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

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

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

# PSoC 4000 Family

# PSoC® 4 Architecture Technical Reference Manual (TRM)

Document No. 001-89309 Rev. *F

June 14, 2024

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709
www.infineon.com

## Copyrights

# Contents Overview

# Contents

# Section A: Overview

This section encompasses the following chapters:

- Introduction chapter on page 10
- Getting Started chapter on page 14
- Document Construction chapter on page 16

## Document Revision History

| Revision | Issue Date | Description of Change |
|---|---|---|
| *A | April 15, 2014 | New PSoC 4000 TRM |
| *B | May 09, 2016 | Corrected links to the register TRM. |
| *C | November 09, 2016 | No content update; sunset review |
| *D | May 30, 2017 | Updated logo and copyright information |
| *E | March 1, 2019 | Modified CapSense chapter. Fixed errors in figure captions in the Power Supply and Monitoring chapter on page 52 chapter. |
| *F | June 14, 2024 | Fixed the broken links. |

# 1. Introduction

PSoC® 4 is a programmable embedded system controller with an Arm® Cortex®-M0 CPU.CY8C4000 family is the smallest member of the PSoC 4 family of devices and is upward-compatible with larger members of PSoC 4.

PSoC 4 devices have these characteristics:

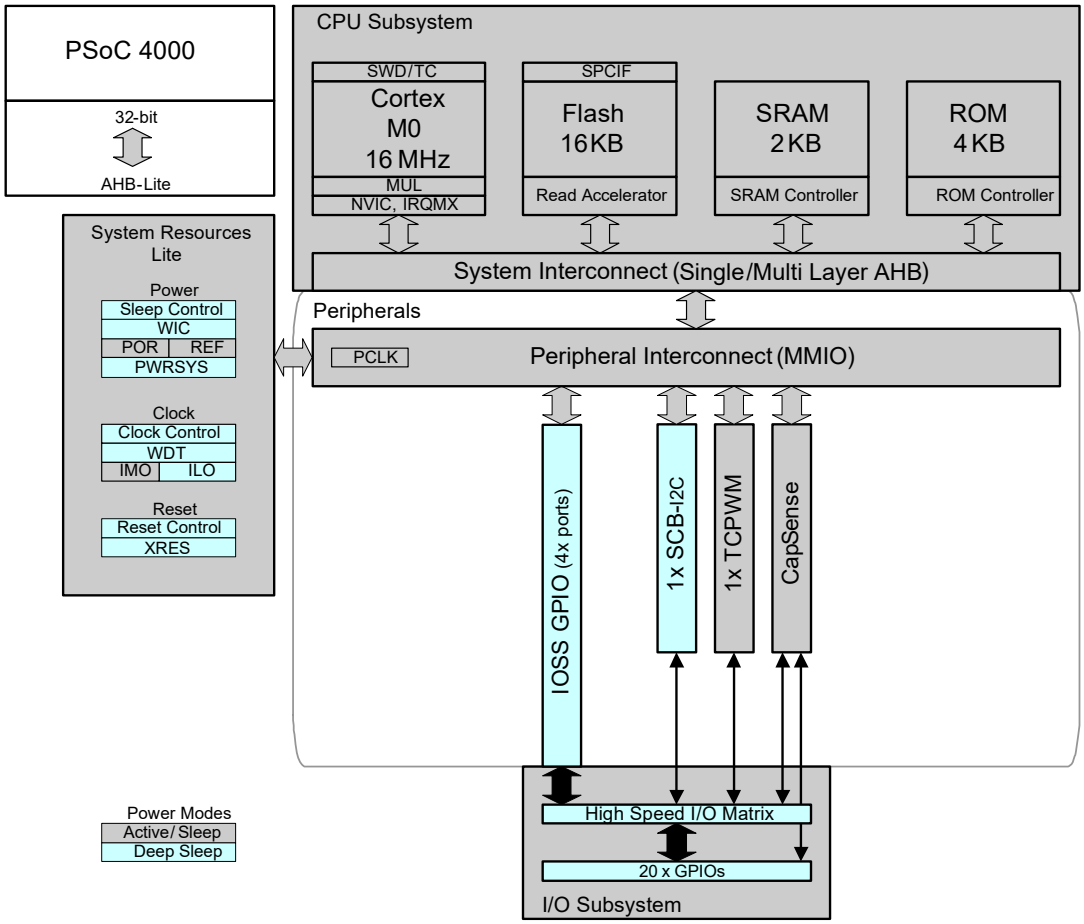
- High-performance, 32-bit single-cycle Cortex-M0 CPU core
- Capacitive touch sensing (CapSense®)
- Configurable Timer/Counter/PWM block
- Configurable $I^2C$ block with master, slave, and multi-master operating modes
- Low-power operating modes – Sleep and Deep-Sleep

This document describes each functional block of the PSoC 4000 device in detail. This information will help designers to create system-level designs.

## 1.1 Top Level Architecture

Figure 1-1 shows the major components of the PSoC 4000 architecture.

Figure 1-1. PSoC 4000 Family Block Diagram




## 1.2 Features

The PSoC 4000 family has these major components:

- 32-bit Cortex-M0 CPU with single-cycle multiply, delivering up to 14 DMIPS at 16 MHz
- Up to 16 KB flash and 2 KB SRAM
- A center-aligned pulse-width modulator (PWM) with complementary, dead-band programmable outputs
- I2C communication block with slave, master, and multi-master operating modes
- CapSense
- Low-power operating modes: Sleep and Deep-Sleep
- Programming and debugging system through serial wire debug (SWD)
- Two current sourcing/sinking DACs (IDACs)
- Comparator with 1.2 V reference
- Fully supported by PSoC Creator™ IDE tool

## 1.3 CPU System

### 1.3.1 Processor

The heart of the PSoC 4 is a 32-bit Cortex-M0 CPU core running up to 16 MHz for PSoC 4000. It is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set. This instruction set enables fully compatible binary upward migration of the code to higher performance processors such as Cortex M3 and M4.

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

### 1.3.2 Interrupt Controller

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

## 1.4 Memory

The PSoC 4 memory subsystem consists of a 16 KB flash module with a flash accelerator, 2 KB SRAM, and 4 KB supervisory ROM options. The flash accelerator improves the average access times from the flash block delivering 85 percent of single-cycle SRAM access performance. A powerful and flexible protection model allows you to selectively lock blocks of memory for read and write protection, securing sensitive information. 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. The supervisory ROM is used to store the boot and configuration routines.

## 1.5 System-Wide Resources

### 1.5.1 Clocking System

The clocking system for the PSoC 4 device consists of the internal main oscillator (IMO) and internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock.

The system clock (SYSCLK) required for the CPU system and the high-frequency clock (HFCLK) required by the peripherals can be as high as 16 MHz. These clocks are generated from the IMO.

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the device. The default IMO frequency is 24 MHz and 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  MHz to 16 MHz can be used to generate the clock derivatives for the functional blocks instead of the IMO.

### 1.5.2 Power System

The PSoC 4 operates with a single external supply in the range 1.71 V to 5.5 V.

PSoC 4 has two low-power modes – Sleep and Deep-Sleep – in addition to the default Active mode. In Active mode, the CPU runs with all the logic powered. In Sleep mode, the CPU is powered off with all other peripherals functional. In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention; the main system clock is OFF while the low-frequency clock is ON and the low-frequency peripherals are in operation.

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

### 1.5.3 GPIO

Every GPIO in PSoC 4 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

The PSoC 4 also supports CapSense capability on 17 out of 20 GPIOs. The pins are organized in a port that is 8-bit wide. A high-speed I/O matrix is used to multiplex between various signals that may connect to an I/O pin. Pin locations for fixed-function peripherals are also fixed.

## 1.6 Fixed-Function Digital

### 1.6.1 Timer/Counter/PWM Block

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

### 1.6.2 Serial Communication BlocksI2C Block

The PSoC 4 has a fixed-function I2C interface. The I2C interface can be used for general-purpose I2C communication and for tuning the CapSense component for optimized operation.

The features of the I2C block include:

- Standard $I^2C$ multi-master and slave function
- EZ function mode support with 32-byte buffer

## 1.7 Special Function Peripherals

### 1.7.1 CapSense

PSoC 4 devices have the CapSense feature, which allows you to use the capacitive properties of your fingers to toggle buttons and sliders. CapSense functionality is supported on all but three GPIO pins in PSoC 4 through a CapSense Sigma-Delta (CSD) block. The CSD also provides waterproofing capability.

#### 1.7.1.1 IDACs and Comparator

The CapSense block has two IDACs and a comparator with a 12-V reference, which can be used for general purposes, if CapSense is not used.

## 1.8 Program and Debug

PSoC 4 devices support programming and debugging features of the device via the on-chip SWD interface. The PSoC Creator IDE provides fully integrated programming and debugging support. The SWD interface is also fully compatible with industry standard third-party tools.

# 2. Getting Started

## 2.1 PSoC 4 resources

Infineon provides a wealth of data at www.infineon.com to help you select the right PSoC device and quickly and effectively integrate it into your design. The following is an abbreviated, hyperlinked list of resources for PSoC 4 MCU:

- Overview: PSoC portfolio
- Product Selectors: PSoC 4 MCU
- Application Notes cover a broad range of topics, from basic to advanced level, and include the following:
  - AN79953: Getting Started with PSoC 4
  - AN88619: PSoC 4 MCU Hardware Design Considerations
  - AN73854: PSoC Creator - Introduction to Bootloaders
  - AN89610: Arm Cortex Code Optimization
  - AN86233: PSoC 4 MCU low-power modes and power reduction techniques
  - AN57821: PSoC 3, PSoC 4, PSoC 5LP Mixed-Signal Circuit Board Layout considerations
  - AN85951: PSoC 4 and PSoC 6 MCU CapSense Design Guide
- Code Examples demonstrate product features and usage, and are also available on GitHub repositories
- Datasheets describe and provide electrical specifications for each family
- PSoC 4 MCU Programming Specification provides the information necessary to program PSoC 4 MCU non-volatile memory
- Development Tools
  - ***PSoC Creator*** is a free Windows-based IDE. It enables concurrent hardware and firmware design of PSoC 3, PSoC 4, PSoC 5LP, and PSoC 6 MCU based systems. Applications are created using schematic capture and over 150 pre-verified, production-ready peripheral Components
  - CY8CKIT-040, PSoC 4000 Pioneer Kit, is an easy-to-use and inexpensive development platform with debugging capability. This kit includes connectors for Arduino compatible shields and Digilent Pmod daughter cards.
  - MiniProg4 (CY8CKIT-005-A) and MiniProg3 (CY8CKIT-002) all-in-one development programmers and debuggers
  - PSoC 4 MCU CAD libraries provide footprint and schematic support for common tools. IBIS models are also available (CY8C4013 family, CY8C4014 family).
- Training videos are available on a wide range of topics including the PSoC MCUs

Infineon Developer Community enables connection with fellow PSoC developers around the world, 24 hours a day, 7 days a week, and hosts a dedicated PSoC 4 MCU community

## 2.2 PSoC Creator

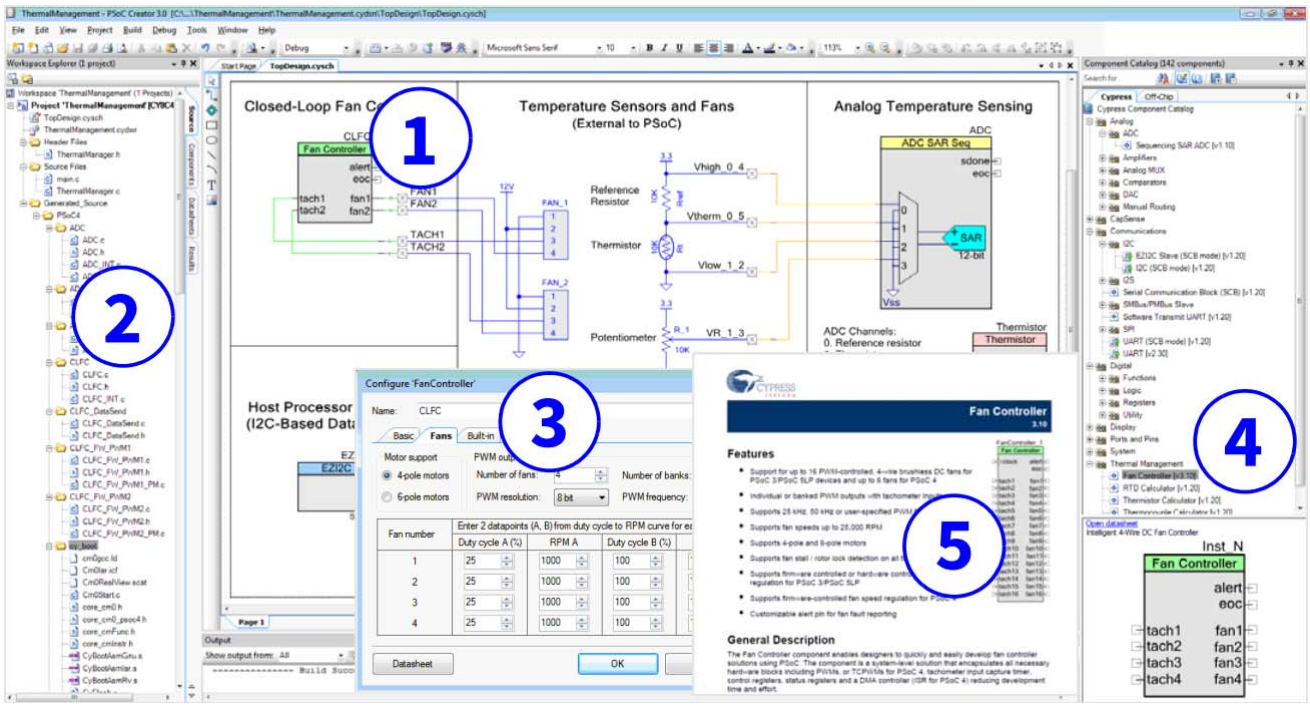
PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables you to design hardware and firmware systems concurrently, based on PSoC 4 MCU. As Figure 2-1 shows, with PSoC Creator you can:

1. Explore the library of 200+ components.
2. Drag and drop component icons to complete your hardware system design in the main design workspace.
3. Configure Components using the Component configuration tools and the Component datasheets.
4. Co-design your application firmware and hardware in the PSoC Creator IDE or build a project for a third-party IDE.

5. Prototype your solution with the PSoC 4 Pioneer kits. If a design change is needed, PSoC Creator and Components enable you to make changes on-the-fly without the need for hardware revisions.

For information on Infineon tools, refer to the documentation delivered with PSoC Creator software, and AN79953: Getting started with PSoC 4 MCU.

Figure 2-1. Multiple-sensor example project in PSoC Creator



# 3. Document Construction

This document includes the following sections:

- Section B: CPU System on page 19
- Section D: System Resources Subsystem (SRSS) on page 36
- Section E: Digital System on page 69
- Section F: Analog System on page 110
- Section G: Program and Debug on page 112

## 3.1 Major Sections

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

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

## 3.2 Documentation Conventions

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

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

### 3.2.1 Register Conventions

Register conventions are detailed in the PSoC 4000 Family: PSoC 4 Registers TRM.

### 3.2.2 Numeric Naming

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

### 3.2.3 Units of Measure

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

Table 3-1. Units of Measure

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

### 3.2.4 Acronyms

This table lists the acronyms used in this document

Table 3-2. Acronyms

| Acronym | Definition |
|---|---|
| ABUS | analog output bus |
| AC | alternating current |
| ADC | analog-to-digital converter |
| AHB | AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm data transfer bus |
| API | application programming interface |

Table 3-2. Acronyms *(continued)*

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

Table 3-2. Acronyms *(continued)*

| Acronym | Definition |
|---|---|
| IRES | initial power on reset |
| IRA | interrupt request acknowledge |
| IRQ | interrupt request |
| ISR | interrupt service routine |
| IVR | interrupt vector read |
| LCD | liquid crystal display |
| LFCLK | low-frequency clock |
| LPCOMP | low-power comparator |
| LRb | last received bit |
| LRB | last received byte |
| LSb | least significant bit |
| LSB | least significant byte |
| LUT | lookup table |
| MISO | master-in-slave-out |
| MMIO | memory mapped input/output |
| MOSI | master-out-slave-in |
| MSb | most significant bit |
| MSB | most significant byte |
| NMI | non-maskable interrupt |
| NVIC | nested vectored interrupt controller |
| PC | program counter |
| PCB | printed circuit board |
| PCH | program counter high |
| PCL | program counter low |
| PD | power down |
| PGA | programmable gain amplifier |
| PM | power management |
| PMA | PSoC memory arbiter |
| POR | power-on reset |
| PPOR | precision power-on reset |
| PRS | pseudo random sequence |
| PSoC® | Programmable System-on-Chip |
| PSRR | power supply rejection ratio |
| PSSDC | power system sleep duty cycle |
| PWM | pulse width modulator |
| RAM | random-access memory |
| RETI | return from interrupt |
| RF | radio frequency |
| ROM | read only memory |
| RMS | root mean square |
| RW | read/write |
| SAR | successive approximation register |
| SC | switched capacitor |
| SCB | serial communication block |
| SIE | serial interface engine |
| SIO | special I/O |
| SE0 | single-ended zero |

Table 3-2. Acronyms *(continued)*

| Acronym | Definition |
|---|---|
| SNR | signal-to-noise ratio |
| SOF | start of frame |
| SOI | start of instruction |
| SP | stack pointer |
| SPD | sequential phase detector |
| SPI | serial peripheral interconnect |
| SPIM | serial peripheral interconnect master |
| SPIS | serial peripheral interconnect slave |
| SRAM | static random-access memory |
| SROM | supervisory read only memory |
| SSADC | single slope ADC |
| SSC | supervisory system call |
| SYSCLK | system clock |
| SWD | single wire debug |
| TC | terminal count |
| TCPWM | timer, counter, PWM |
| TD | transaction descriptors |
| UART | universal asynchronous receiver/transmitter |
| UDB | universal digital block |
| USB | universal serial bus |
| USBIO | USB I/O |
| WCO | watch crystal oscillator |
| WDT | watchdog timer |
| WDR | watchdog reset |
| XRES | external reset |
| XRES_N | external reset, active low |

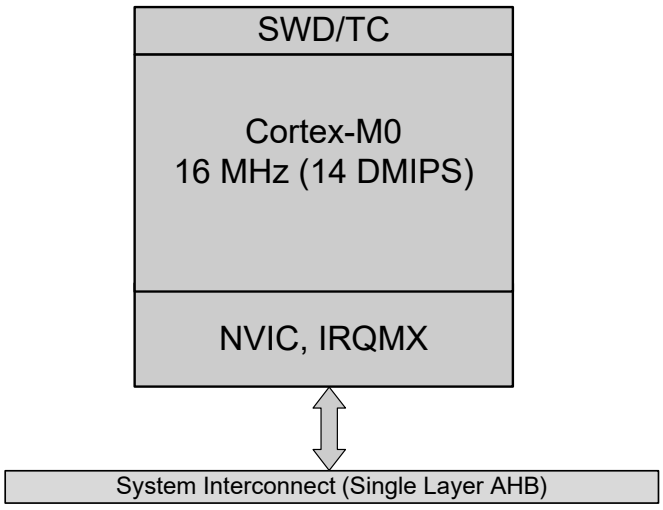
# Section B: CPU System

This section encompasses the following chapters:

- Cortex-M0 CPU chapter on page 20
- Interrupts chapter on page 25

## 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 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

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

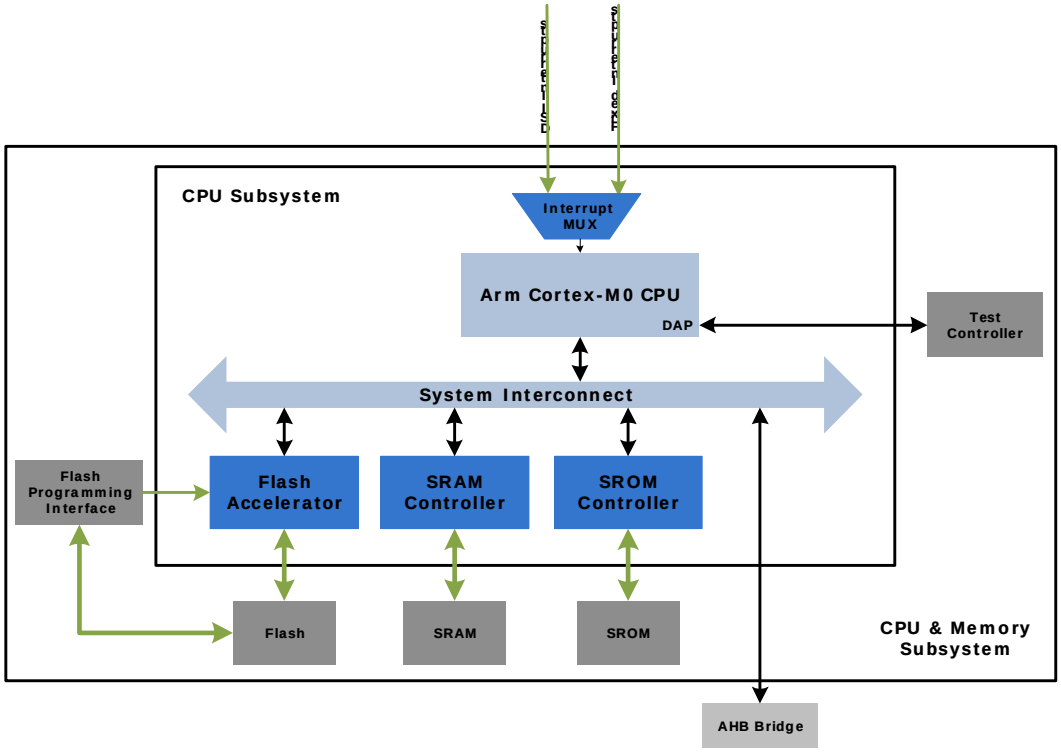
## 4.1 Features

The PSoC 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 the Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

## 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 (see “Operating Modes” on page 23). It has a single-cycle 32-bit multiplication instruction.

## 4.4 Address Map

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

Table 4-1. Cortex-M0 Address Map

| Address Range | Name | Use |
|---|---|---|
| 0x00000000 - 0x1FFFFFFF | Code | Program code region. You can also place data here. Includes the exception vector table, which starts at address 0. |
| 0x20000000 - 0x3FFFFFFF | SRAM | Data region. You can also execute code from this region. |
| 0x40000000 - 0x5FFFFFFF | Peripheral | All peripheral registers. You cannot execute code from this region. |
| 0x60000000 - 0xDFFFFFFF | | Not used. |
| 0xE0000000 - 0xE00FFFFF | PPB | Peripheral registers within the CPU core. |
| 0xE0100000 - 0xFFFFFFFF | Device | PSoC 4 implementation-specific. |

## 4.5 Registers

The Cortex-M0 has 16 32-bit registers, as Table 4-2 shows:

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

Table 4-2. Cortex-M0 Registers

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

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

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

Table 4-3. Cortex-M0 PSR Bit Assignments

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

Table 4-3. Cortex-M0 PSR Bit Assignments

| Bit | PSR Register | Name | Usage |
|---|---|---|---|
| 27 – 25 | – | – | Reserved |
| 24 | EPSR | T | Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception. |
| 23 – 6 | – | – | Reserved |
| 5 – 0 | IPSR | N/A | Exception number of current ISR:<br>0 = thread mode<br>1 = reserved<br>2 = NMI<br>3 = HardFault<br>4 – 10 = reserved<br>11 = SVCall<br>12, 13 = reserved<br>14 = PendSV<br>15 = SysTick<br>16 = IRQ0<br>…<br>24 = IRQ8 |

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 25 for a list of exceptions.

