltage detect (LVD) A circuit that senses V_{DDD} and provides an interrupt to the system when V_{DDD} falls below a selected threshold.

M

M8CP An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PSoC device by interfacing to the flash, SRAM, and register space.

macro A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.

mask

1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference.
2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.

master device A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.

microcontroller An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.

mnemonic A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode A distinct method of operation for software or hardware. For example, the Digital PSoC block may be in either counter mode or timer mode.

modulation A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux)

1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

negative edge A transition from a logic 1 to a logic 0. Also known as a falling edge.

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise

1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR See *Boolean Algebra*.

NOT See *Boolean Algebra*.

O

OR See *Boolean Algebra*.

oscillator A circuit that may be crystal controlled and is used to generate a clock frequency.

output The electrical signal or signals which are produced by an analog or digital block.

P

parallel The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.

parameter Characteristics for a given block that have either been characterized or may be defined by the designer.

parameter block A location in memory where parameters for the SSC instruction are placed prior to execution.

parity A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).

path

1. The logical sequence of instructions executed by a computer.
2. The flow of an electrical signal through a circuit.

pending interrupts An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.

phase The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).

pin A terminal on a hardware component. Also called lead.

pinouts The pin number assignment: the relation between the logical inputs and outputs of the PSoC device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names.

port A group of pins, usually eight.

positive edge A transition from a logic 0 to a logic 1. Also known as a rising edge.

posted interrupts An interrupt that is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.

Power On Reset (POR) A circuit that forces the PSoC device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*.

program counter The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.

protocol A set of rules. Particularly the rules that govern networked communications.

PSoC® Cypress's Programmable System-on-Chip (PSoC®) devices.

PSoC blocks *See analog blocks* and *digital blocks*.

PSoC Creator™ The software for Cypress's next generation Programmable System-on-Chip technology.

pulse A rapid change in some characteristic of a signal (for example, phase or frequency), from a baseline value to a higher or lower value, followed by a rapid return to the baseline value.

pulse-width modulator (PWM) An output in the form of duty cycle which varies as a function of the applied measure.

R

RAM An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.

register A storage device with a specific capacity, such as a bit or byte.

reset A means of bringing a system back to a know state. See *hardware reset* and *software reset*.

resistance The resistance to the flow of electric current measured in ohms for a conductor.

revision ID A unique identifier of the PSoC device.

ripple divider An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to $2^n - 1$.

rising edge *See positive edge*.

ROM An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.

routine A block of code, called by another block of code, that may have some general or frequent use.

routing Physically connecting objects in a design according to design rules set in the reference library.

runt pulses In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling The process of converting an analog signal into a series of digital values or reversed.

schematic A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.

seed value An initial value loaded into a linear feedback shift register or random number generator.

serial

1. Pertaining to a process in which all events occur one after the other.
2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.

set To force a bit/register to a value of logic 1.

settling time The time it takes for an output signal or value to stabilize after the input has changed from one value to another.

shift The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.

shift register A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.

sign bit The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.

signal A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.

silicon ID A unique identifier of the PSoC silicon.

skew The difference in arrival time of bits transmitted at the same time, in parallel transmission.

slave device A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.

software A set of computer programs, procedures, and associated documentation about the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed.

software reset A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not PSoC blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at flash address 0x0000.

SRAM An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.

SROM An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash.

stack A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.

stack pointer A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.

state machine The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.

sticky A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The PSoC SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous

1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the PSoC device and reading this manual.

user modules Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital PSoC Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

V_{DDD} A name for a power net meaning "voltage drain." The most positive power supply signal. Usually 5 or 3.3 volts.

volatile Not guaranteed to stay the same value or level when not in scope.

V_{SS} A name for a power net meaning "voltage source." The most negative power supply signal.

W

watchdog timer A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time.

waveform The representation of a signal as a plot of amplitude versus time.

X

XOR See *Boolean Algebra*.

Index