## 4.6 Operating Modes

The Cortex-M0 processor supports two operating modes:

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

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

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

## 4.7 Instruction Set

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

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

Table 4-4. Thumb Instruction Set

| Mnemonic | Brief Description |
|---|---|
| ADCS | Add with carry |
| ADD{S}[a] | Add |
| ADR | PC-relative address to register |
| ANDS | Bit wise AND |
| ASRS | Arithmetic shift right |
| B{cc} | Branch {conditionally} |
| BICS | Bit clear |
| BKPT | Breakpoint |
| BL | Branch with link |
| BLX | Branch indirect with link |
| BX | Branch indirect |
| CMN | Compare negative |
| CMP | Compare |
| CPSID | Change processor state, disable interrupts |
| CPSIE | Change processor state, enable interrupts |
| DMB | Data memory barrier |
| DSB | Data synchronization barrier |
| EORS | Exclusive OR |
| ISB | Instruction synchronization barrier |
| LDM | Load multiple registers, increment after |
| LDR | Load register from PC-relative address |
| LDRB | Load register with word |
| LDRH | Load register with half-word |
| LDRSB | Load register with signed byte |
| LDRSH | Load register with signed half-word |
| LSLS | Logical shift left |
| LSRS | Logical shift right |
| MOV{S}[a] | Move |
| MRS | Move to general register from special register |
| MSR | Move to special register from general register |
| MULS | Multiply, 32-bit result |
| MVNS | Bit wise NOT |
| NOP | No operation |
| ORRS | Logical OR |
| POP | Pop registers from stack |
| PUSH | Push registers onto stack |
| REV | Byte-reverse word |
| REV16 | Byte-reverse packed half-words |
| REVSH | Byte-reverse signed half-word |
| RORS | Rotate right |
| RSBS | Reverse subtract |
| SBCS | Subtract with carry |

Table 4-4. Thumb Instruction Set

| Mnemonic | Brief Description |
|---|---|
| SEV | Send event |
| STM | Store multiple registers, increment after |
| STR | Store register as word |
| STRB | Store register as byte |
| STRH | Store register as half-word |
| SUB{S}[a] | Subtract |
| SVC | Supervisor call |
| SXTB | Sign extend byte |
| SXTH | Sign extend half-word |
| TST | Logical AND-based test |
| UXTB | Zero extend a byte |
| UXTH | Zero extend a half-word |
| WFE | Wait for event |
| WFI | Wait for interrupt |

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

### 4.7.1 Address Alignment

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

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

### 4.7.2 Memory Endianness

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

## 4.8 Systick Timer

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

## 4.9 Debug

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

# 5. 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, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or interrupt service routine (ISR) being executed by the CPU. The device provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

## 5.1 Features

PSoC 4 supports the following interrupt features:

- Supports 9 interrupts
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

## 5.2 How It Works

Figure 5-1. PSoC 4 Interrupts Block Diagram

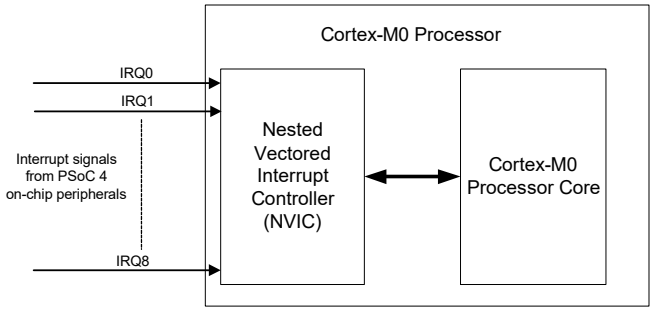



Figure 5-1 shows the interaction between interrupt signals and the Cortex-M0 CPU. PSoC 4 has nine interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are 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.

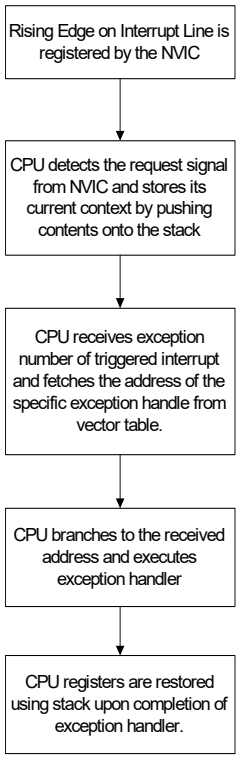
# 5.3 Interrupts and Exceptions - Operation

## 5.3.1 Interrupt/Exception Handling

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

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

Figure 5-2. Interrupt Handling When Triggered




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

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

## 5.3.2 Level and Pulse Interrupts

NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IRQ8). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 5-3. Level Interrupts

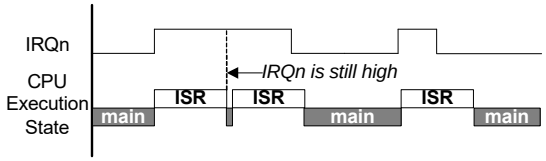



Figure 5-4. Pulse Interrupts

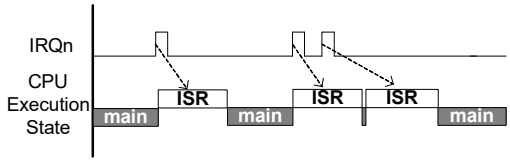



Figure 5-3 and Figure 5-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

1. On a rising edge event of the interrupt signal, the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the interrupt requests have not yet been serviced by the CPU.
2. The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
3. When the ISR is being executed by the CPU, one or more rising edges of the interrupt signal are logged as a single pending request. The pending interrupt is serviced again after the current ISR execution is complete (see Figure 5-4 for pulse interrupts).
4. If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. Figure 5-3 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. The CPU fetches the appropriate address based on the exception number.

Table 5-1. Exception Vector Table

| Exception Number | Exception | Exception Priority | Vector Address |
|---|---|---|---|
| – | Initial Stack Pointer Value | Not applicable (NA) | Base_Address - 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM) |
| 1 | Reset | –3, the highest priority | Base_Address + 0x04 |
| 2 | Non Maskable Interrupt (NMI) | –2 | Base_Address + 0x08 |
| 3 | HardFault | –1 | Base_Address + 0x0C |
| 4-10 | Reserved | NA | Base_Address + 0x10 to Base_Address + 0x28 |
| 11 | Supervisory Call (SVCall) | Configurable (0 - 3) | Base_Address + 0x2C |
| 12-13 | Reserved | NA | Base_Address + 0x30 to Base_Address + 0x34 |
| 14 | PendSupervisory (PendSV) | Configurable (0 - 3) | Base_Address + 0x38 |
| 15 | System Timer (SysTick) | Configurable (0 - 3) | Base_Address + 0x3C |
| 16 | External Interrupt(IRQ0) | Configurable (0 - 3) | Base_Address + 0x40 |
| … | … | Configurable (0 - 3) | … |
| 24 | External Interrupt(IRQ8) | Configurable (0 - 3) | Base_Address + 0x52 |

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 (base address of 0x00000000) or SRAM (base address of 0x20000000). This configuration is done by writing to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. When the VECT_IN_RAM bit field is '1', CPU fetches exception handler addresses from the SRAM vector table location. When this bit field is '0' (reset state), the vector table in flash memory is used for exception address fetches. You must set the VECT_IN_RAM bit field as part of the device boot code to configure the vector table to be in SRAM. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

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

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

The exception sources (exception numbers 1 to 15) are explained in 5.4 Exception Sources. The exceptions marked as Reserved in Table 5-1 are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 24) 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 always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of supervisory read-only memory (SROM). The boot code and other data in SROM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in Table 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 the SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted.

### 5.4.2 Non-Maskable Interrupt (NMI) Exception

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

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

### 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 and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0 in PSoC 4 uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the Chip Operational Modes chapter on page 57 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in the device. The application developer must define the SVCall exception handler according to the end application requirements.

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

### 5.4.5 PendSV Exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, 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 bit fields PRI_14[23:22] of the System Handler Priority Register 3 (CM0_SHPR3). See the Armv6-M Architecture Reference Manual for more details.

### 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 timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_15[31:30] of the System Handler Priority Register 3 (SHPR3). The SysTick exception can always be generated in software at any instant by writing a one to the PENDSTSETb bit in the Interrupt Control State Register, 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 nine interrupts (IRQ0 to IRQ8 or exception numbers 16 – 24) from peripherals. The source of each interrupt is listed in . PSoC 4 provides flexible sourcing options for each interrupt line. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM serial communication block, CSD block, and interrupts from ports. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. These interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously.

See the I/O System chapter on page 37 for details on GPIO interrupts.

Table 5-2. List of PSoC 4 Interrupt Sources

| Interrupt | Cortex-M0 Exception No. | Interrupt Source |
|---|---|---|
| NMI (see "Exception Sources" on page 27) | 2 | – |
| IRQ0 | 16 | GPIO Interrupt - Port 0 |
| IRQ1 | 17 | GPIO Interrupt - Port 1 |
| IRQ2 | 18 | GPIO Interrupt - Port 2 |
| IRQ3 | 19 | GPIO Interrupt - Port 3 |
| IRQ4 | 20 | WDT (Watchdog timer) or Temp |
| IRQ5 | 21 | SCB (Serial Communication Block) |
| IRQ6 | 22 | SPC (System Performance Controller) |
| IRQ7 | 23 | CSD (CapSense block counter overflow interrupt) |
| IRQ8 | 24 | TCPWM0 (Timer/Counter/PWM 0) |

## 5.6 Exception Priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. PSoC 4 provides flexibility in choosing priority values for different exceptions. All exceptions other than Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In PSoC 4, lower priority numbers represent higher priorities. This means that the Reset, NMI, and HardFault exceptions have the highest priorities. The other exceptions can be assigned a configurable priority level between 0 and 3.

PSoC 4 supports nested exceptions in which a higher priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as 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 to 15 are explained in "Exception Sources" on page 27.

The priority of the nine interrupts (IRQ0 to IRQ8) can be configured by writing to the Interrupt Priority registers (CM0_IPR). This is a group of four 32-bit registers with each register storing the priority values of four interrupts, as given in Table 5-3. The other bit fields in the register are not used.

Table 5-3. Interrupt Priority Register Bit Definitions

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

## 5.7 Enabling and Disabling Interrupts

The NVIC provides registers to individually enable and disable the nine 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 are 32-bit wide registers and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 5-4 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 5-4. 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 interrupts IRQ0 to IRQ8. These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in “Exception Sources” on page 27.

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

## 5.8 Exception States

Each exception can be in one of the following states.

Table 5-5. Exception States

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

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

- The VECTACTIVE bits ([8:0]) in the CM0_ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ8) is in a pending state.

### 5.8.1 Pending Exceptions

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

The NVIC allows software pending of the nine 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 are 32-bit wide registers and each bit corresponds to the same numbered interrupt.

Table 5-6 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 5-6. 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 ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0_ISER register.

Note that the CM0_ISPR and CM0_ICPR registers are used only for the nine peripheral interrupts (exception numbers 16–47). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in “Exception Sources” on page 27.

## 5.9 Stack Usage for Exceptions

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

When the CPU is in thread mode and an exception request comes, the CPU uses the stack pointer defined in the control register to store the general-purpose register contents. After the stack push operations, the CPU enters handler mode to execute the exception handler. When another higher priority exception occurs while executing the current exception, the MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the Cortex-M0 CPU chapter on page 20 for details.

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

## 5.10 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 ISR of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0 CPU, triggers the transition into Sleep and Deep-Sleep modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 58. Chip low-power modes have two categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available only in the Active and Deep-Sleep modes (watchdog timer interrupt, I2C interrupts, and GPIO interrupts)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

## 5.11 Exceptions – 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 VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. This register write is done as part of device initialization code.

   It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents. PSoC Creator IDE uses the vector table in SRAM by default.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   b. Define the exception handler function and write the address of the function to the exception vector table. Table 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 29.
   d. Enable the exception, as explained in "Enabling and Disabling Interrupts" on page 30.

## 5.12 Registers

Table 5-7. List of Registers

| Register Name | Description |
|---|---|
| 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 Register |
| CPUSS_CONFIG | CPU Subsystem Configuration Register |
| CPUSS_SYSREQ | System Request Register |

## 5.13 Associated Documents

- Armv6-M Architecture Reference Manual – This document explains the Arm Cortex-M0 architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

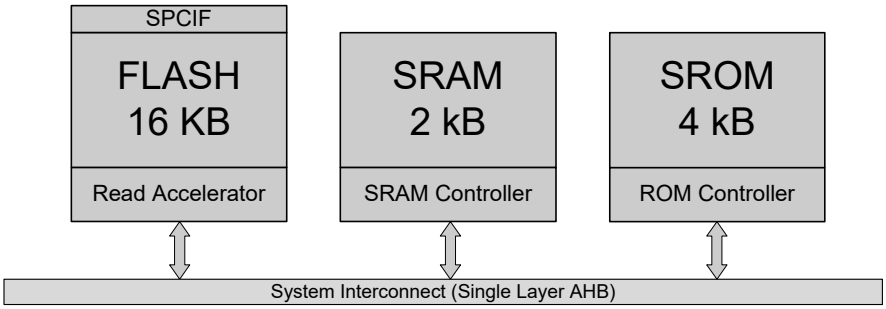
# Section C: Memory System

This section presents the following chapter:

- Memory Map chapter on page 34

## Top Level Architecture

Memory System Block Diagram



# 6. Memory Map

All PSoC® 4 memory (flash, SRAM, and 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:

- 16K bytes flash, 2K 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 fixed-function I2C block. For more information, see the Cortex-M0 CPU chapter on page 20.

## 6.2 How It Works

The PSoC 4 memory map is detailed in the following tables. For additional information, refer to the PSoC 4000 Family: PSoC 4 Registers 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 - 0x00003FFF | 16 KB flash |
| 0x0FFFF000 - 0x10000000 | 4 KB supervisory flash |
| 0x20000000 - 0x200007FF | 2 KB SRAM |
| 0x40100000 - 0x4011FFFF | CPU subsystem registers |
| 0x40020000 - 0x40023FFF | I/O port control (high-speed I/O matrix) registers |
| 0x40040000- -0x40043FFF | I/O port registers |
| 0x40050000- -0x4005FFFF | TCPWM registers |
| 0x40060000- -0x4006FFFF | Fixed-function I2C registers |
| 0x40080000- -0x4008FFFF | CapSense registers |
| 0x40030000- -0x4003FFFF | Power, clock, reset control registers |
| 0xE0000000 - 0xE00FFFFF | Cortex-M0 PPB registers |
| 0xF0000000 - 0xF0000FFF | CoreSight ROM |

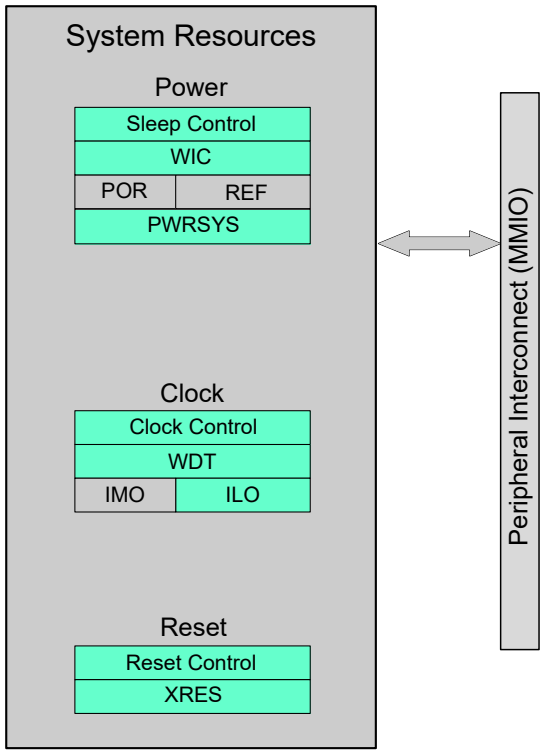
# Section D: System Resources Subsystem (SRSS)

This section encompasses the following chapters:

- I/O System chapter on page 37
- Clocking System chapter on page 46
- Power Supply and Monitoring chapter on page 52
- Chip Operational Modes chapter on page 57
- Power Modes chapter on page 58
- Watchdog Timer chapter on page 62
- Reset System chapter on page 65
- Device Security chapter on page 67

## Top Level Architecture

System-Wide Resources Block Diagram



# 7. I/O System

This chapter explains the PSoC® 4 I/O system, its features, architecture, operating modes, and interrupts. The GPIO pins in PSoC 4 are grouped into ports; a port can have a maximum of eight GPIOs. PSoC 4000 family has a maximum of 20 GPIOs arranged in four ports.

## 7.1 Features

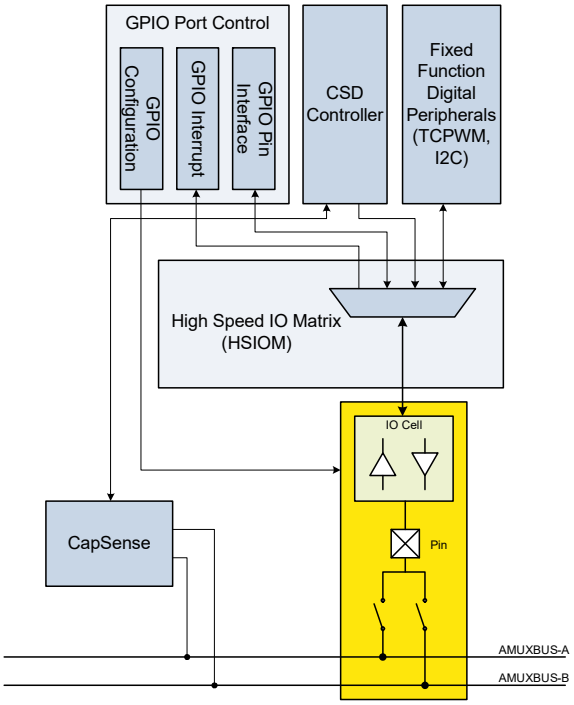
The PSoC 4 GPIOs have these features:

- Analog and digital input and output capabilities
- Eight drive strength modes
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Hold mode for latching previous state (used for retaining I/O state in Deep-Sleep mode)
- Selectable CMOS and low-voltage LVTTL input buffer mode
- CapSense support

## 7.2 GPIO Interface Overview

PSoC 4 is equipped with analog and digital peripherals. Figure 7-1 shows an overview of the routing between the peripherals and pins.

Figure 7-1. GPIO Interface Overview

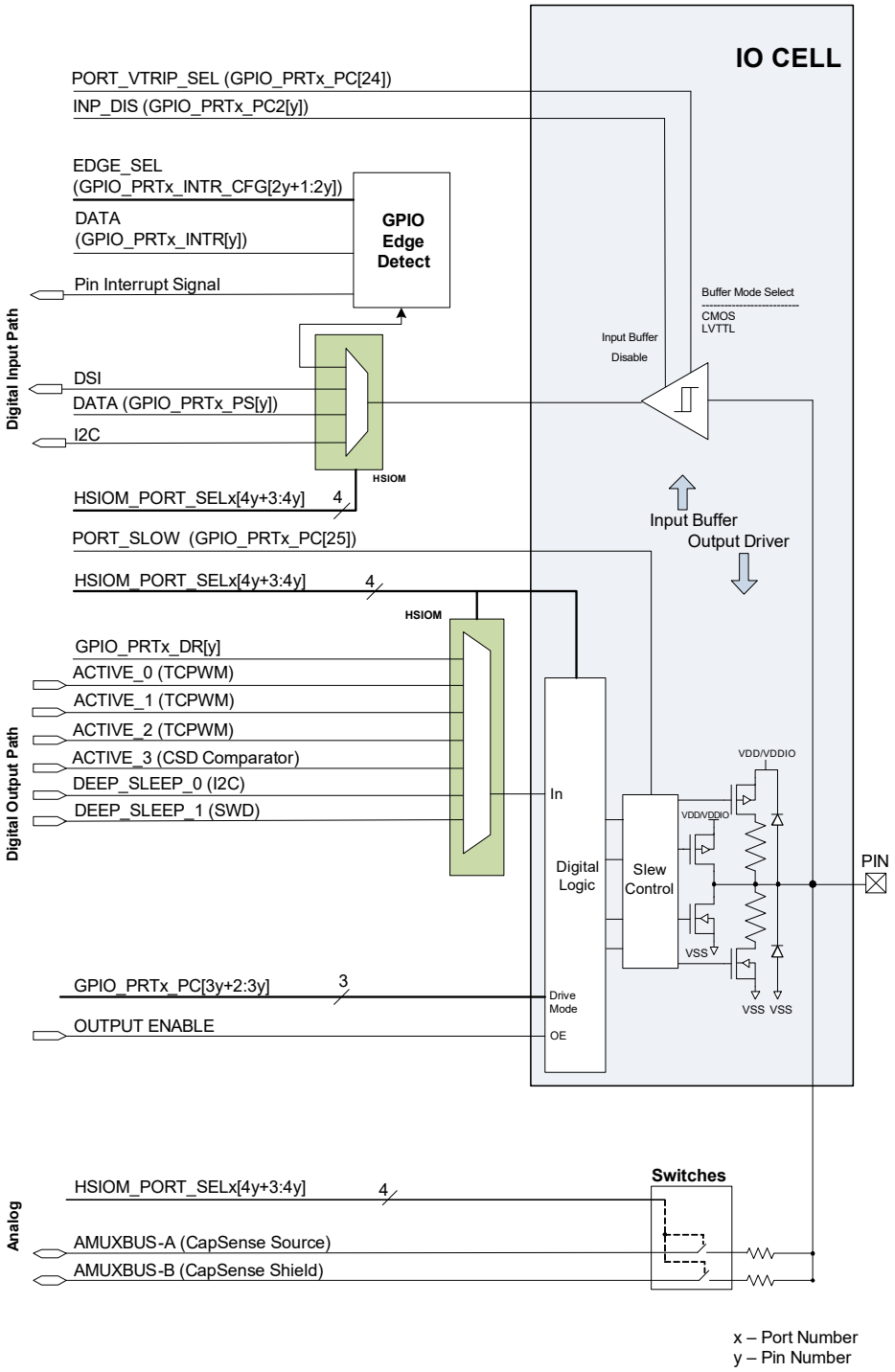



GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high input impedance and a driver for the digital output signals. The digital peripherals connect to the I/O cells via the high-speed I/O matrix (HSIOM). HSIOM contains multiplexers to connect between a peripheral selected by the user and the pin. The CapSense block is connected to the GPIO pins through the AMUX buses.

## 7.3 I/O Cell Architecture

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

Figure 7-2. I/O Cell Architecture in PSoC 4000



### 7.3.1 Digital Input Buffer

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

- CMOS
- LVTTL

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

Table 7-1. Input Buffer Modes

| PORT_VTRIP_SEL | Input Buffer Mode |
|---|---|
| 0b | CMOS |
| 1b | LVTTL |

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

### 7.3.2 Digital Output Driver

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

PSoC 4000 has a dedicated I/O supply voltage pin VDDIO in the 16-QFN package; in the remaining devices, I/Os are driven with the VDD supply. Each GPIO pin has ESD diodes to clamp the pin voltage to the I/O supply source. Ensure that the voltage at the pin does not exceed the I/O supply voltage $V_{DDIO}/V_{DD}$ and drop below $V_{SS}$. For the absolute maximum and minimum GPIO voltage, see the device datasheet. The digital output driver can be enabled and disabled using the DSI signal from the peripheral or data register (GPIO_PRTx_DR) associated with the output pin. See 7.4 High-Speed I/O Matrix to know about the peripheral source selection for the data and to enable or disable control source selection.

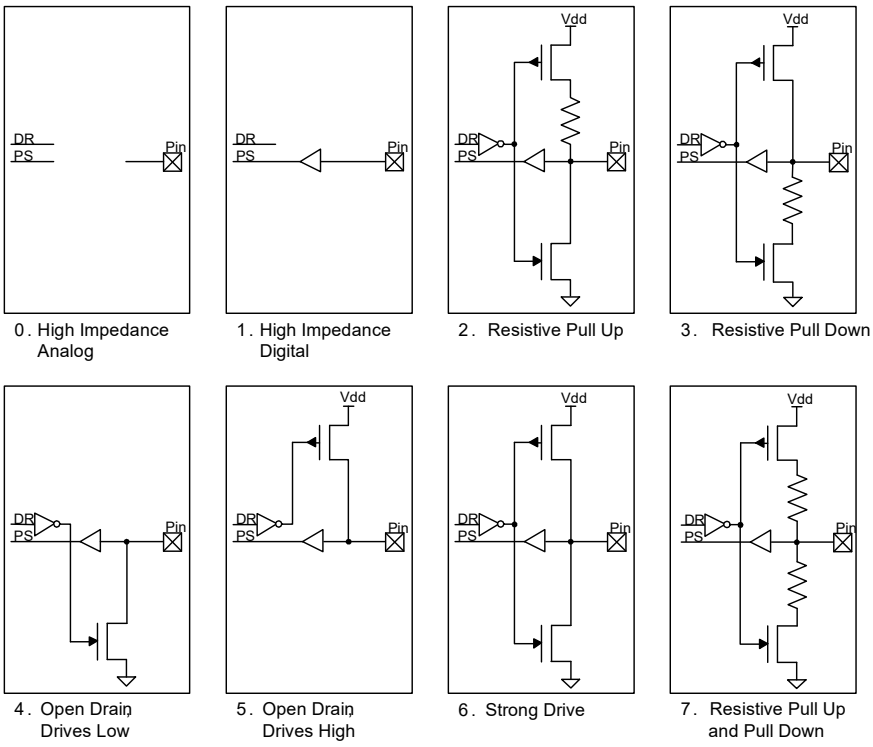
#### *7.3.2.1 Drive Modes*

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

Table 7-2. Drive Mode Settings

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

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




- High-Impedance Analog

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

- High-Impedance Digital

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

- Resistive Pull-Up or Resistive Pull-Down

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

- Open Drain Drives High and Open Drain Drives Low

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

- Strong Drive

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

- Resistive Pull-Up and Resistive Pull-Down

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

#### *7.3.2.2 Slew Rate Control*

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

## 7.4 High-Speed I/O Matrix

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

Table 7-3. PSoC 4000 HSIOM Port Settings

| HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number) | | | |
|---|---|---|---|
| **Bits** | **Name (SEL 'y')** | **Value** | **Description (Selects pin 'y' source (0 ≤ y ≤ 7)** |
| 4y+3 : 4y | DR | 0 | Pin is firmware-controlled GPIO. |
| | 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 GPIO pre-charging of tank capacitors. |
| | ACTIVE_0 | 8 | Pin-specific Active source # 0 (TCPWM, EXT_CLK). |
| | ACTIVE_1 | 9 | Pin-specific Active source #1 (TCPWM). |
| | ACTIVE_2 | 10 | Pin-specific Active source #2 (TCPWM). |
| | ACTIVE_3 | 11 | Pin-specific Active source #3 (CSD comparator). |
| | DEEP_SLEEP_0 | 14 | Pin-specific Deep-Sleep source #0 (SCB - $I^2C$). |
| | DEEP_SLEEP_1 | 15 | Pin-specific Deep-Sleep source #1 (SWD). |

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

## 7.5 I/O State on Power Up

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

## 7.6 Behavior in Low-Power Modes

Table 7-4 shows the status of GPIOs in low-power modes.

Table 7-4. GPIO in Low-Power Modes

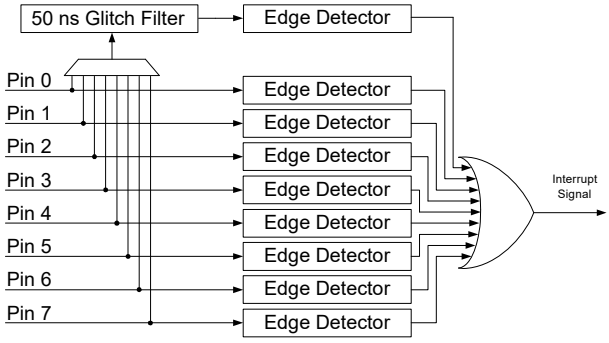
| Low-Power Mode | Status |
|---|---|
| Sleep | ■ GPIOs are active and can be driven by peripherals such as CapSense, TCPWM, and $I^2C$, which can work in sleep mode.<br>■ Input buffers are active; thus an interrupt on any I/O can be used to wake up the CPU. |
| Deep-Sleep | ■ GPIO output pin states are latched and remain in the frozen state, except the $I^2C$ pins. $I^2C$ block can work in the deep-sleep mode and can wake up the CPU on address match event.<br>■ Input buffers are also active in this mode; pin interrupts are functional. |

## 7.7 Interrupt

In the PSoC 4 device, all the port pins have the capability to generate interrupts. As shown in Figure 7-2, the pin signal is routed to the interrupt controller through the GPIO Edge Detect block.

Figure 7-4 shows the GPIO Edge Detect block architecture.

Figure 7-4. GPIO Edge Detect Block Architecture




An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, and both edges without reconfiguration. The edge detector is configured by writing into the EDGE_SEL bits of the Port Interrupt Configuration register, GPIO_PRTx_INTR_CFG, as shown in Table 7-5.

Table 7-5. Edge Detector Configuration

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

Besides the pins, edge detector is also present at the glitch filter output. This filter can be used on one of the pins of a port. The pin is selected by writing to the FLT_SEL field of the GPIO_PRTx_INTR_CFG register as shown in Table 7-6.

Table 7-6. Glitch filter Input Selection

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

The edge detector outputs of a port are ORed together and then routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port. On a pin interrupt, it is required to know which pin caused an interrupt. This is done by reading the Port Interrupt Status register, GPIO_PRTx_INTR. This register not only includes the information on which pin triggered the interrupt, it also

includes the pin status; it allows the CPU to read both information in a single read operation. This register has one more important use – to clear the interrupt. Writing ‘1’ to the corresponding status bit clears the pin interrupt. It is important to clear the interrupt status bit; otherwise, the interrupt will occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. Also, note that when the Port Interrupt Control Status register is read when an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code. Table 7-7 shows the Port Interrupt Status register bit fields.

Table 7-7. Port Interrupt Status Register

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

The edge detector block output is routed to the Interrupt Source Multiplexer shown in Figure 5-3 on page 26, which gives an option of Level and Rising Edge detect. If the Level option is selected, an interrupt is triggered repeatedly as long as the Port Interrupt Status register bit is set. If the Rising Edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the Edge Detect block is used.

## 7.8 Peripheral Connections

### 7.8.1 Firmware Controlled GPIO

See Table 7-3 to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

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

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

### 7.8.2 CapSense

The pins that support CSD can be configured as CapSense widgets such as buttons, slider elements, touchpad elements, or proximity sensors. CapSense also requires external tank capacitors and shield lines. Table 7-8 shows the GPIO and HSIOM settings required for CapSense. See the CapSense chapter on page 111 for more information.

Table 7-8. CapSense Settings

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

### 7.8.3 Timer, Counter, and Pulse Width Modulator (TCPWM) Block

TCPWM has dedicated connections to the pin. See the device datasheet for details on these dedicated pins of PSoC 4. Note that when the TCPWM block inputs such as start and stop are taken from the pins, the drive mode can be only high-z digital because the TCPWM block disables the output buffer at the input pins.

## 7.9 Registers

Table 7-9. I/O Registers

| Name | Description |
|---|---|
| GPIO_PRTx_DR | Port Output Data Register |
| GPIO_PRTx_DR_SET | Port Output Data Set Register |
| GPIO_PRTx_DR_CLR | Port Output Data Clear Register |
| GPIO_PRTx_DR_INV | Port Output Data Inverting Register |
| GPIO_PRTx_PS | Port Pin State Register - Reads the logical pin state of I/O |
| GPIO_PRTx_PC | Port Configuration Register - Configures the output drive mode, input threshold, and slew rate |
| GPIO_PRTx_PC2 | Port Secondary Configuration Register - Configures the input buffer of I/O pin |
| GPIO_PRTx_INTR_CFG | Port Interrupt Configuration Register |
| GPIO_PRTx_INTR | Port Interrupt Status Register |
| HSIOM_PORT_SELx | HSIOM Port Selection Register |

**Note** The 'x' in the GPIO register name denotes the port number. For example, GPIO_PTR1_DR is the Port 1 output data register.

# 8. Clocking System

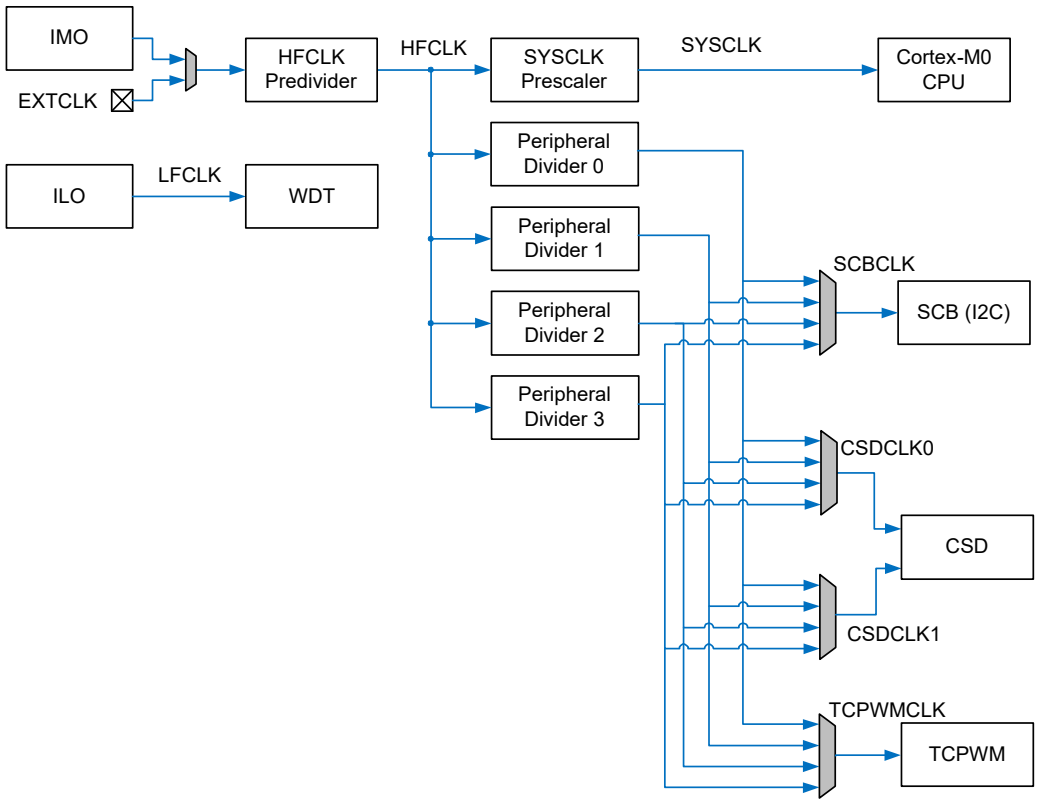
The PSoC® 4 clock system includes these clock resources:

- Two internal clock sources:
  - 24–48 MHz internal main oscillator (IMO) with ±2 percent accuracy across all frequencies with trim
  - 40-kHz internal low-speed oscillator (ILO) (can be calibrated using the IMO)
- External clock (EXTCLK) generated using a signal from an I/O pin
- High-frequency clock (HFCLK) of up to 48 MHz, selected from IMO or external clock
  - Dedicated prescaler for HFCLK
- Low-frequency clock (LFCLK) sourced by ILO
- Dedicated prescaler for system clock (SYSCLK) of up to 16 MHz sourced by HFCLK
- Four peripheral clocks, each with a 16-bit divider

## 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 IMO, EXTCLK, and ILO, as shown in Figure 8-1. The HFCLK mux selects the HFCLK source from the EXTCLK or the IMO. The HFCLK predivider divides the HFCLK input. The ILO sources the LFCLK.

## 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 frequencies spanning24-48 MHz in 4-MHz increments. Frequencies are selected by setting the frequency in the CLK_IMO_TRIM2 register and setting the IMO trim in the CLK_IMO_TRIM1 register The frequency setting in CLK_IMO_TRIM2 determines the IMO frequency output. Table 8-1 provides the setting corresponding to the IMO frequency output. In addition to setting the frequency in CLK_IMO_TRIM2, the user needs to load corresponding trim values in the CLK_IMO_TRIM1. Frequency selection follows an algorithm to ensure no intermediate state is programmed to a value higher than 48 MHz. Each PSoC device has IMO trim settings determined during manufacturing to meet datasheet specifications; the trim is stored in manufacturing configuration data in SFLASH. There are TRIM values corresponding to the frequency selected by the user. The TRIM values from SFLASH are loaded in the corresponding trim registers – CLK_IMO_TRIM1. These values may be loaded at startup to achieve the desired configuration. Firmware can retrieve these trim values and reconfigure the device to change the frequency at run-time.

To configure the IMO frequency, follow this algorithm:

- If ((new_freq ≥ 43 MHz) and (old_freq ≥ 43 MHz)),
  Change CLK_IMO_TRIM2 to a lower frequency such as 24 MHz
  Apply CLK_IMO_TRIM1, PWR_BG_TRIM4, and PWR_BG_TRIM5 for the new_freq
  Wait ≥ 5 µs
  Change CLK_IMO_TRIM2 to new_freq
- else if (new_freq > old_freq),
  Apply CLK_IMO_TRIM1, PWR_BG_TRIM4, and PWR_BG_TRIM5 for new_freq
  Wait ≥ 5 µs
  Change CLK_IMO_TRIM2 to new_freq
- else
  Change CLK_IMO_TRIM2 to new_freq
  Wait ≥ 5 cycles
  Apply CLK_IMO_TRIM1, PWR_BG_TRIM4, and PWR_BG_TRIM5 for new_freq

Table 8-1. IMO Frequency Configuration

| CLK_IMO_TRIM2 | | | | | | Frequency in MHz |
|---|---|---|---|---|---|---|
| Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| 0 | 0 | 0 | 1 | 1 | 0 | 6 |
| 0 | 0 | 0 | 1 | 1 | 1 | 7 |
| 0 | 0 | 1 | 0 | 0 | 0 | 8 |
| 0 | 0 | 1 | 0 | 0 | 1 | 9 |
| 0 | 0 | 1 | 0 | 1 | 0 | 10 |
| 0 | 0 | 1 | 0 | 1 | 1 | 11 |
| 0 | 0 | 1 | 1 | 0 | 0 | 12 |
| 0 | 0 | 1 | 1 | 1 | 0 | 13 |
| 0 | 0 | 1 | 1 | 1 | 1 | 14 |
| 0 | 1 | 0 | 0 | 0 | 0 | 15 |
| 0 | 1 | 0 | 0 | 0 | 1 | 16 |
| 0 | 1 | 0 | 0 | 1 | 0 | 17 |
| 0 | 1 | 0 | 0 | 1 | 1 | 18 |
| 0 | 1 | 0 | 1 | 0 | 0 | 19 |
| 0 | 1 | 0 | 1 | 0 | 1 | 20 |
| 0 | 1 | 0 | 1 | 1 | 0 | 21 |
| 0 | 1 | 0 | 1 | 1 | 1 | 22 |
| 0 | 1 | 1 | 0 | 0 | 0 | 23 |
| 0 | 1 | 1 | 0 | 0 | 1 | 24 |
| 0 | 1 | 1 | 0 | 1 | 1 | 25 |
| 0 | 1 | 1 | 1 | 0 | 0 | 26 |
| 0 | 1 | 1 | 1 | 0 | 1 | 27 |
| 0 | 1 | 1 | 1 | 1 | 0 | 28 |
| 0 | 1 | 1 | 1 | 1 | 1 | 29 |
| 1 | 0 | 0 | 0 | 0 | 0 | 30 |
| 1 | 0 | 0 | 0 | 0 | 1 | 31 |
| 1 | 0 | 0 | 0 | 1 | 0 | 32 |
| 1 | 0 | 0 | 0 | 1 | 1 | 33 |
| 1 | 0 | 0 | 1 | 0 | 1 | 34 |
| 1 | 0 | 0 | 1 | 1 | 0 | 35 |
| 1 | 0 | 0 | 1 | 1 | 1 | 36 |
| 1 | 0 | 1 | 0 | 0 | 0 | 37 |
| 1 | 0 | 1 | 0 | 0 | 1 | 38 |
| 1 | 0 | 1 | 0 | 1 | 0 | 39 |
| 1 | 0 | 1 | 0 | 1 | 1 | 40 |
| 1 | 0 | 1 | 1 | 1 | 0 | 41 |
| 1 | 0 | 1 | 1 | 1 | 1 | 42 |
| 1 | 1 | 0 | 0 | 0 | 0 | 43 |
| 1 | 1 | 0 | 0 | 0 | 1 | 44 |
| 1 | 1 | 0 | 0 | 1 | 0 | 45 |
| 1 | 1 | 0 | 0 | 1 | 1 | 46 |
| 1 | 1 | 0 | 1 | 0 | 0 | 47 |
| 1 | 1 | 0 | 1 | 0 | 1 | 48 |

#### 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. The HFCLK predivider is initially set to a divide value of 4 to reduce current consumption at startup.

### 8.2.2 Internal Low-speed Oscillator

The internal low-speed oscillator operates with no external components and outputs a stable clock at 40-kHz nominal. The ILO is relatively low power and low accuracy. It can be calibrated periodically using a higher accuracy, high-frequency clock to improve accuracy. The ILO is available in all power modes. The ILO is always used as the system low-frequency clock LFCLK in the device. The ILO is a relatively inaccurate (±60 percent overvoltage and temperature) oscillator, which is used to generate low-frequency clocks. If calibrated against the IMO when in operation, the ILO is accurate to ±10 percent for stable temperature and voltage. The ILO is recommended to be always on, because it is the source of the WDT, which is required for reliable system operation. The ILO can be disabled by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDT reset will automatically enable the ILO.

**Note** Disabling the ILO reset is not recommended if:

- WDT protection is required against firmware crashes
- WDT protection is required against the power supply events that produce sudden brownout events that may in turn compromise the CPU functionality.

See the Watchdog Timer chapter on page 62 for details.

### 8.2.3 External Clock (EXTCLK)

The external clock (EXTCLK) is a MHz range clock that can be generated from a signal on a designated PSoC 4 pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is0–16 MHz. The device always starts up using the IMO and the external clock must be enabled in user mode; so the device cannot be started from a reset, which is clocked by the external clock.

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the I/O System chapter on page 37 for more details.

## 8.3 Clock Distribution

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

- HFCLK input selection
- HFCLK predivider configuration
- SYSCLK prescaler configuration
- Peripheral 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 the CLK_SELECT register's DIRECT_SEL bits, as described in Table 8-2.

Table 8-2. HFCLK Input Selection Bits DIRECT_SEL

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

### 8.3.2 HFCLK Predivider Configuration

The HFCLK predivider allows the device to divide the HFCLK selection mux input before use as HFCLK. The predivider is capable of dividing the HFCLK by powers of 2 between 1 and 8. The predivider value is set using register CLK_SELECT bits HFCLK_DIV, as described in Table 8-3. The HFCLK predivider is set to a divide value of 4 during boot to reduce current consumption.

**Note** HFCLK's frequency cannot exceed 16 MHz.

Table 8-3. HFCLK Predivider Value Bits HFCLK_DIV

| Name | Description |
|---|---|
| HFCLK_DIV[1:0] | HFCLK predivider value<br>0: No divider on HFCLK<br>1: Divides HFCLK by 2<br>2: Divides HFCLK by 4<br>3: Divides HFCLK by 8 |

### 8.3.3 SYSCLK Prescaler Configuration

The SYSCLK Prescaler allows the device to divide the HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by1, 2, 4, or 8. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in Table 8-4. The prescaler is initially configured to divide by 1.

**Note** The SYSCLK frequency cannot exceed 16 MHz.

Table 8-4. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

| Name | Description |
|---|---|
| SYSCLK_DIV[1:0] | SYSCLK prescaler divide value<br>0: SYSCLK = HFCLK<br>1: SYSCLK = HFCLK/2<br>2: SYSCLK = HFCLK/4<br>3: SYSCLK = HFCLK/8 |

### 8.3.4 Peripheral Clock Divider Configuration

The four peripheral clocks are derived from the HFCLK using the 16-bit peripheral clock dividers. Each is capable of dividing the input clock by values between 1 and 65,536. Each of the four dividers is controlled by a PERI_DIV_16_CTL register, whose mapping is explained in Table 8-5.

Table 8-5. Peripheral Clock Divider Control Register PERI_DIV_16_CTLx

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enables or disables the divider<br>0: Divider disabled<br>1: Divider enabled |
| 23:8 | INT16_DIV | Divide value for the divider. Output = Input/(INT16_DIV + 1)<br>Acceptable divide values range from 0 to 65,536. |

The PERI_DIV_CMD register can be used to enable, disable, and select the type of clock dividers for all peripheral clock dividers. See the PERI_DIV_CMD in the PSoC 4000 Family: PSoC 4 Registers TRM for more details.

Input clocks to the peripherals are selected by PERI_PCLK_CTLx registers. Table 8-6 shows the peripheral clocks and their respective registers. See the PSoC 4000 Family: PSoC 4 Registers TRM for more details.

Table 8-6. Selecting Peripheral Clocks

| Clock | Register |
|---|---|
| SCB (I2C) | PERI_PCLK_CTL0 |
| CSD0 | PERI_PCLK_CTL1 |
| CSD1 | PERI_PCLK_CTL2 |
| TCPWM | PERI_PCLK_CTL3 |

## 8.4 Low-Power Mode Operation

Table 8-7. Non-Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_CTLx

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

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

## 8.5 Register List

Table 8-8. Clocking System Register List

| Register Name | Description |
|---|---|
| CLK_IMO_TRIM1 | IMO Trim Register - This register contains IMO trim, allowing fine manipulation of its frequency. |
| CLK_IMO_TRIM2 | IMO Frequency Selection Register - This register controls the frequency range of the IMO, allowing gross manipulation of its frequency. |
| CLK_ILO_CONFIG | ILO Configuration Register - This register controls the ILO configuration. |
| CLK_IMO_CONFIG | IMO Configuration Register - This register controls the IMO configuration. |
| CLK_SELECT | Clock Select - This register controls clock tree configuration, selecting different sources for the system clocks. |
| PERI_DIV_16_CTLx | Peripheral Clock Divider Control Registers - These registers configure each of the peripheral clock dividers, enabling or disabling the divider and setting the integer divide value. |
| PERI_PCLK_CTLx | Programmable clock control registers - These registers are used to select the input clocks to peripherals. |

# 9. Power Supply and Monitoring

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

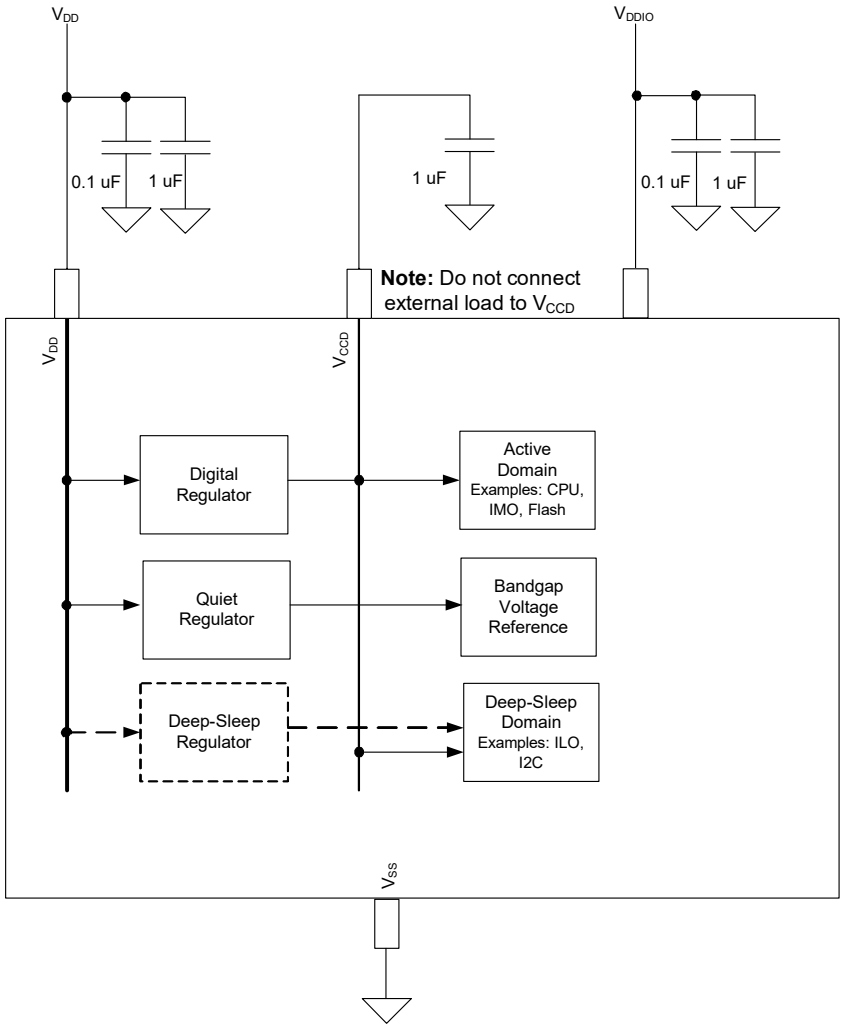
- 1.80 V to 5.50 V supply input to the internal regulators
- 1.71 V to 1.89 V[1] direct supply

There are different internal regulators to support the various power modes. These include Active digital regulator, Quiet regulator, and Deep-Sleep regulator.

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

## 9.1 Block Diagram

Figure 9-1. Power System Block Diagram




The Active digital regulator allows the external $V_{DD}$ supply to be regulated to the nominal 1.8 V required for the digital core. The output pin of this regulator has a specific capacitor requirement, as shown in Figure 9-1. This Active digital regulator is designed to supply the internal circuits only; therefore, it **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, $V_{DD}$ 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 $V_{DD}$ 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.

The $V_{DDIO}$ pin, available in certain package types, provides a separate voltage domain for the I2C pins. The chip can thus communicate with an I2C system, running at a different voltage (where $V_{DDIO} \leq V_{DD}$). For example, $V_{DD}$ can be 3.3 V and $V_{DDIO}$ can be 1.8 V. See the device datasheet for details.

One additional regulator is used to provide power in the Deep-Sleep mode.

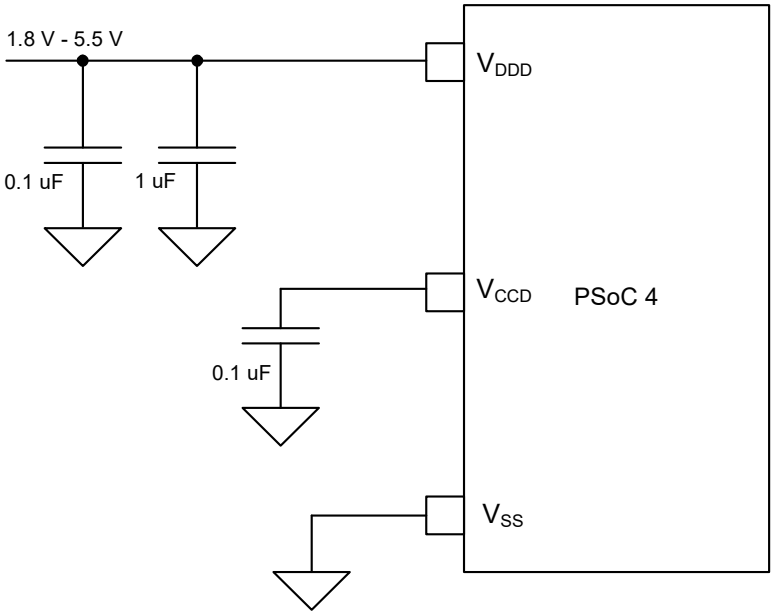
## 9.2 Power Supply Scenarios

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

### 9.2.1 Single 1.8 V to 5.5 V Unregulated Supply

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

Figure 9-2. Single Unregulated $V_{DD}$ Supply



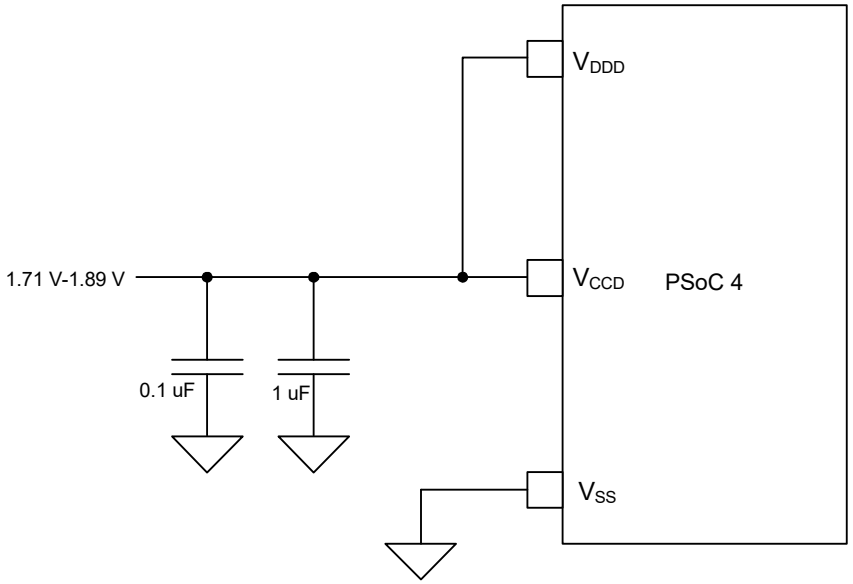

In this mode, the device is powered by an external power supply that can be anywhere in the range of 1.8 V to 5.5 V. This range is also designed for battery-powered operation; for instance, the chip can be powered from a battery system that starts at 3.5 V and works down to 1.8 V. In this mode, the internal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

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

### 9.2.2 Direct 1.71 V to 1.89 V Regulated Supply

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

Figure 9-3. Single Regulated $V_{DD}$ Supply




In this mode, $V_{CCD}$ and $V_{DDD}$ pins are shorted together and bypassed. The internal regulator should be disabled in firmware. See 9.3.1.1 Active Digital Regulator on page 55 for details.

### 9.2.3 $V_{DDIO}$ Supply.

The $V_{DDIO}$ pin, available in certain package types, provides a separate voltage domain for the I2C pins. See the device datasheet for the power supply connections when $V_{DDIO}$ is present. In applications where $V_{DDIO}$ supply is present and $V_{DD}$ is off, make sure that P3[0] and P3[1] are not floating.

## 9.3 How It Works

The regulators in Figure 9-1 power the various domains of the device. All the core regulators and digital I/Os draw their input power from the $V_{DD}$ pin supply. Digital I/Os are supplied from $V_{DD}$. The $V_{DDIO}$ pin, available in certain package types, provides a separate voltage domain for the I2C pins. See the device datasheet for details.

### 9.3.1 Regulator Summary

The Active digital regulator and Quiet regulator are enabled during the Active or Sleep power modes. They are turned off in the Deep-Sleep mode (see Table 9-1 and Figure 9-1).

Table 9-1. Regulator Status in Different Power Modes

| Mode | Active Regulator | Quiet Regulator | | |
|---|---|---|---|---|
| Stop | Off | Off | | |
| Hibernate | Off | Off | | |
| Deep Sleep | Off | Off | | |
| Sleep | On | On | | |
| Active | On | On | | |

#### *9.3.1.1 Active Digital Regulator*

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

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

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

#### 9.3.1.2 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. In Active and Sleep power modes, the main output of this regulator is connected to the output of the Active 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.

## 9.4 Voltage Monitoring

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

### 9.4.1 Power-On-Reset (POR)

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

#### 9.4.1.1 Brownout-Detect (BOD)

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

To ensure reliable operation of the device, the watchdog timer should be used in all designs. Watchdog timer provides protection against abnormal brownout conditions that may compromise the CPU functionality. See Watchdog Timer chapter on page 62 for more details.

## 9.5 Register List

Table 9-2. Power Supply and Monitoring Register List

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

# 10. Chip Operational Modes

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

PSoC 4 operational modes are:

- Boot
- User
- Privileged
- Debug

## 10.1 Boot

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

## 10.2 User

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

## 10.3 Privileged

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

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see “Performing a System Call” on page 121. Exit from this mode returns the device to user mode.

## 10.4 Debug

Debug mode is an operational mode that allows observation of the PSoC 4 operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as PSoC Creator and Arm MDK to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the Program and Debug Interface chapter on page 113.

For more details on protection modes, see the Device Security chapter on page 67.

# 11. Power Modes

The PSoC® 4 provides three power modes, intended to minimize the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep-Sleep

Active, Sleep, and Deep-Sleep are standard Arm-defined power modes, supported by the Arm CPUs and instruction set architecture (ISA).

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

- Enabling/disabling peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the PSoC 4

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

Figure 11-1. Power Mode Transitions State Diagram

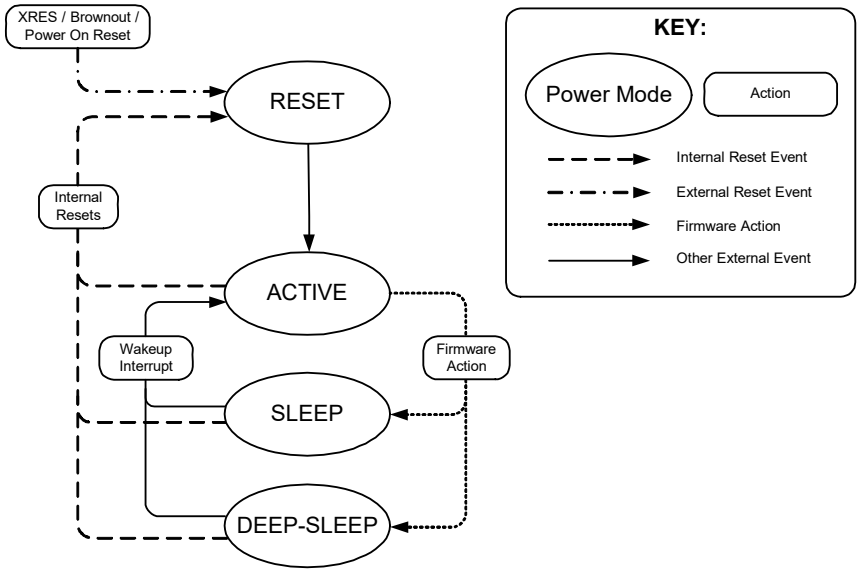


Table 11-1 illustrates the power modes offered by PSoC 4.

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 | Not applicable | All (programmable) | | All regulators are available. The Active digital regulator can be disabled if external regulation is used. |
| Sleep | CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable). | Manual register write | Any interrupt | All (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 the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency (32 kHz) clock is available.<br>Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup. | Manual register write | GPIO interrupt, I2C, watchdog timer | ILO (32 kHz) | Interrupt | Deep-Sleep regulator |

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

## 11.1 Active Mode

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

## 11.2 Sleep Mode

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

## 11.3 Deep-Sleep Mode

In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The high-frequency clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (32 kHz) oscillator remains on and low-frequency peripherals continue to operate. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, $I^2C$ slave) continue to operate. Interrupts from low-speed, asynchronous or low-power analog peripherals can cause a wakeup from Deep-Sleep mode.

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

## 11.4 Power Mode Summary

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

Table 11-2. Available Peripherals

| Peripheral | Active | Sleep | Deep-Sleep |
|---|---|---|---|
| CPU | Available | Retention[a] | Retention |
| SRAM | Available | Retention | Retention |
| High-speed peripherals | Available | Available | Retention |
| Low-speed peripherals | Available | Available | Available (optional) |
| Internal main oscillator (IMO) | Available | Available | Not Available |
| Internal low-speed oscillator (ILO,  kHz) | Available | Available | Available (optional) |
| Asynchronous peripherals | Available | Available | Available |
| Power-on-reset, Brownout detection | Available | Available | Available |
| Regular analog peripherals | Available | Available | Not Available |
| GPIO output state | Available | Available | Available |

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

Table 11-3. Wakeup Sources

| Power Mode | Wakeup Source | Wakeup Action |
|---|---|---|
| Sleep | Any interrupt source | Interrupt |
| | Any reset source | Reset |
| Deep-Sleep | GPIO interrupt | Interrupt |
| | I2C address match | Interrupt |
| | Watchdog timer | Interrupt/Reset |
| | XRES (external reset pin)[a], Brownout | Reset |
| Deep-Sleep | GPIO interrupt | Interrupt |
| | I2C address match | Interrupt |
| | Watchdog timer | Interrupt/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.

## 11.5 Low-Power Mode Entry and Exit

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

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

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

The LPM READY bit in the PWR_CONTROL register shows the status of Deep-Sleep regulator. If the firmware tries to enter Deep-Sleep mode before the regulators are ready, then PSoC 4 goes to Sleep mode first, and when the regulators are ready, the device enters Deep-Sleep mode. This operation is automatically done in hardware.

In Sleep and Deep-Sleep modes, a selection of peripherals are available (see Table 11-3), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode. See the Interrupts chapter on page 25 and the Reset System chapter on page 65 for details.

## 11.6 Register List

Table 11-4. Power Mode Register List

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

# 12. Watchdog Timer

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

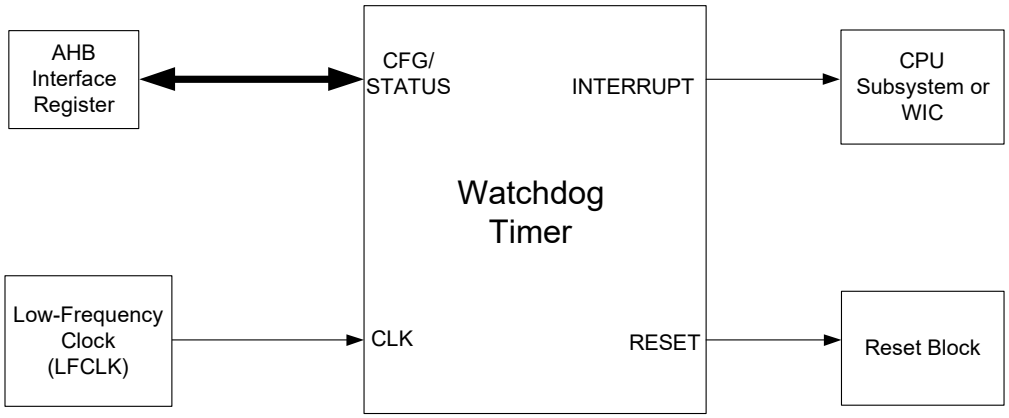
## 12.1 Features

The WDT has these features:

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

## 12.2 Block Diagram

Figure 12-1. Watchdog Timer Block Diagram




## 12.3 How It Works

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

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

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

The IGNORE_BITS in the WDT_MATCH register can be used to reduce the entire WDT counter period. The ignore bits can specify the number of MSBs that need to be discarded. For example, if the IGNORE_BITS value is 3, then the WDT counter becomes a 13-bit counter. For details, see the WDT_COUNTER, WDT_MATCH, and SRSS_INTR registers in the PSoC 4000 Family: PSoC 4 Registers TRM.

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

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

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

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

1. Write the desired IGNORE_BITS in the WDT_MATCH register to set the counter resolution.
2. Write the desired match value to the WDT_MATCH register.
3. Clear the WDT_MATCH bit in SRSS_INTR to clear any pending WDT interrupt.
4. Enable the WDT interrupt by setting the WDT_MATCH bit in SRSS_INTR_MASK
5. Enable global WDT interrupt in the CM0_ISER register (See the Interrupts chapter on page 25 for details).
6. In the ISR, clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.

For more details on interrupts, see the Interrupts chapter on page 25.

### 12.3.1 Enabling and Disabling WDT

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

**Note** Disabling the WDT reset is not recommended if:

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

### 12.3.2 WDT Interrupts and Low-Power Modes

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

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

For more details on device power modes, see the Power Modes chapter on page 58.

### 12.3.3 WDT Reset Mode

The RESET_WDT bit in the RES_CAUSE register indicates the reset generated by the WDT. This bit remains set until cleared or until a power-on reset (POR), brownout reset (BOD), or external reset (XRES) occurs. All other resets leave this bit untouched. For more details, see the Reset System chapter on page 65.

## 12.4 Register List

Table 12-1. WDT Registers

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

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

The reset system has these sources:

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

## 13.1 Reset Sources

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

### 13.1.1 Power-on Reset

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

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

### 13.1.2 Brownout Reset

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

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.

### 13.1.3 Watchdog Reset

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

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

For more details, see the Watchdog Timer chapter on page 62.

### 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 (CM0_AIRCR) forces a device reset when a '1' is written into the SYSRESETREQ bit. CM0_AIRCR requires a value of A05F written to the top two bytes for writes. Therefore, write A05F0004 for the reset.

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

### 13.1.5 External Reset

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

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

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

### 13.1.6 Protection Fault Reset

Protection fault reset (PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code. For details about privilege code, see "Privileged" on page 57.

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

## 13.2 Identifying Reset Sources

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

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

## 13.3 Register List

Table 13-1. Reset System Register List

| Register Name | Description |
|---|---|
| WDT_DISABLE_KEY | Disables the WDT when 0XACED8865 is written, for any other value WDT works normally |
| CM0_AIRCR | Cortex-M0 Application Interrupt and Reset Control Register - This register allows initiation of software resets, among other Cortex-M0 functions. |
| RES_CAUSE | Reset Cause Register - This register captures the cause of recent resets. |

# 14. Device Security

PSoC® 4 offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and enabling flash protection provide a high level of security.

The debug circuits are enabled by default and can only be disabled in firmware. If disabled, the only way to re-enable them is to erase the entire device, clear flash protection, and reprogram the device with new firmware that enables debugging. Additionally, all device interfaces can be permanently disabled for applications concerned about phishing attacks due to a maliciously reprogrammed device or attempts to defeat security by starting and interrupting flash programming sequences. Permanently disabling interfaces is not recommended for most applications because the designer cannot access the device. For more information, as well as a discussion on flash row and chip protection, see the CY8C4xxx, CYBLxxxx Programming specifications.

**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.

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

## 14.2 How It Works

### 14.2.1 Device Security

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

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

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

### 14.2.2 Flash Security

The PSoC 4 devices include a flexible flash-protection system that controls access to flash memory. This feature is designed to secure proprietary code, but it can also be used to protect against inadvertent writes to the bootloader portion of flash.

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

For more details, see the Nonvolatile Memory Programming chapter on page 120.

Table 14-1. Flash Protection Levels

| Protection Setting | Allowed | Not Allowed |
|---|---|---|
| Unprotected | External read and write,<br>Internal read and write | – |
| Full Protection | External read[a]<br>Internal read | External write,<br>Internal write |

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

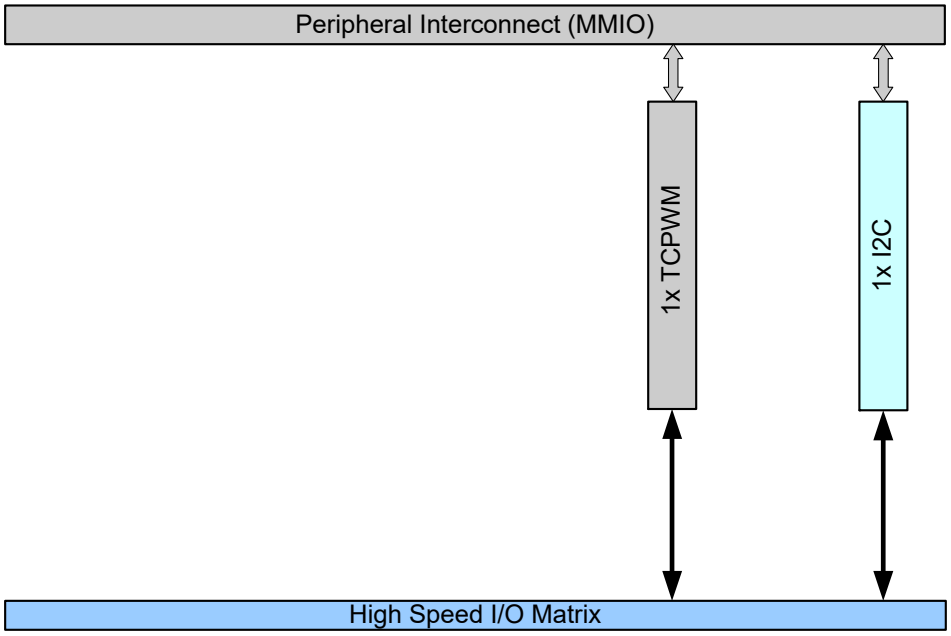
# Section E: Digital System

This section encompasses the following chapters:

- Inter-Integrated Circuit (I2C) chapter on page 70
- Timer, Counter, and PWM chapter on page 87

## Top Level Architecture

Digital System Block Diagram



# 15. Inter-Integrated Circuit (I2C)

PSoC 4 contains a Serial Communication Block (SCB) configured to operate as a fixed-function I2C block. This section explains the I²C implementation in PSoC. For more information on the I²C protocol specification, refer to the I²C-bus specification available on the UM10204, I²C-bus specification and user manual.

## 15.1 Features

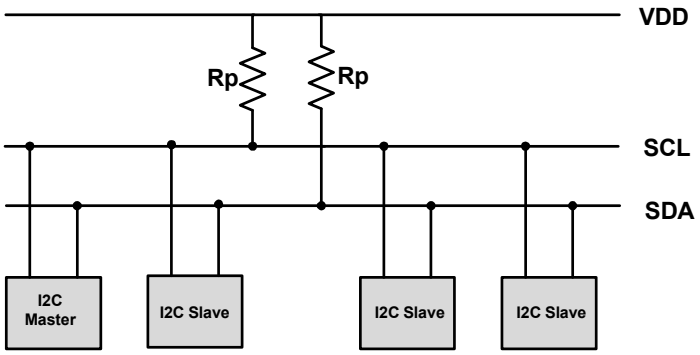
This block supports the following features:

- Master, slave, and master/slave mode
- Slow-mode (50 kbps), standard-mode (100 kbps), and fast-mode (400 kbps)data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of I²C clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I²C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface

## 15.2 General Description

Figure 15-1 illustrates an example of an I²C communication network.

Figure 15-1. I²C Interface Block Diagram




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

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

I²C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address. PSoC also supports 10-bit address matching for I²C with firmware support.

### 15.2.1 Terms and Definitions

Table 15-1 explains the commonly used terms in an I²C communication network.

Table 15-1. 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 |

#### *15.2.1.1 Clock Stretching*

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

#### *15.2.1.2 Bus Arbitration*

The I²C protocol is a multi-master, multi-slave interface. Bus arbitration is implemented on master devices by monitoring the SDA line. Bus collisions are detected when the master observes an SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value '1' on the SDA line and master 2 is driving the value '0' on the SDA line, the actual line value will be '0' due to the implementation of the I/O signal interface. Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

### 15.2.2 I²C Modes of Operation

I²C is a synchronous single master, multi-master, multi-slave serial interface. Devices operate in either master mode, slave mode, or master/slave mode. In master/slave mode, the device switches from master to slave mode when it is addressed. Only a single master may be active during a data transfer. The active master is responsible for driving the clock on the SCL line. Table 15-2 illustrates the I²C modes of operation.

Table 15-2. I²C Modes

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

Data transfer through the I²C bus follows a specific format. Table 15-3 lists some common bus events that are part of an I²C data transfer. The Write Transfer and Read Transfer sections explain the I²C bus bit format during data transfer.

Table 15-3. I²C Bus Events Terminology

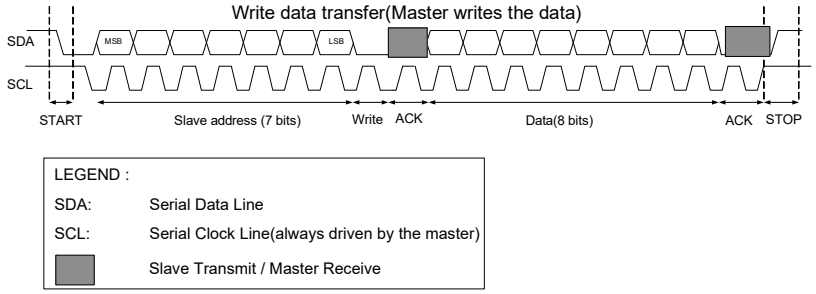
| Bus Event | Description |
|---|---|
| START | A HIGH to LOW transition on the SDA line while SCL is HIGH |
| STOP | A LOW to HIGH transition on the SDA line while SCL is HIGH |
| ACK | The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse, after the transmitter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

When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see Table 15-3, Figure 15-2, and Figure 15-3). The master looks for a STOP signal as an indicator that it can start its data transmission.

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt, so that the device is ready to respond to any other master on the bus. With all of these modes, there are two types of transfer - read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in "Master Mode Transfer Examples" on page 79, "Slave Mode Transfer Examples" on page 81, and "Multi-Master Mode Transfer Example" on page 85.

### 15.2.2.1 Write Transfer

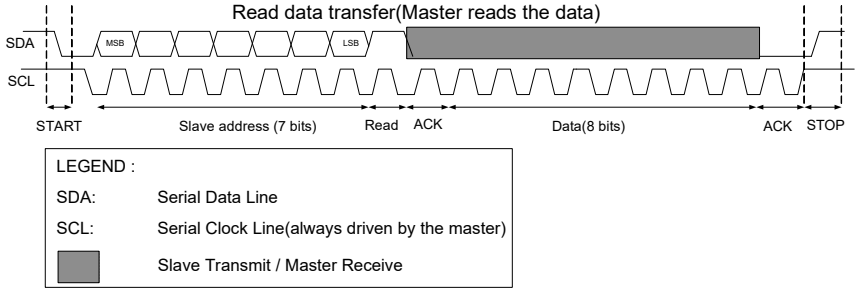
Figure 15-2. Master Write Data Transfer




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

### 15.2.2.2 Read Transfer

Figure 15-3. Master Read Data Transfer




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

## 15.2.3 Easy I2C (EZI2C) Protocol

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

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

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

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

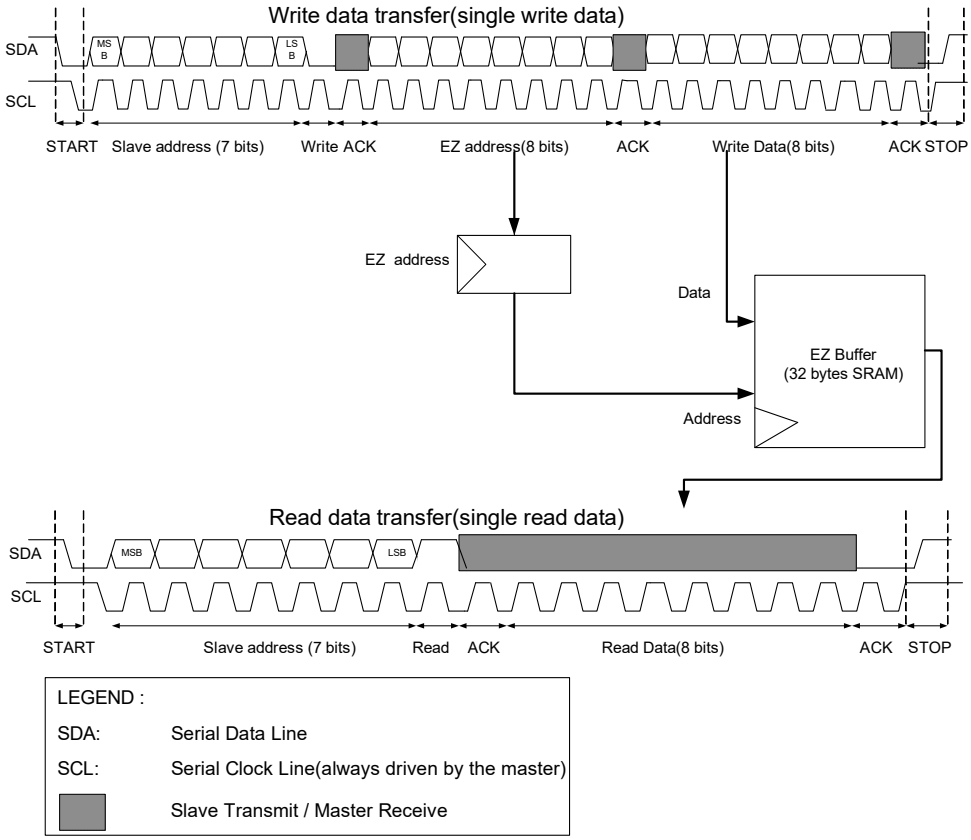
### *15.2.3.1 Memory Array Write*

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

### *15.2.3.2 Memory Array Read*

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

Figure 15-4. EZI2C Write and Read Data Transfer



### 15.2.4 I2C Registers

The I²C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in Table 15-4.

Table 15-4. I2C Registers

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

**Note** Detailed descriptions of the I²C register bits are available in the PSoC 4000 Family: PSoC 4 Registers TRM.

### 15.2.5 I2C Interrupts

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

I2C Master

- I2C master lost arbitration
- I2C master received NACK
- I2C master received ACK
- I2C master sent STOP
- I2C bus error (unexpected stop/start condition detected)

I2C Slave

- I2C slave lost arbitration
- I2C slave received NACK
- I2C slave received ACK
- I2C slave received STOP
- I2C slave received START
- I2C slave address matched
- I2C bus error (unexpected stop/start condition detected)

TX

- TX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_TX_FIFO_CTRL
- TX FIFO is not full
- TX FIFO is empty
- TX FIFO overflow
- TX FIFO underflow

RX

- RX FIFO has less entries than the value specified by TRIGGER_LEVEL in SCB_RX_FIFO_CTRL
- RX FIFO is full
- RX FIFO is not empty
- RX FIFO overflow
- RX FIFO underflow

I2C Externally Clocked

- Wake up request on address match
- I2C STOP detection at the end of each transfer
- I2C STOP detection at the end of a write transfer
- I2C STOP detection at the end of a read transfer

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

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the PSoC 4000 Family: PSoC 4 Registers TRM.

### 15.2.6 Enabling and Initializing the I2C

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

#### *15.2.6.1 I2C Standard (Non-EZ) Mode Configuration*

The I2C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to Table 15-5. 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 15-6.
   a. Specify the data frame width.
   b. Specify that MSB is the first bit to be transmitted/ received.
3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in Table 15-7.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCB_CTRL register to enable the I2C block and select the I2C mode. These register bits are shown in Table 15-8. For a complete description of the I2C registers, see the PSoC 4000 Family: PSoC 4 Registers TRM.

Table 15-5. SCB_I2C_CTRL Register

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

Table 15-6. SCB_TX_CTRL/SCB_RX_CTRL Register

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

Table 15-7. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL

| Bits | Name | Description |
|---|---|---|
| [3:0] | TRIGGER_LEVEL | Trigger level. When the transmitter FIFO has less entries or the receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case. |
| 16 | CLEAR | When '1', the transmitter or receiver FIFO and the shift registers are cleared. |
| 17 | FREEZE | When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer. |

Table 15-8. SCB_CTRL Registers

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

### *15.2.6.2 EZI2C Mode Configuration*

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

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

## 15.2.7 Internal and External Clock Operation in I2C

The I2C block supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the PSoC system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the Clocking System chapter on page 46.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality.

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

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

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

These two register fields determine the functional behavior of I2C. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power modes. Improper setting may result in faulty behavior in certain power modes. Table 15-9 and Table 15-10 describe the settings for I2C in EZ and non-EZ mode.

### 15.2.7.1 I2C Non-EZ Mode of Operation

Externally clocked operation is not supported for non-EZ functionality because there is no FIFO support for this mode. So, the EC_OP_MODE should always be set to '0' for non-EZ mode. However, EC_AM_MODE can be set to '0' or '1'. Table 15-9 gives an overview of the possibilities. The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

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

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

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

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

Table 15-9. I2C Operation in Non-EZ Mode

| I2C (Non-EZ) Mode | | | | |
|---|---|---|---|---|
| System Power Mode | EC_OP_MODE = 0 | | EC_OP_MODE = 1 | |
| | EC_AM_MODE = 0 | EC_AM_MODE = 1 | EC_AM_MODE = 0 | EC_AM_MODE = 1 |
| Active and Sleep | Address match using internal clock. Operation using internal clock. | Address match using external clock. Operation using internal clock. | Not supported | |
| Deep-Sleep | Not supported | Address match using external clock. Operation using internal clock. | | |

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

### 15.2.7.2 I2C EZ Operation Mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-10 gives an overview of the possibilities. The grey cells indicate a possible, yet not recommended setting because it involves a switch from the externally clocked logic (slave selection) to the inter-

nally 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 15-10. I2C Operation in EZ Mode

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

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

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

### 15.2.8 Wake up from Sleep

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

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

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

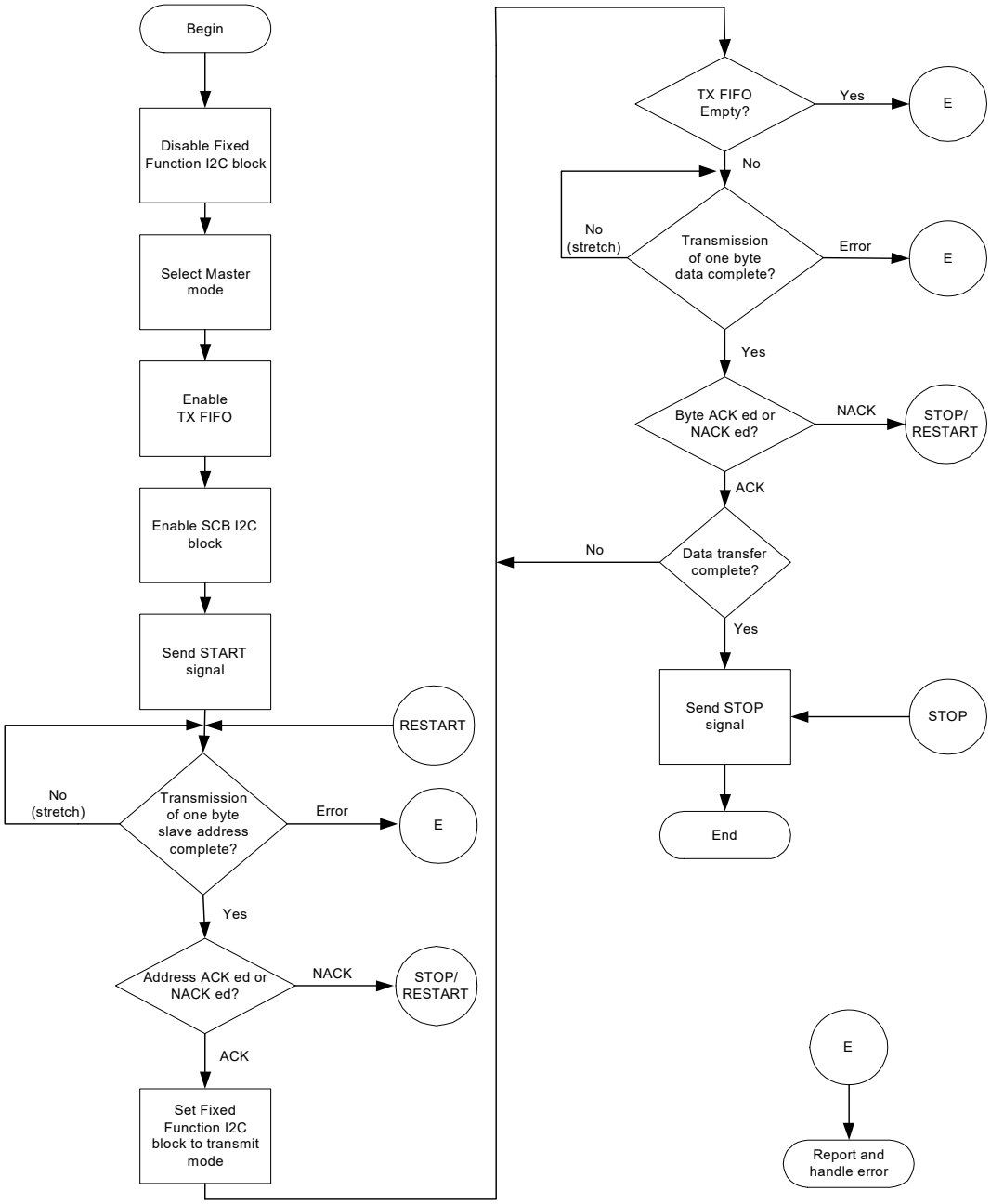
**Note** The interrupt bit WAKE_UP (bit 0) of the SCB_INTR_I2C_EC register must be enabled for the I2C to wake up the device on slave address match while switching to the Sleep mode.

## 15.2.9 Master Mode Transfer Examples

Master mode transmits or receives data.

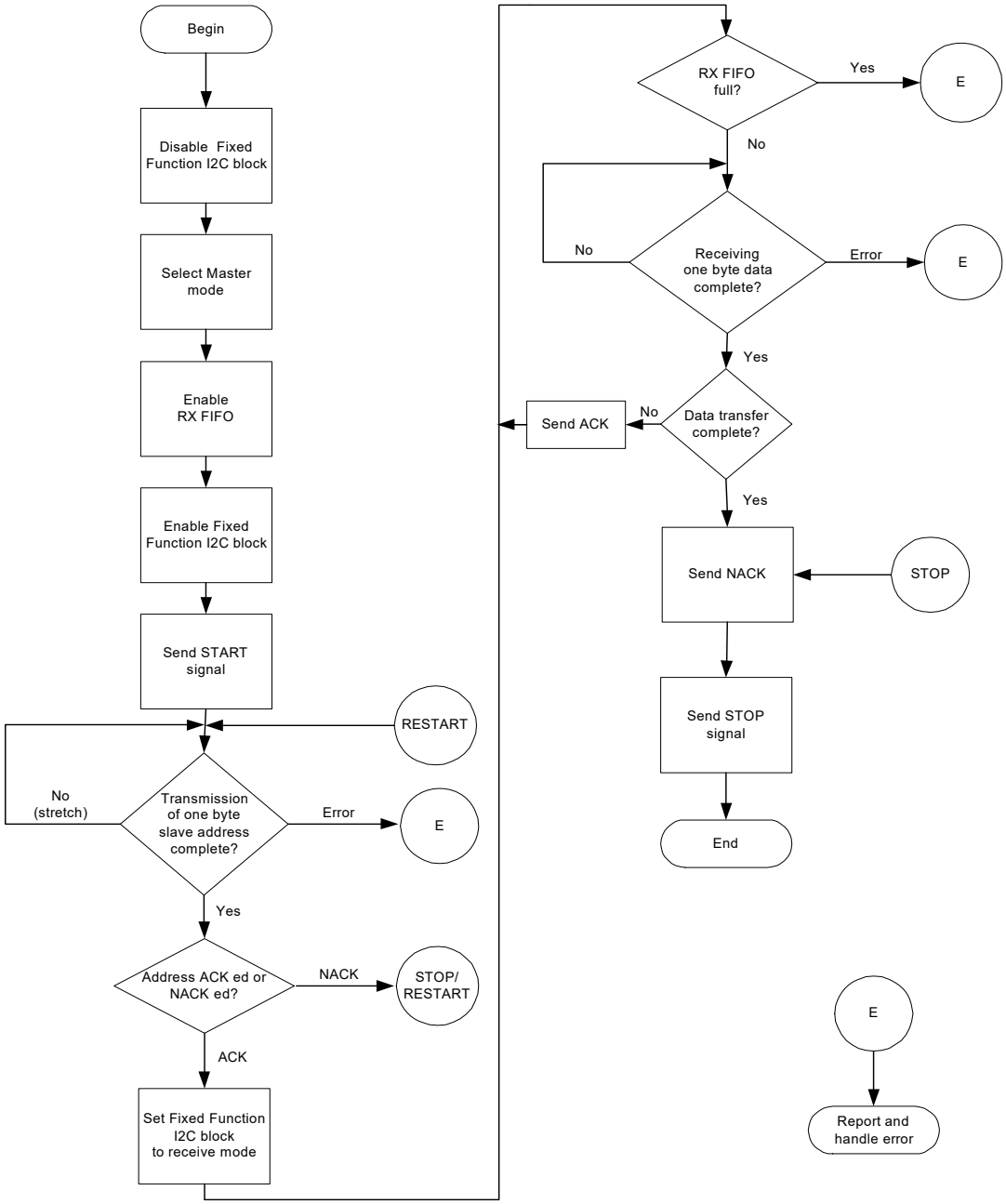
### *15.2.9.1 Master Transmit*

Figure 15-5. Single Master Mode Write Operation Flow Chart



### 15.2.9.2 Master Receive

Figure 15-6. Single Master Mode Read Operation Flow Chart

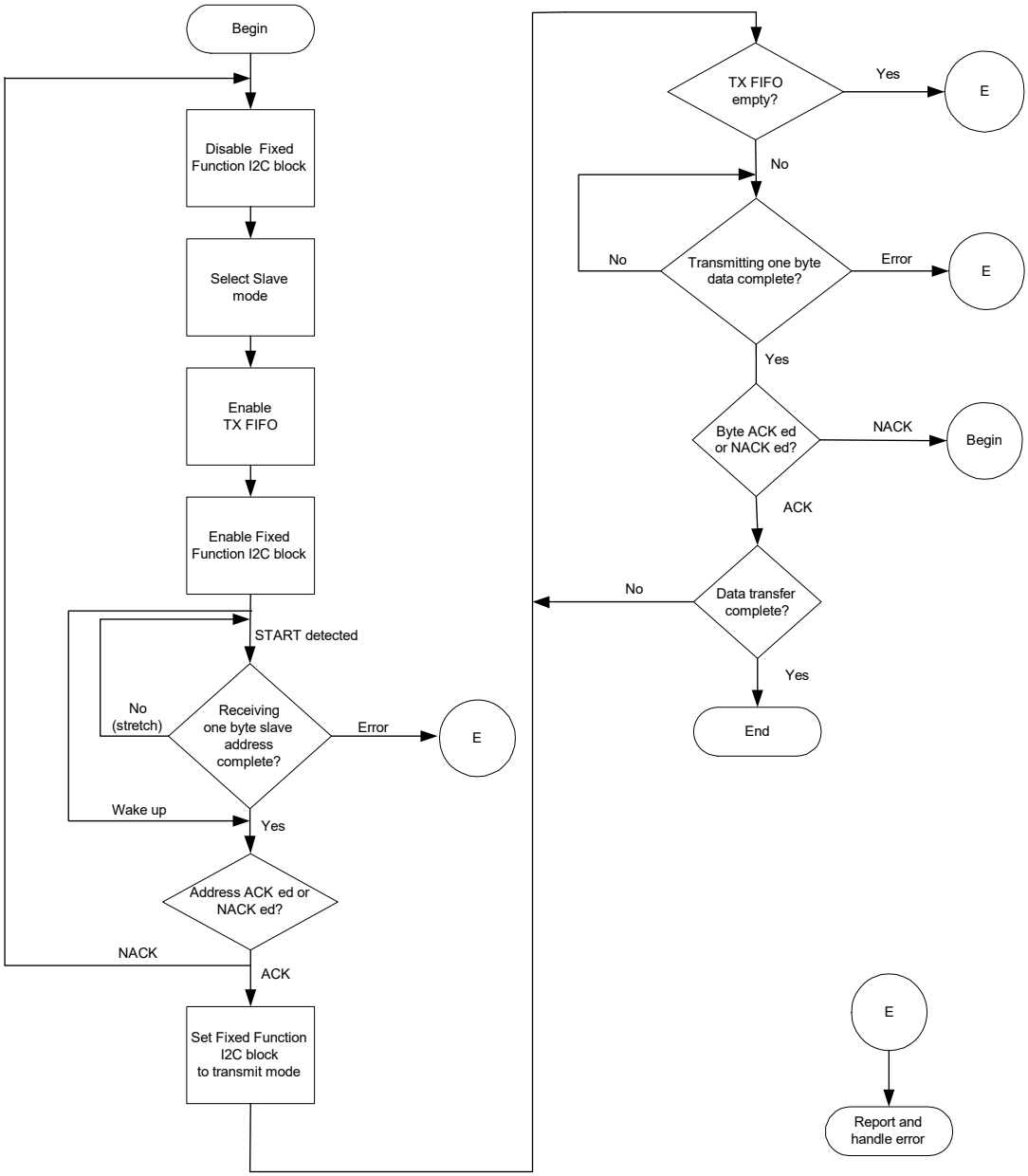


## 15.2.10 Slave Mode Transfer Examples

Slave mode transmits or receives data.

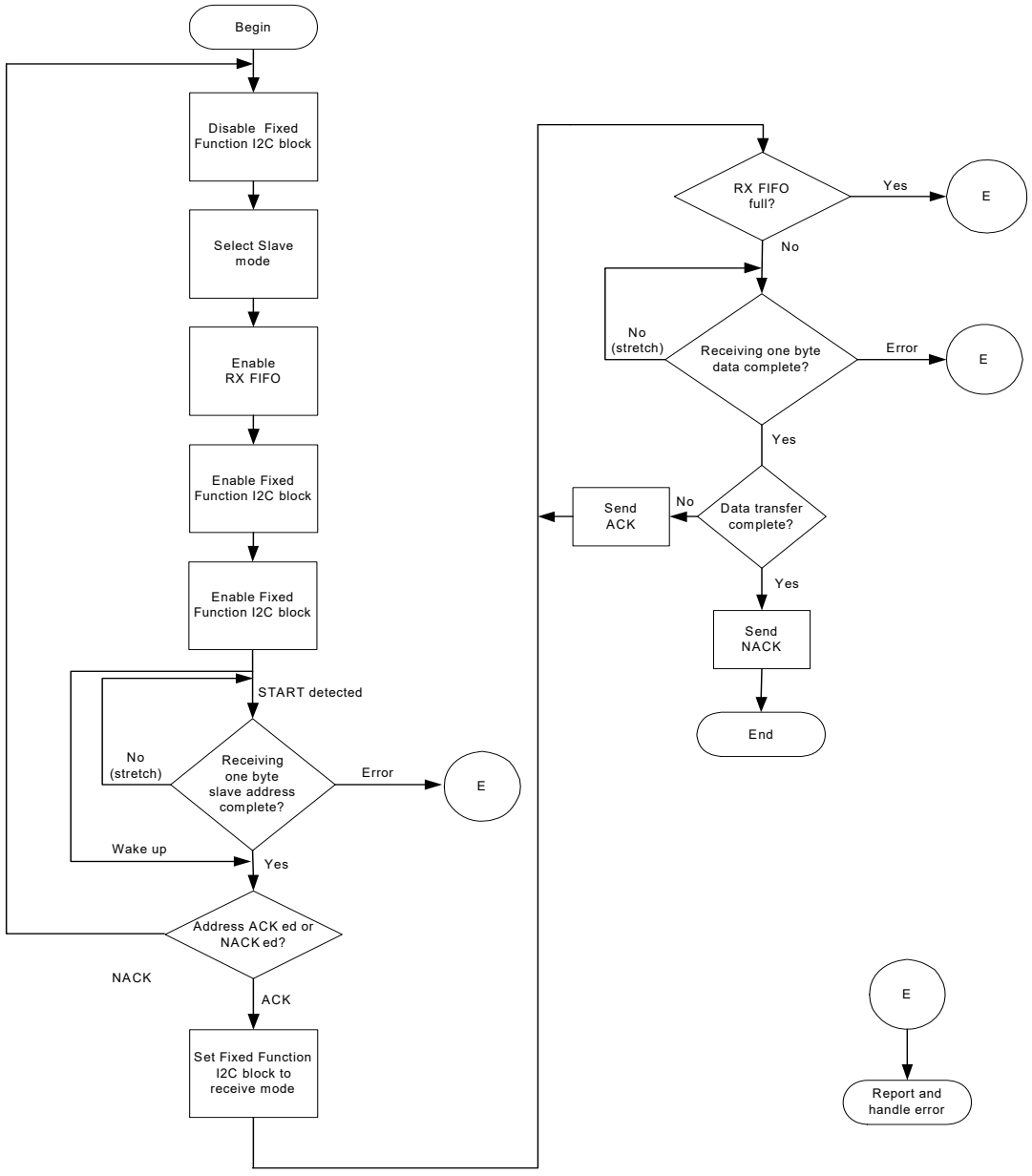
### *15.2.10.1 Slave Transmit*

Figure 15-7. Slave Mode Write Operation Flow Chart



### *15.2.10.2 Slave Receive*

Figure 15-8. Slave Mode Read Operation Flow Chart

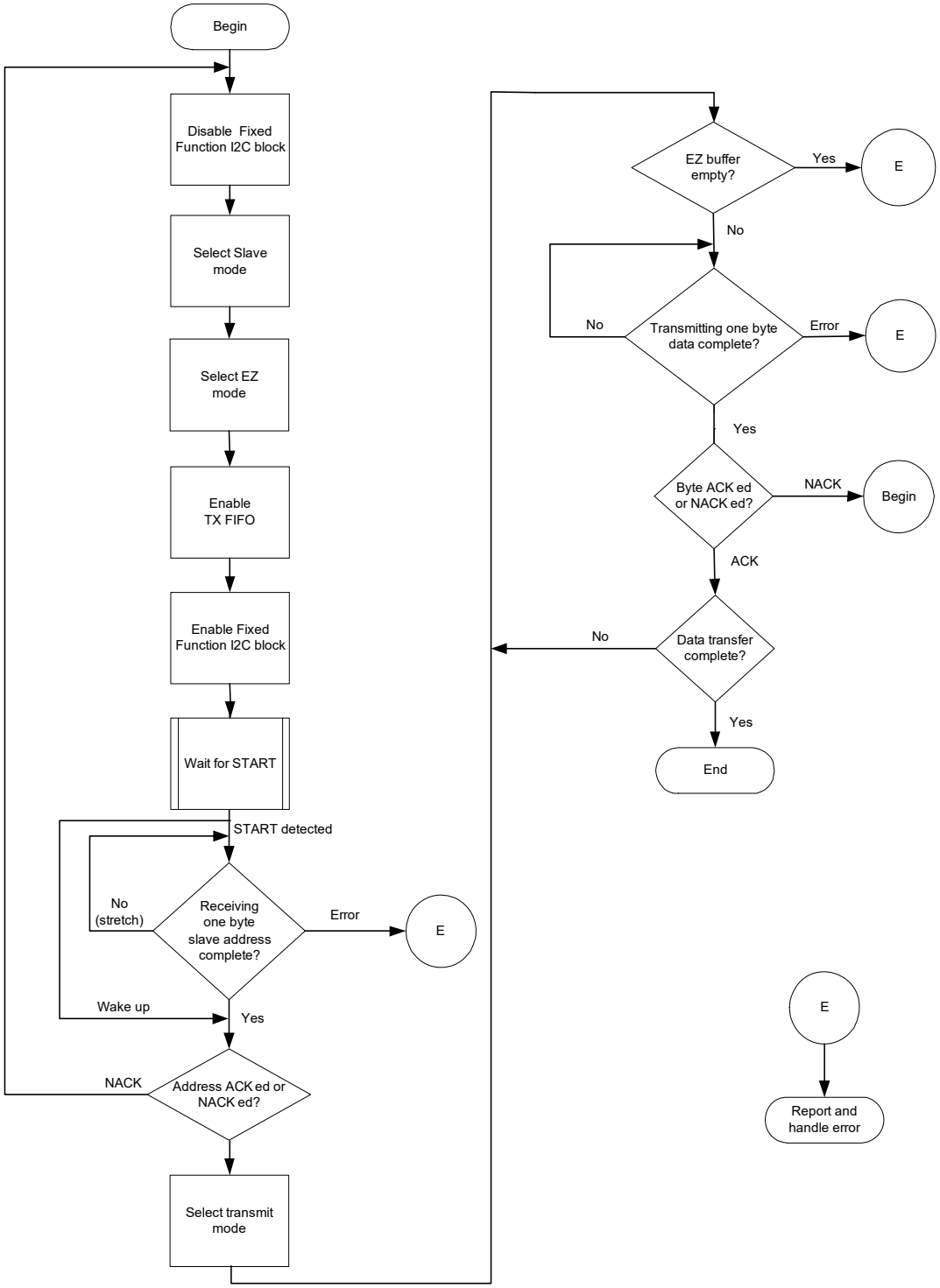


## 15.2.11 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

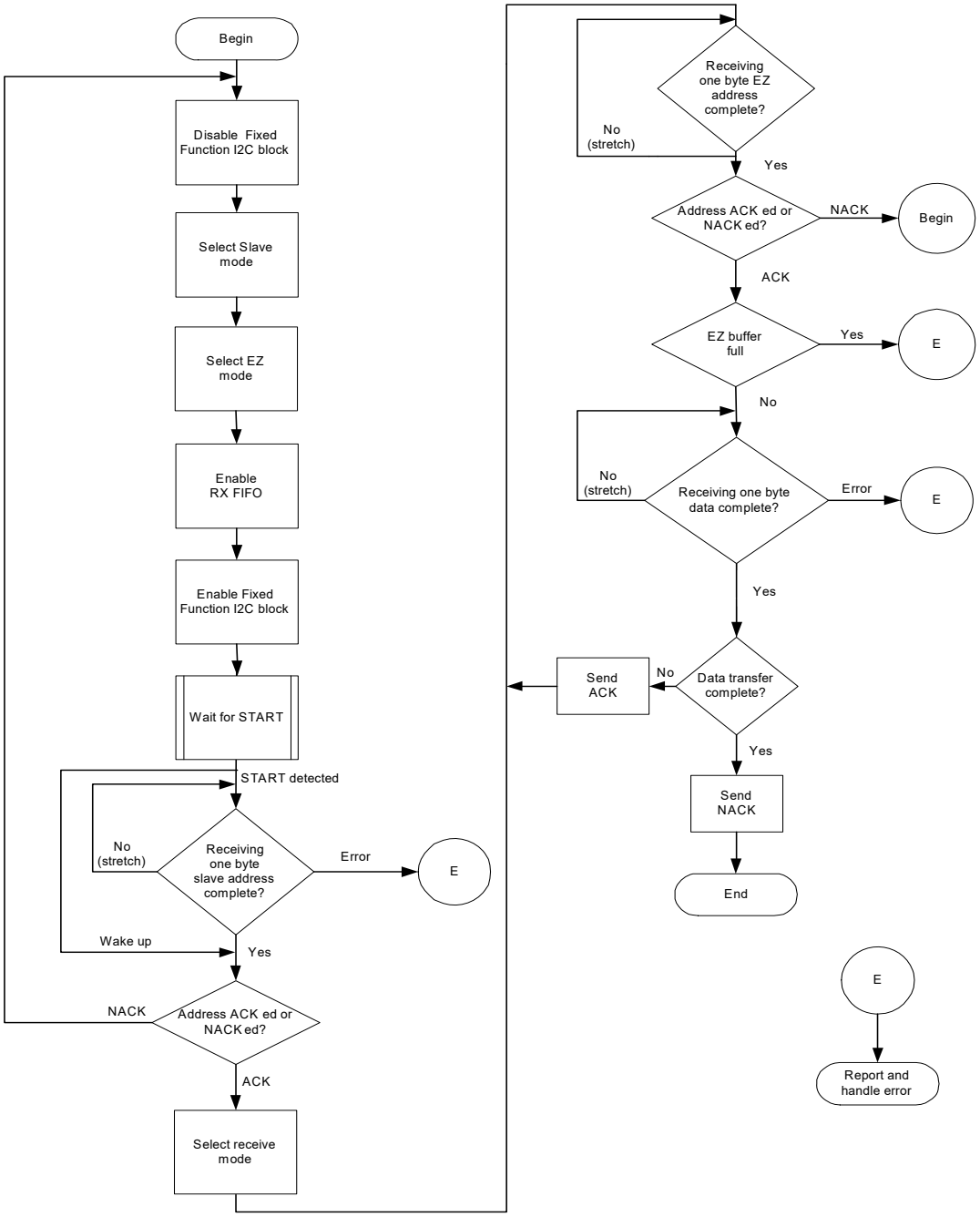
### *15.2.11.1 EZ Slave Transmit*

Figure 15-9. EZI2C Slave Mode Write Operation Flow Chart



### *15.2.11.2 EZ Slave Receive*

Figure 15-10. EZI2C Slave Mode Read Operation Flow Chart

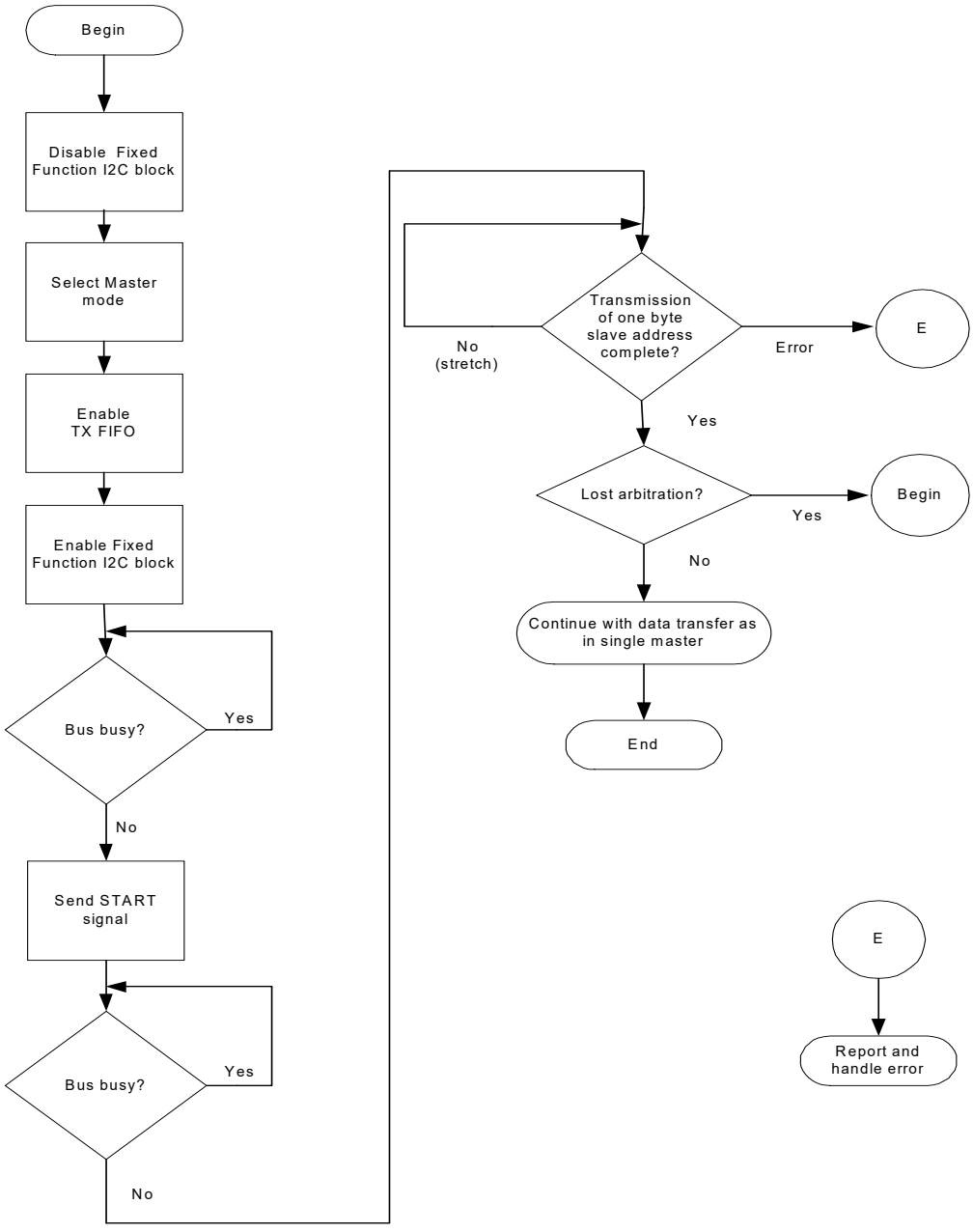


## 15.2.12 Multi-Master Mode Transfer Example

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

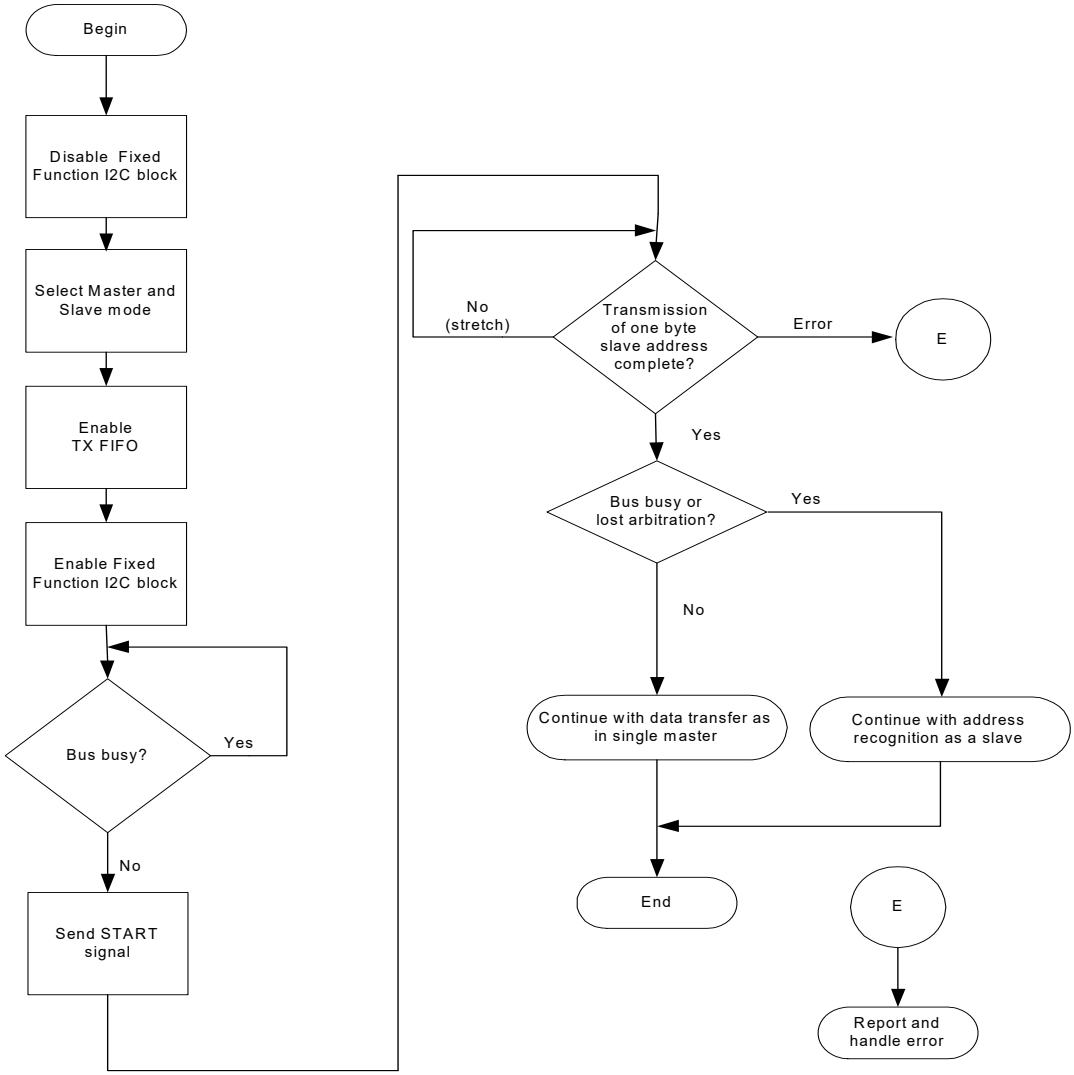
### *15.2.12.1 Multi-Master - Slave Not Enabled*

Figure 15-11. Multi-Master, Slave Not Enabled Flow Chart



### 15.2.12.2 Multi-Master - Slave Enabled

Figure 15-12. Multi-Master, Slave Enabled Flow Chart



# 16. Timer, Counter, and PWM

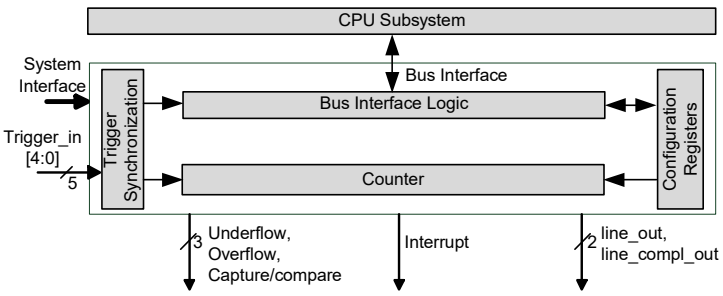
The Timer, Counter, and Pulse Width Modulator (TCPWM) block in PSoC® 4 implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular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

## 16.1 Features

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

## 16.2 Block Diagram

Figure 16-1. TCPWM Block Diagram



The block has these interfaces:

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

This TCPWM block can be configured by writing to the TCPWM registers. See "TCPWM Registers" on page 109 for more information on all registers required for this block.

### 16.2.1 Enabling and Disabling Counter in TCPWM Block

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

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

### 16.2.2 Clocking

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

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

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

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

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

### 16.2.3 Events Based on Trigger Inputs

These are the events triggered by hardware or software.

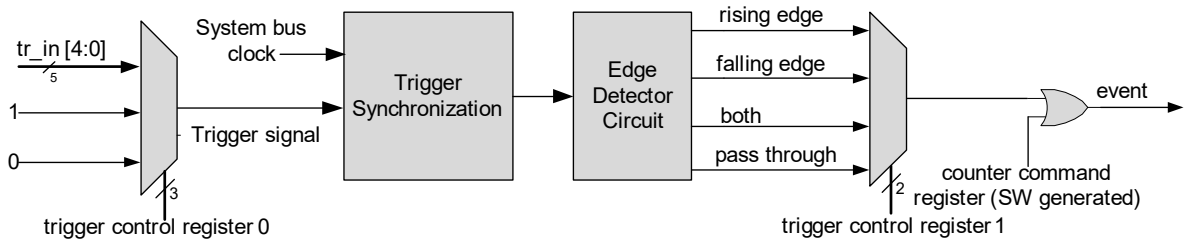
- Reload
- Start
- Stop
- Count
- Capture/switch

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

Any edge (rising, falling, or both) or level (high) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNT_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately.

Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 16-2.

Figure 16-2. Trigger Signal Edge Detection




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

- **Reload:** A reload event initializes and starts the counter.
  - In UP counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with '0'.
  - In DOWN counting mode, the counter is initialized with the period value stored in the TCPWM_CNT_PERIOD register.
  - In UP/DOWN counting mode, the count register is initialized with '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; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
  - In quadrature mode, the start event acts as quadrature phase input phiB, which is explained in detail in "Quadrature Decoder Mode" on page 98.
- **Count:** A count event causes the counter to increment or decrement, depending on its configuration.
  - In quadrature mode, the count event acts as quadrature phase input phiA.
- **Stop:** A stop event stops the counter from incrementing or decrementing. A start event will start the counting again.
  - In the PWM modes, the stop event acts as a kill event. A kill event disables all the PWM output lines.
- **Capture:** A capture event copies the counter register value to the capture register and capture register value to the buffer capture register. In the PWM modes, the capture event acts as a switch event. It switches the values of the capture/compare and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

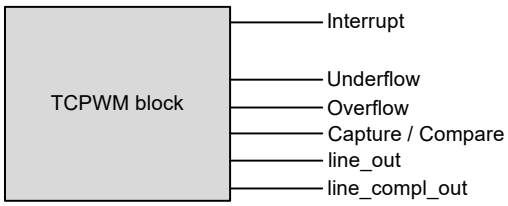
**Notes**

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

## 16.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 16-3.

Figure 16-3. TCPWM Output Signals




### *16.2.4.1 Signals upon Trigger Conditions*

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

- The capture/compare (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:
  - The counter value equals the compare value.
  - A capture event occurs - When a capture event occurs, the TCPWM_CNT_COUNTER register value is copied to the capture register and the capture register value is copied to the buffer capture register.

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

### *16.2.4.2 Interrupts*

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific. All eight interrupt output signals from the eight TCPWMs 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 16-2.

Table 16-2. Interrupt Register

| Interrupt Registers | Bits | Name | Description |
|---|---|---|---|
| TCPWM_CNT_INTR (Interrupt request register) | 0 | TC | This bit is set to '1', when a terminal count is detected. Write '1' to clear this bit. |
| | 1 | CC_MATCH | This bit is set to ‘1’ when the counter value matches capture/compare register value. Write '1' to clear this bit. |
| TCPWM_CNT_INTR_SET (Interrupt set request register) | 0 | TC | Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status. |
| | 1 | CC_MATCH | Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status. |
| TCPWM_CNT_INTR_MASK (Interrupt mask register) | 0 | TC | Mask bit for the corresponding TC bit in the interrupt request register. |
| | 1 | CC_MATCH | Mask bit for the corresponding CC_MATCH bit in the interrupt request register. |
| TCPWM_CNT_INTR_MASKED (Interrupt masked request register) | 0 | TC | Logical AND of the corresponding TC request and mask bits. |
| | 1 | CC_MATCH | Logical AND of the corresponding CC_MATCH request and mask bits. |

### *16.2.4.3 Outputs*

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

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

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

### 16.2.5 Power Modes

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

Table 16-4. Power Modes in TCPWM Block

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

## 16.3 Modes of Operation

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

Table 16-5. Operational Mode Configuration

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

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

Table 16-6. Counting Mode Configuration

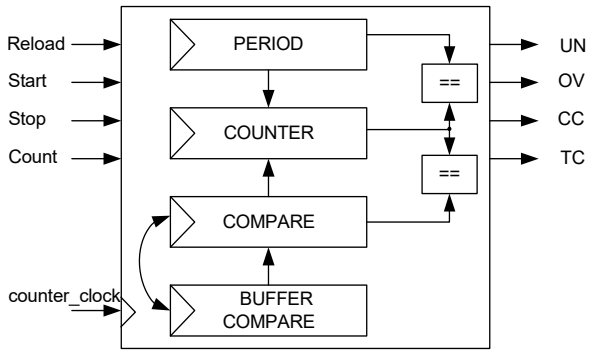
| Counting Modes | UP_DOWN_MODE[17:16] | Description |
|---|---|---|
| UP Counting Mode | 00 | Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the counter reaches the period value. |
| DOWN Counting Mode | 01 | Decrements the counter from the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

## 16.3.1 Timer Mode

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

### *16.3.1.1 Block Diagram*

Figure 16-4. Timer Mode Block Diagram




### *16.3.1.2 How It Works*

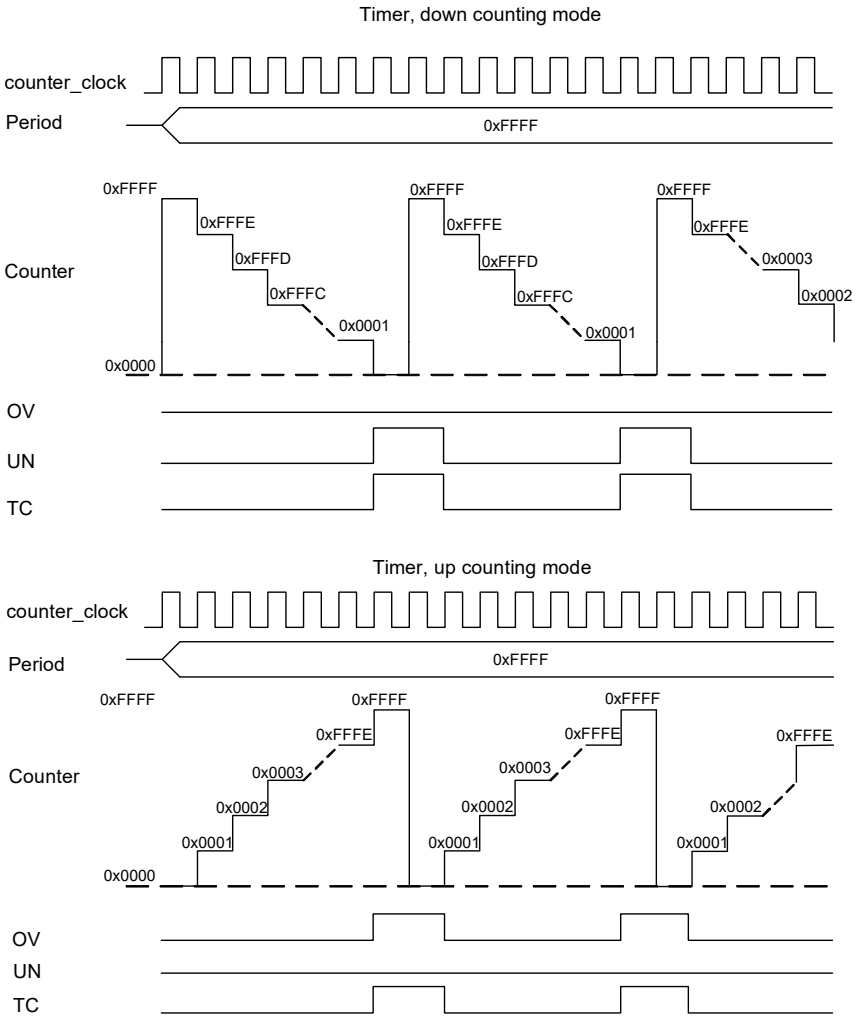
The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode. The following explains the working of the timer:

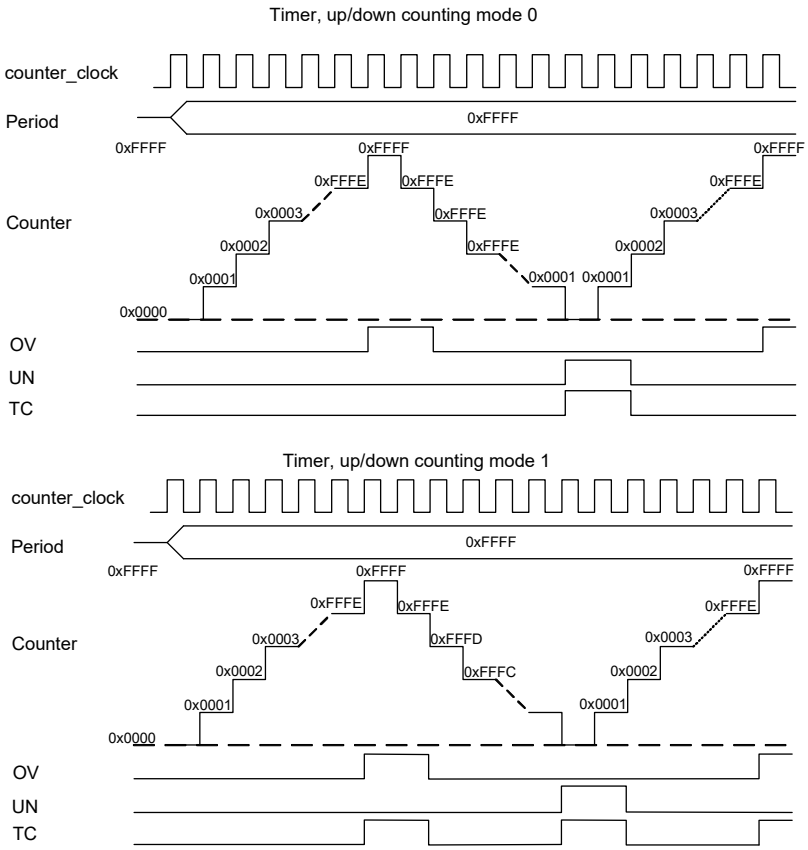
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
    **Note** It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The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

Figure 16-5 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 (0 and 1), a period value of A results in 2*A counter cycles (0 to A and back to 0).

Figure 16-5. Timing Diagram for Timer in Multiple Counting Modes






**Note** The OV and UN signals remain at logic high for two cycles of the HFCLK, as explained in "Signals upon Trigger Conditions" on page 89. The figures in this chapter assume that HFCLK and counter clock are the same.

### *16.3.1.3 Configuring Counter for Timer Mode*

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

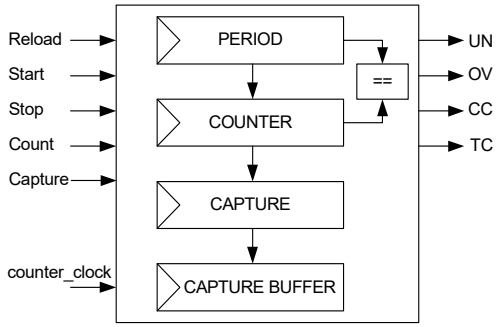
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register.
5. Set AUTO_RELOAD_CC field of the TCPWM_CNT_CTRL register, if required to switch values at every CC condition.
6. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
7. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in Table 16-6.
8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of TCPWM_CNT_CTRL.
9. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
10. Set the TCPWM_CNT_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
11. If required, set the interrupt upon TC or CC condition, as shown in "Interrupts" on page 90.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

## 16.3.2 Capture Mode

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

### *16.3.2.1 Block Diagram*

Figure 16-6. Capture Mode Block Diagram




### *16.3.2.2 How it Works*

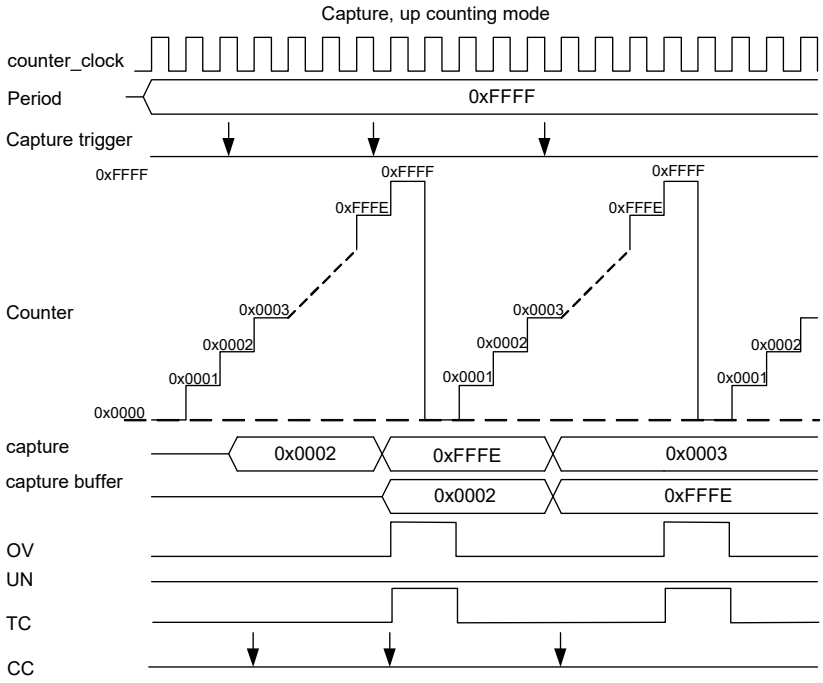
The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNT_CTRL).

Operation in capture mode occurs as follows:

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

Figure 16-7 illustrates the capture behavior in the up counting mode.

Figure 16-7. Timing Diagram of Counter in Capture Mode, Up Counting Mode



In the figure, observe that:

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

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

### *16.3.2.3 Configuring Counter for Capture Mode*

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

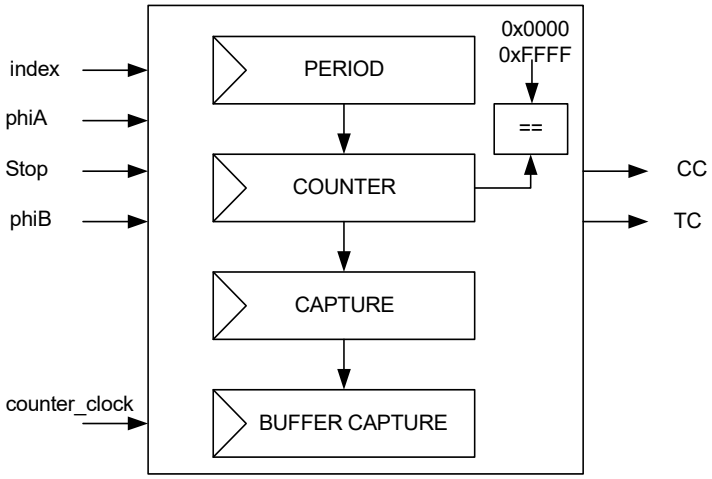
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register, as shown in Table 16-6.
6. Counter can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_-SHOT[18] field of the TCPWM_CNT_CTRL register.
7. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
8. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Stop, Capture, and Count).
9. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 90.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

## 16.3.3 Quadrature Decoder Mode

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

### *16.3.3.1 Block Diagram*

Figure 16-8. Quadrature Mode Block Diagram




### *16.3.3.2 How It Works*

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

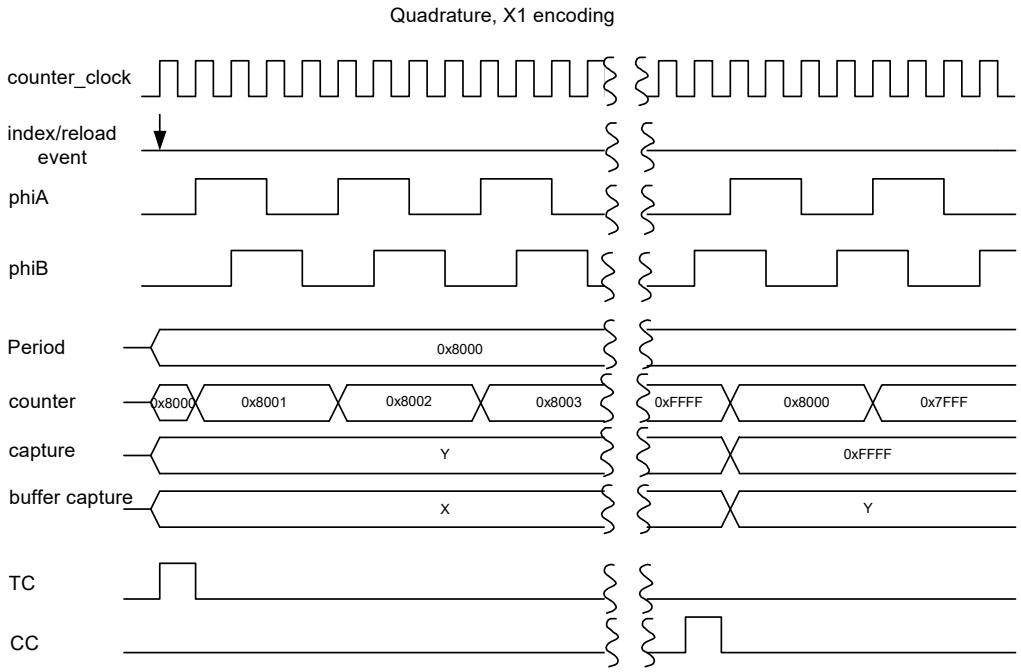
The Quadrature mode operation occurs as follows:

- Quadrature phases phiA and phiB: Counting direction is determined by the phase relationship between phiA and phiB. These phases are connected to the count and the start trigger inputs, respectively as hardware input to the decoder.
- Quadrature index signal: This is connected to the reload signal as a hardware input. This event generates a TC condition, as shown in Figure 16-9.

  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 condition can be used to generate an interrupt request
- The value in the capture register can be used to determine which condition caused the event and whether:
  - A counter underflow occurred (value 0)
  - A counter overflow occurred (value 0xFFFF)
  - An index/TC event occurred (value is not equal to either 0 or 0xFFFF)
- The DOWN bit field of counter status (TCPWM_CNTx_STATUS) register can be read to determine the current counting direction. Value '0' indicates a previous increment operation and value '1' indicates previous decrement operation. Figure 16-9 illustrates quadrature behavior in the X1 encoding mode.
  - A positive edge on phiA increments the counter when phiB is '0' and decrements the counter when phiB is '1'.
  - The count register is initialized with the period value on an index/reload event.
  - Terminal count is generated when the counter is initialized by index event. This event can be used to generate an interrupt.
  - When the count register reaches 0xFFFF (the maximum count register value), the count register value is copied to the capture register and the count register is initialized with period value (0x8000 in this case).

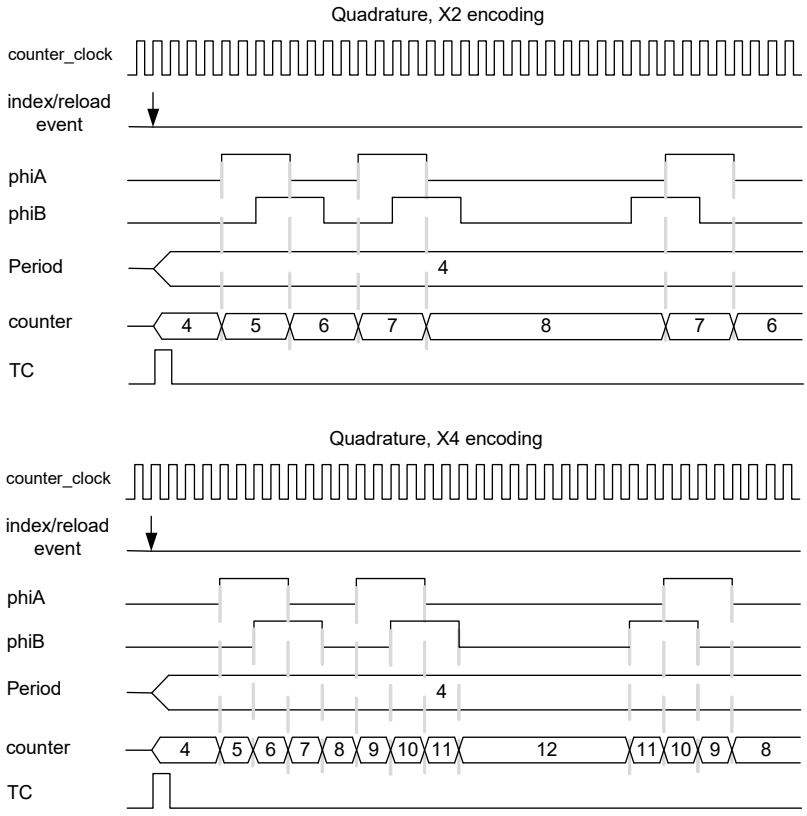
Figure 16-9. Timing Diagram for Quadrature Mode, X1 Encoding




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

Figure 16-10 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

Figure 16-10. Timing Diagram for Quadrature Mode, X2 and X4 Encoding




### 16.3.3.3 Configuring Counter for Quadrature Mode

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

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of the TCPWM_CNT_CTRL register.
5. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Index and Stop).
6. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Index and Stop).
7. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 90.
8. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

## 16.3.4 Pulse Width Modulation Mode

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

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

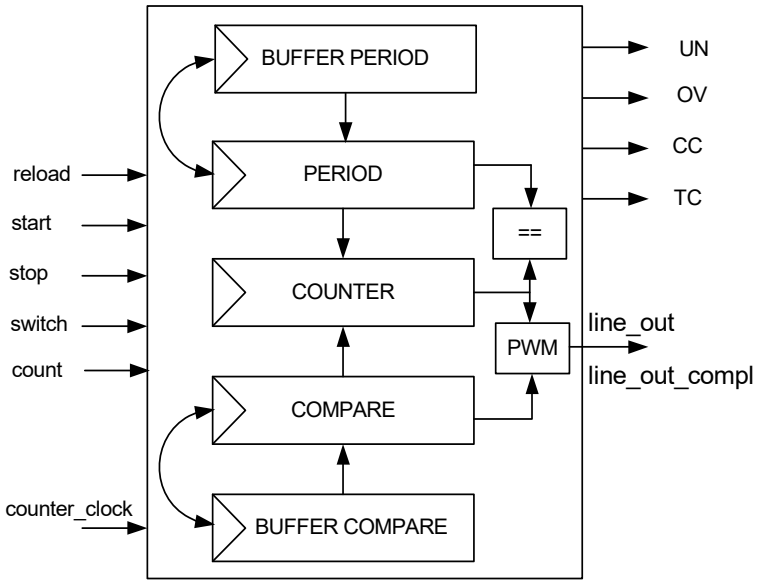
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

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

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

### *16.3.4.1 Block Diagram*

Figure 16-11. PWM Mode Block Diagram




### *16.3.4.2 How It Works*

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

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

The recommended way to modify the duty cycle is:

- The buffer period register and buffer compare register are updated with new values.
- On TC, the period and compare registers are automatically updated with the buffer period and buffer compare registers when there is an active switch event. The AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register are set to '1'. When a switch event is detected, it is remembered until the next TC event. Pass through signal (selected during event detection setting) cannot trigger a switch event.
- Updates to the buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in Figure 16-13.

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

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

Figure 16-12 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 16-12. Timing Diagram for Center Aligned PWM

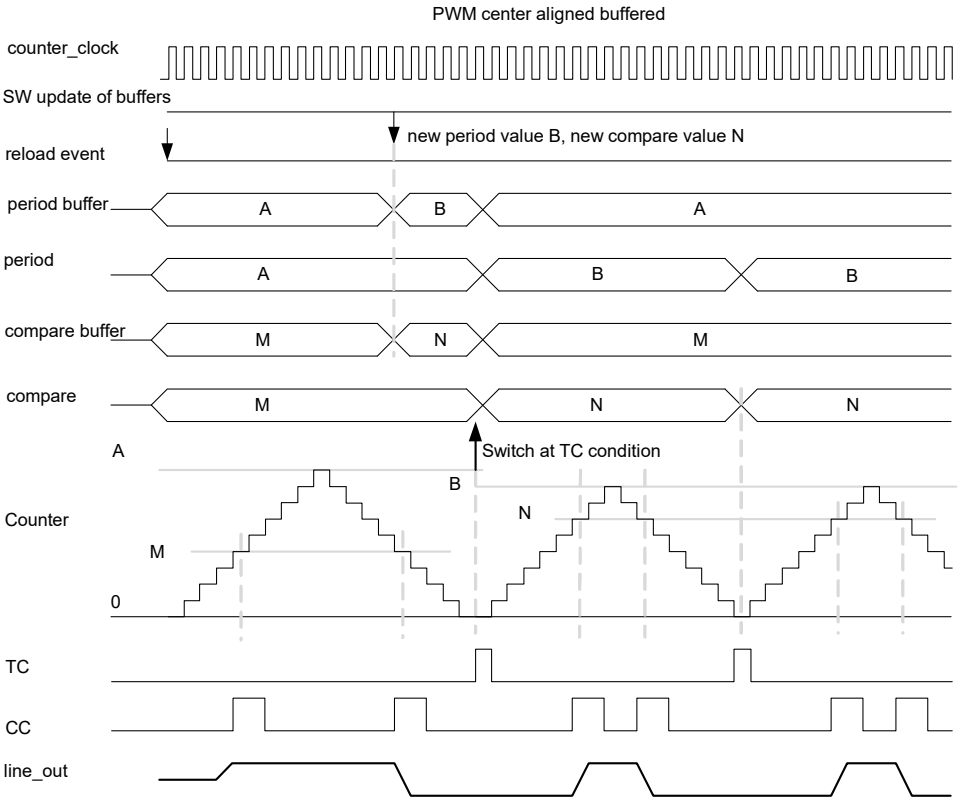
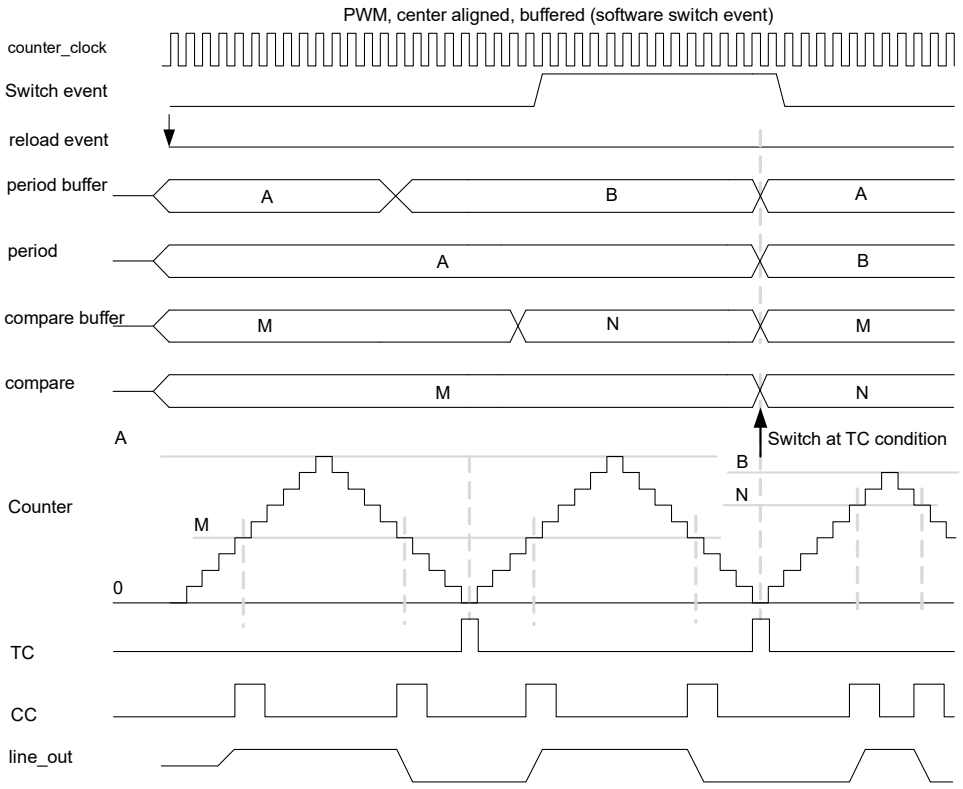



Figure 16-12 illustrates center-aligned PWM with software generated switch events:

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

Figure 16-13. Timing Diagram for Center Aligned PWM (software switch event




### 16.3.4.3 Other Configurations

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

### 16.3.4.4 Kill Feature

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

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

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

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

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

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

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

### *16.3.4.5 Configuring Counter for PWM Mode*

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

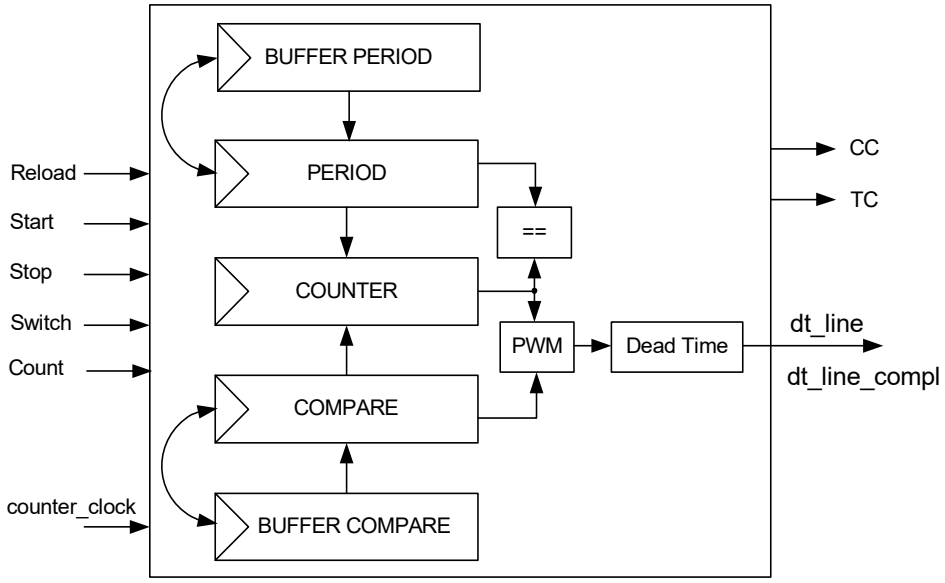
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 16-6.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 90.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

## 16.3.5 Pulse Width Modulation with Dead Time Mode

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

### *16.3.5.1 Block Diagram*

Figure 16-14. PWM-DT Mode Block Diagram




### *16.3.5.2 How It Works*

The PWM operation with Dead Time mode occurs as follows:

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

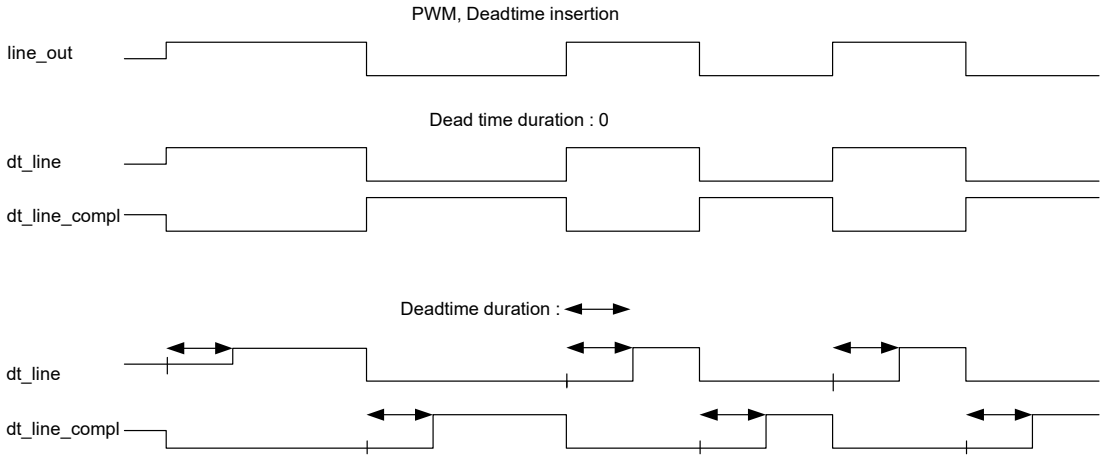
This mode follows PWM mode and supports the following features available with that mode:

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

This mode does not support clock prescaling.

Figure 16-15 illustrates how the complementary output lines dt_line and dt_line_compl are generated from the PWM output line, line_out.

Figure 16-15. Timing Diagram for PWM, with and without Dead Time




### 16.3.5.3 Configuring Counter for PWM with Dead Time Mode

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

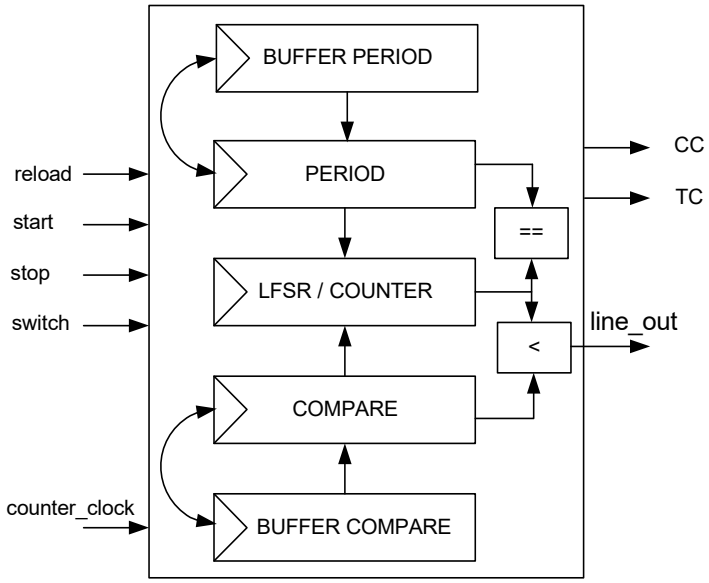
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 16-1.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 16-6.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required, as shown in the “Pulse Width Modulation Mode” on page 101.
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 90.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if hardware start signal is not enabled.

## 16.3.6 Pulse Width Modulation Pseudo-Random Mode

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

### *16.3.6.1 Block Diagram*

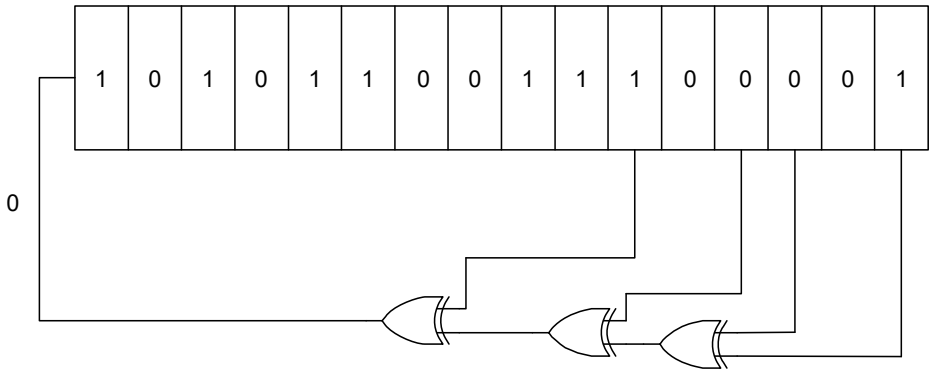
Figure 16-16. PWM-PR Mode Block Diagram




### *16.3.6.2 How It Works*

The counter register is used to implement LFSR with the polynomial: $x^{16}+x^{14}+x^{13}+x^{11}+1$, as shown in Figure 16-17. 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 non-zero value.

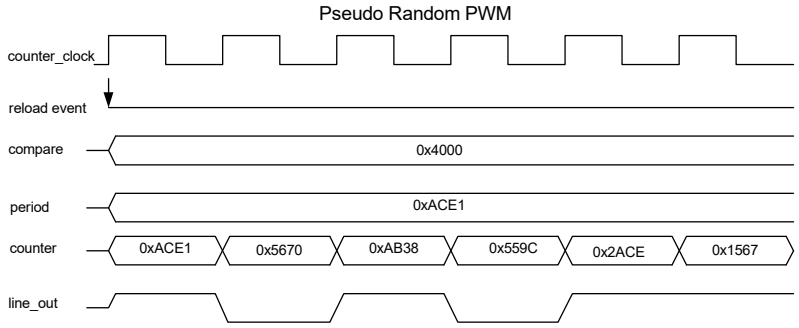
Figure 16-17. Pseudo-Random Sequence Generation using Counter Register



The following steps describe the process:

- The PWM output line, ‘line_out’, is driven with '1' when the lower 15-bit value of the counter register is smaller than the value in the compare register (when counter[14:0] < compare[15:0]). A compare value of ‘0x8000’ or higher always results in a '1' on the PWM output line. A compare value of ‘0’ always results in a '0' on the PWM output line.
- A reload event behaves similar to a start event; however, it does not initialize the counter.
- Terminal count is generated when the counter value equals the period value. LFSR generates a predictable pattern of counter values for a certain initial value. This predictability can be used to calculate the counter value after a certain amount of LFSR iterations ‘n’. This calculated counter value can be used as a period value and the TC is generated after ‘n’ iterations.
- At TC, a switch/capture event conditionally switches the compare and period register pairs (based on the AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register).
- A kill event can be programmed to stop the counter as described in previous sections.
- One shot mode can be configured by setting the ONE_-SHOT field of the counter control register. At terminal count, the counter is stopped by hardware.
- In this mode, underflow, overflow, and trigger condition events do not occur.
- CC condition occurs when the counter is running and its value equals compare value. Figure 16-18 illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50 percent duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

Figure 16-18. 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.

### *16.3.6.3 Configuring Counter for Pseudo-Random PWM Mode*

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

1. Disable the counter by writing '0' to COUNTER_ENABLED of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNT_PERIOD register and buffer period value in the TCPWM_CNT_PE-RIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_C-C_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
6. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 90.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

## 16.4 TCPWM Registers

Table 16-9. List of TCPWM Registers

| Register | Comment | Features |
|---|---|---|
| TCPWM_CTRL | TCPWM control register | Enables the counter block |
| TCPWM_CMD | TCPWM command register | Generates software events |
| TCPWM_INTR_CAUSE | TCPWM counter interrupt cause register | Determines the source of the combined interrupt signal |
| TCPWM_CNT_CTRL | Counter control register | Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock pre-scaling, and counting direction |
| TCPWM_CNT_STATUS | Counter status register | Reads the direction of counting, dead time duration, and clock pre-scaling; checks if the counter is running |
| TCPWM_CNT_COUNTER | Count register | Contains the 16-bit counter value |
| TCPWM_CNT_CC | Counter compare/capture register | Captures the counter value or compares the value with counter value |
| TCPWM_CNT_CC_BUFF | Counter buffered compare/capture register | Buffer register for counter CC register; switches period value |
| TCPWM_CNT_PERIOD | Counter period register | Contains upper value of the counter |
| TCPWM_CNT_PERIOD_BUFF | Counter buffered period register | Buffer register for counter period register; switches compare value |
| TCPWM_CNT_TR_CTRL0 | Counter trigger control register 0 | Selects trigger for specific counter events |
| TCPWM_CNT_TR_CTRL1 | Counter trigger control register 1 | Determine edge detection for specific counter input signals |
| TCPWM_CNT_TR_CTRL2 | Counter trigger control register 2 | Controls counter output lines upon CC, OV, and UN conditions |
| TCPWM_CNT_INTR | Interrupt request register | Sets the register bit when TC or CC condition is detected |
| TCPWM_CNT_INTR_SET | Interrupt set request register | Sets the corresponding bits in interrupt request register |
| TCPWM_CNT_INTR_MASK | Interrupt mask register | Mask for interrupt request register |
| TCPWM_CNT_INTR_MASKED | Interrupt masked request register | Bitwise AND of interrupt request and mask registers |

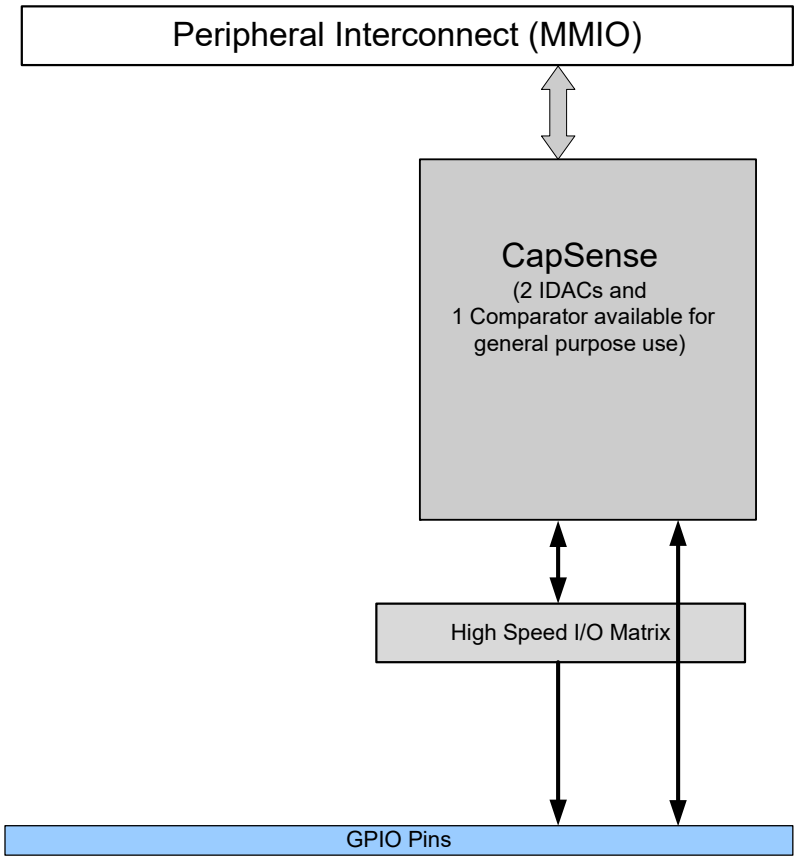
# Section F: Analog System

This section encompasses the following chapter:

- CapSense chapter on page 111

## Top Level Architecture

Analog System Block Diagram



# 17. CapSense

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

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

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

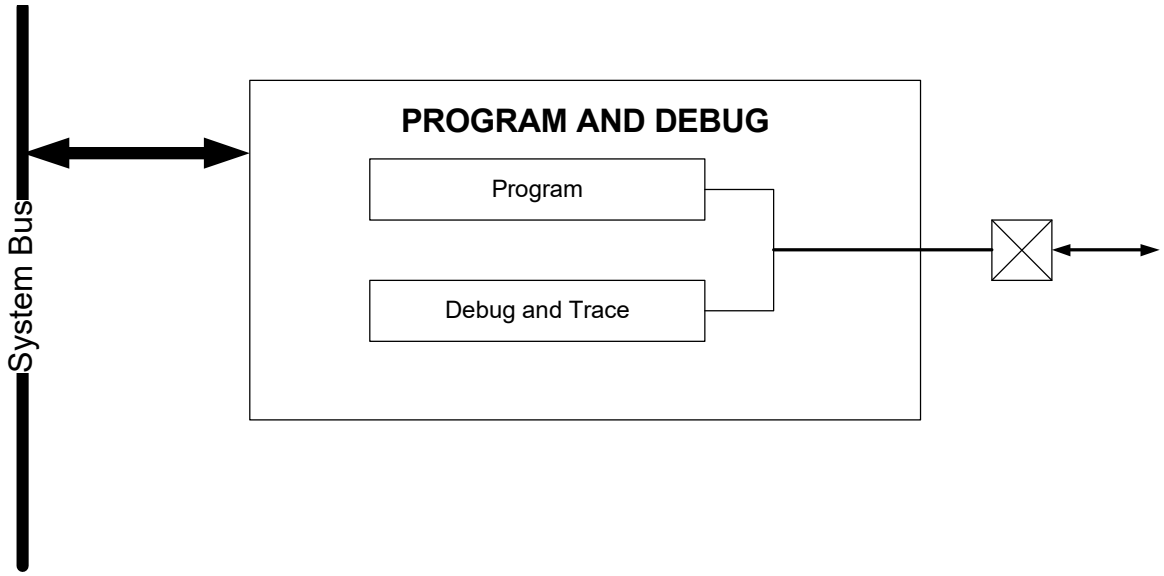
# Section G: Program and Debug

This section encompasses the following chapters:

- Program and Debug Interface chapter on page 113
- Nonvolatile Memory Programming chapter on page 120

## Top Level Architecture

Program and Debug Block Diagram



# 18. Program and Debug Interface

The PSoC® 4 Program and Debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be a Cypress-supplied programmer and debugger, or a third-party device that supports programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and PSoC 4.

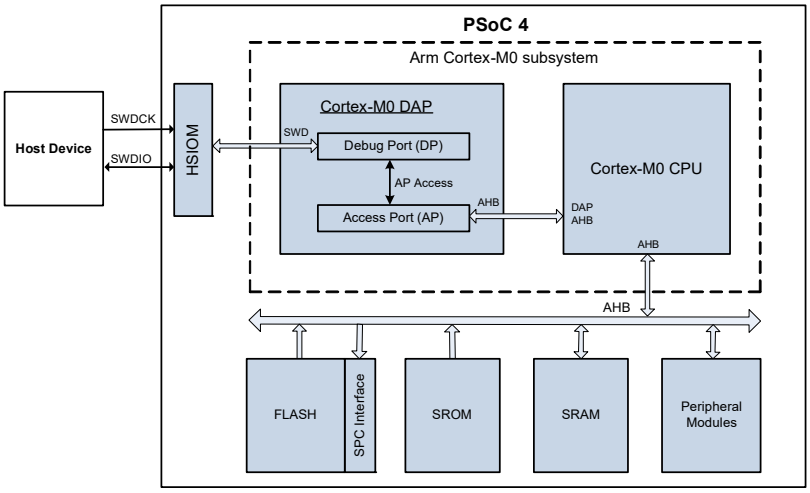
## 18.1 Features

- 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

## 18.2 Functional Description

Figure 18-1 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 I/O matrix (HSIOM). See the I/O System chapter on page 37 for details on HSIOM.

Figure 18-1. Program and Debug Interface




The DAP communicates with the Cortex-M0 CPU using the Arm-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside the device, which facilitates memory and peripheral register access by the AHB master. The device has two AHB masters – Arm CM0 CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

## 18.3 Serial Wire Debug (SWD) Interface

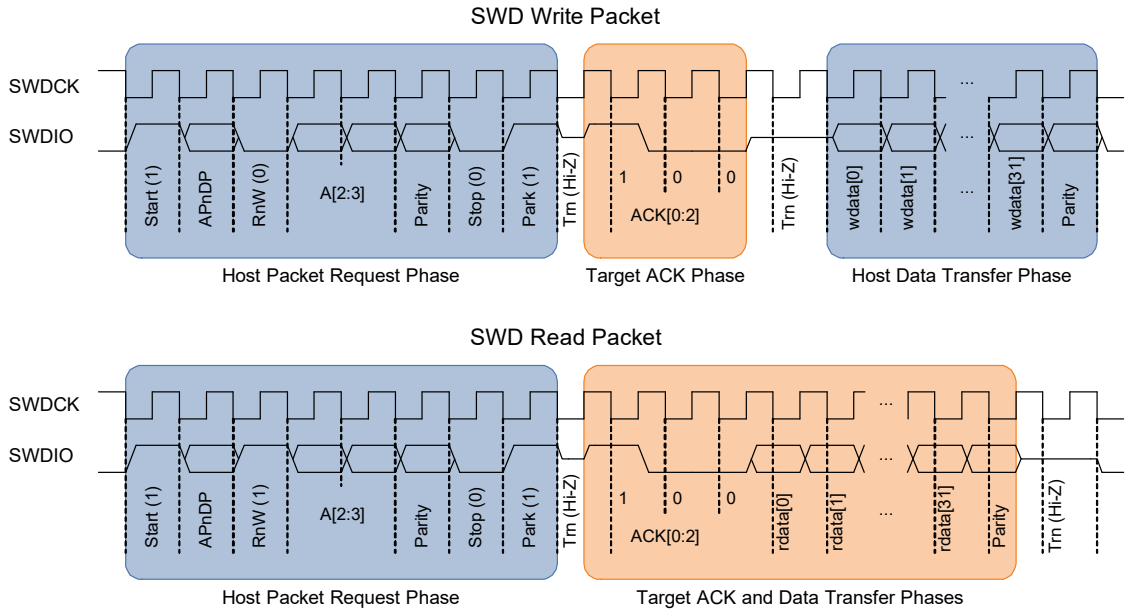
PSoC 4's Cortex-M0 supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock 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 18-2 shows the timing diagrams of read and write SWD packets.

Figure 18-2. SWD Write and Read Packet Timing Diagrams




The sequence to transmit SWD read and write packets are as follows:

1. Host Packet Request Phase: SWDIO driven by the host
   a. The start bit initiates a transfer; it is always logic 1.
   b. The "AP not DP" (APnDP) bit determines whether the transfer is an AP access – 1b1 or a DP access – 1b0.
   c. The "Read not Write" bit (RnW) controls which direction the data transfer is in. 1b1 represents a 'read from' the target, or 1b0 for a 'write to' the target.
   d. The Address bits (A[3:2]) are register select bits for AP or DP, depending on the APnDP bit value. See Table 18-3 and Table 18-4 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 18-1 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.

### 18.3.1 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 18-1 and Figure 18-2 illustrate the timing of SWDIO bit writes and reads.

Table 18-1. 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 | | |

### 18.3.2 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 18-2 shows the ACK bit-field decoding details.

Table 18-2. 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.

### 18.3.3 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 18-2. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This action ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor the PSoC 4 should drive the SWDIO line during the Trn period.

## 18.4 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.

### 18.4.1 Debug Port (DP) Registers

Table 18-3 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 18-3. 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. |

### 18.4.2 Access Port (AP) Registers

Table 18-4 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 18-4. 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 |

## 18.5 Programming the PSoC 4 Device

PSoC 4 is programmed using the following sequence.

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.

### 18.5.1 SWD Port Acquisition

#### *18.5.1.1 SWD Port Acquire Sequence*

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After removing the XRES signal, the host must send an SWD connect sequence for the device within the acquire window to connect to the SWD interface in the DAP. The pseudo code for the sequence is given here.

Code 1. SWD Port Acquire Pseudo Code

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute Arm's connection sequence to acquire SWD-port
do
{
       SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
       ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register

}while ((ack != OK) && time_elapsed < 2 ms); // retry connection until OK ACK or timeout

if (time_elapsed >= 2 ms) return FAIL; //check for acquire time out

if (ID != CM0_ID) return FAIL; //confirm SWD ID of Cortex-M0 CPU. (0x0BB11477)
```

In this pseudo code, SWD_LineReset() is the standard Arm command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (2 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.

### 18.5.2 SWD Programming Mode Entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudo code for entering the programming mode are detailed in the  document.

### 18.5.3 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 120.

## 18.6 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.

### 18.6.1 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 (CM0_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 (CM0_DFSR) – This register describes the reason a debug event has occurred and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0_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 (CM0_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0_DCRSR register.
- Debug Exception and Monitor Control Register (CM0_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

### 18.6.2 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 (CM0_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in the PSoC 4).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0_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.

### 18.6.3 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 (CM0_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 (CM0_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0_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 (CM0_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, non-invasive profiling of the program counter register.

### 18.6.4 Debugging the PSoC 4 Device

The host debugs the target PSoC 4 by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the 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 is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the cor-

rect CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the I/O System chapter on page 37 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

Debugging the target device is also affected by the overall device protection setting, which is explained in the Device Security chapter on page 67. Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

## 18.7 Registers

Table 18-5. List of Registers

| Register Name | Description |
|---|---|
| CM0_DHCSR | Debug Halting Control and Status Register |
| CM0_DFSR | Debug Fault Status Register |
| CM0_DCRSR | Debug Core Register Selector Register |
| CM0_DCRDR | Debug Core Register Data Register |
| CM0_DEMCR | Debug Exception and Monitor Control Register |
| CM0_BP_CTRL | Breakpoint control register |
| CM0_BP_COMPx | Breakpoint Compare Register |
| CM0_DWT_COMPx | Watchpoint Compare Register |
| CM0_DWT_MASKx | Watchpoint Mask Register |
| CM0_DWT_FUNCTIONx | Watchpoint Function Register |
| CM0_DWT_PCSR | Watchpoint Comparator PC Sample Register |

# 19. Nonvolatile Memory Programming

Nonvolatile memory programming refers to the programming of flash memory in the PSoC® 4 device. This chapter explains the different functions that are part of device programming, such as erase, write, program, and checksum calculation. Cypress-supplied programmers and other third-party programmers can use these functions to program the PSoC 4 device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

## 19.1 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

## 19.2 Functional Description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0 CPU requests the system call by writing the function opcode and parameters to the System Performance Controller Interface (SPCIF) input registers, and then requesting the SROM to execute the function. Based on the function opcode, the System Performance Controller (SPC) executes the corresponding system call from SROM and updates the SPCIF status register. The DAP or the CPU should read this status register for the pass/fail result of the function execution. As part of function execution, the code in SROM interacts with the SPCIF to do the actual flash programming operations.

PSoC 4 flash is programmed using a Program Erase Program (PEP) sequence. The flash cells are all programmed to a known state, erased, and then the selected bits are programmed. This sequence increases the life of the flash by balancing the stored charge. When writing to flash the data is first copied to a page latch buffer. The flash write functions are then used to transfer this data to flash.

External programmers program the flash memory in PSoC 4 using the SWD protocol by sending the commands to the Debug and Access Port (DAP). The programming sequence for the PSoC 4 device with an external programmer is given in the CY8C4xxx, CYBLxxxx Programming Specifications. Flash memory can also be programmed by the CM0 CPU by accessing the relevant registers through the AHB interface. This type of programming is typically used to update a portion of the flash memory as part of a bootload operation, or other application requirements, such as updating a lookup table stored in the flash memory. All write operations to flash memory, whether from the DAP or from the CPU, are done through the SPCIF.

**Note** It can take as much as 20 milliseconds to write to flash. During this time, the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources (see the Reset System chapter on page 65) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

## 19.3 System Call Implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: Two 8-bit parameters are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows:

  key1 = 0xB6

  key2 = 0xD3 + Opcode

  The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, additional parameters may be required depending on the specific function being called.
- Return Values: Some system calls also return a value on completion of their execution, such as the silicon ID or a checksum.
- Completion Status: Each system call returns a 32-bit status that the CPU or DAP can read to verify success or determine the reason for failure.

## 19.4 Blocking and Non-Blocking System Calls

System call functions can be categorized as blocking or non-blocking based on the nature of their execution. Blocking system calls are those where the CPU cannot execute any other task in parallel other than the execution of the system call. When a blocking system call is called from a process, the CPU jumps to the code corresponding in SROM. When the execution is complete, the original thread execution resumes. Non-blocking system calls allow the CPU to execute some other code in parallel and communicate the completion of interim system call tasks to the CPU through an interrupt.

Non-blocking system calls are only used when the CPU initiates the system call. The DAP will only use system calls during the programming mode and the CPU is halted during this process.

The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

Because the CPU cannot execute code from flash while doing an erase or program operation on the flash, the non-blocking system calls can only be called from a code executing out of SRAM. If the non-blocking functions are called from flash memory, the result is undefined and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

The System Performance Controller (SPC) is the block that generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called from SRAM, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure that the subsequent steps in the system call are completed. Because the CPU can execute code only from the SRAM when a non-blocking write or program operation is being done, the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. The Resume Non-Blocking function call done in the SPC ISR is called once in a non-blocking program operation and thrice in a non-blocking write operation.

The pseudo code for using a non-blocking write system call and executing user code out of SRAM is given later in this chapter.

### 19.4.1 Performing a System Call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, additional parameters) are:
   a. Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b. Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits

are not cleared after one second, the operation should be considered a failure and aborted without executing the following steps. Note that unlike the DAP, the CPU application code cannot poll these bits during system call execution. This is because the CPU executes code out of the SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.

4. Check the completion status: After the PRIVILEGED and SYSCALL_REQ bits are cleared to indicate completion of the system call, the CPUSS_SYSARG register should be read to check for the status of the system call. If the 32-bit value read from the CPUSS_SYSARG register is 0xAXXXXXXX (where 'X' denotes don't care hex values), the system call was successfully executed. For a failed system call, the status code is 0xF00000YY where YY indicates the reason for failure. See Table 19-1 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.

## 19.5 System Calls

Table 19-1 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 67 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 19-1. 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 64 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 | – | – | – | ? |

### 19.5.1 Silicon ID

This function returns a 12-bit family ID, 16-bit silicon ID, and an 8-bit revision ID, and the current device protection mode. These values are returned to the CPUSS_SYSARG and CPUSS_SYSREQ registers. Parameters are passed through the CPUSS_SYSARG and CPUSS_SYSREQ registers.

#### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| CPUSS_SYSARG Register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD3 | Key2 |

| Address | Value to be Written | Description |
|---|---|---|
| Bits [31:16] | 0x0000 | Not used |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0000 | Silicon ID opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

### Return

| Address | Return Value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | Silicon ID Lo | See the device datasheet for Silicon ID values for different part numbers |
| Bits [15:8] | Silicon ID Hi | |
| Bits [19:16] | Minor Revision Id | |
| 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 0x09A for PSoC 4000 |
| Bits [15:12] | Chip Protection | See the Device Security chapter on page 67 |
| Bits [31:16] | 0xXXXX | Not used |

## 19.5.2 Configure Clock

This function initializes the clock necessary for flash programming and erasing operations. This API is used to ensure that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz prior to calling the flash write and flash erase APIs. The flash write and erase APIs will exit without acting on the flash and return the "Invalid Pump Clock Frequency" status if the IMO is the source of the charge pump clock and is not 48 MHz.

### 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] | 0xE8 | 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] | 0x0015 | Configure clock 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) |

### 19.5.3 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 64 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<br>0x00 – Byte 0 of latch buffer<br>0x3F – Byte 63 of latch buffer |
| Bits [31:24] | Flash Macro Select | 0x00 – Flash Macro 0<br>0x01 – Flash Macro 1<br>(Refer to the Cortex-M0 CPU chapter on page 20 for the number of flash macros in the device) |
| SRAM Address- 32’hYY + 0x04 | | |
| Bits [7:0] | Load Size | Number of bytes to be written to the page latch buffer.<br>0x00 – 1 byte<br>0x3F – 64 bytes |
| Bits [15:8] | 0xXX | Don’t care parameter |
| Bits [23:16] | 0xXX | Don’t care parameter |
| Bits [31:24] | 0xXX | Don’t care parameter |
| SRAM Address- From (32’hYY + 0x08) to (32’hYY + 0x08 + Load Size) | | |
| Byte 0 | Data Byte [0] | First data byte to be loaded |
| . | . | . |
| . | . | . |
| Byte (Load size –1) | Data Byte [Load size –1] | Last data byte to be loaded |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32’hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0004 | Load Flash Bytes opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

#### Return

| Address | Return Value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don’t care) |

### 19.5.4 Write Row

This function erases and then programs the addressed row of flash with the data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The parameters for this function are stored in SRAM. The start address of the stored parameters is written to the CPUSS_SYSARG register. This function clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write protected.

Note that the SROM does not modify, enable, or disable any clock during any flash operation. Refer to the CLK_IMO_CONFIG register in the PSoC 4000 Family: 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<br>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) |

### 19.5.5 Program Row

This function programs the addressed row of the flash with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The row must be in an erased state before calling this function. It clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

#### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD9 | Key2 |
| Bits [31:16] | Row ID | Row number to program<br>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] | 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) |

### 19.5.6 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in supervisory flash row 0 of each flash macro.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. This API can be called only from the DAP in the programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation.

**Parameters**

| Address | Value to be Written | Description |
|---|---|---|
| SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDD | Key2 |
| Bits [31:16] | 0xXXXX | Don't care |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x000A | Erase All opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return Value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don't care) |

### 19.5.7 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When

performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDE | Key2 |
| Bits [31:16] | Row ID | Selects the flash row number on which the checksum operation is done<br>Row number – 16 bit flash row number<br>or<br>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 |

## 19.5.8 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by eight. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

Then, the Write Protection function is called, which programs the flash protection bytes from the page latch to be the corresponding flash macro’s supervisory row. In flash macro zero, which also stores the device protection settings, the device level protection setting is passed as a parameter in the CPUSS_SYSARG register.

### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xE0 | Key2 |
| Bits [23:16] | Device Protection Byte | Parameter applicable only for Flash Macro 0<br>0x01 – OPEN mode<br>0x02 – PROTECTED mode<br>0x04 – KILL mode |
| Bits [31:24] | Flash Macro Select | 0x00 – Flash Macro 0<br>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 | |

## 19.5.9 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from SRAM while the write operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 121.

The non-blocking write row system call has three phases: Pre-program, Erase, Program. Pre-program is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all of the bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code from SRAM. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After the completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

**Note** The device firmware must not attempt to put the device to sleep during a non-blocking write row. This action will reset the page latch buffer and the flash will be written with all zeroes.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0 CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDA | Key2 |
| Bits [31:16] | Row ID | Row number to write<br>0x0000 – Row 0 |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0007 | Non-Blocking Write Row opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

### Return

| Address | Return Value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don't care) |

## 19.5.10 Non-Blocking Program Row

This function is used when a flash row needs to be programmed by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from the SRAM when the program operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 121. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

Unlike the Non-Blocking Write Row system call, the Program system call only has a single phase. Therefore, the Resume Non-Blocking function only needs to be called once from the SPC interrupt when using the Non-Blocking Program Row system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0 CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

### Parameters

| Address | Value to be Written | Description |
|---|---|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDB | Key2 |
| Bits [31:16] | Row ID | Row number to write<br>0x0000 – Row 0 |
| CPUSS_SYSARG register | | |

| Address | Value to be Written | Description |
|---|---|---|
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 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) |

## 19.5.11 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are explained in Blocking and Non-Blocking System Calls on page 121.

### 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) |

## 19.6 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 19-2. System Call Status Codes

| Status Code (32-bit value in CPUSS_SYSARG register) | Description |
|---|---|
| AXXXXXXXh | Success – The "X" denotes a don't care value, which has a value of '0' returned by the SROM, unless the API returns parameters directly to the CPUSS_SYSARG register. |
| F0000001h | Invalid Chip Protection Mode – This API is not available during the current chip protection mode. |
| F0000003h | Invalid Page Latch Address – The address within the page latch buffer is either out of bounds or the size provided is too large for the page address. |
| F0000004h | Invalid Address – The row ID or byte address provided is outside of the available memory. |
| F0000005h | Row Protected – The row ID provided is a protected row. |
| F0000007h | Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API. |
| F0000008h | Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other APIs may be called. |
| F0000009h | System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume. |
| F000000Ah | Checksum Zero Failed – The calculated checksum was not zero. |
| F000000Bh | Invalid Opcode – The opcode is not a valid API opcode. |
| F000000Ch | Key Opcode Mismatch – The opcode provided does not match key1 and key2. |
| F000000Eh | Invalid Start Address – The start address is greater than the end address provided. |
| F0000012h | Invalid Pump Clock Frequency - IMO must be set to 48 MHz and HF clock source to the IMO clock source before flash write/erase operations. |

## 19.7 Non-Blocking System Call Pseudo Code

This section contains pseudo code to demonstrate how to set up a non-blocking system call and execute code out of SRAM during the flash programming operations.

```
#define REG(addr)          (*((volatile uint32 *) (addr)))
#define CM0_ISER_REG            REG( 0xE000E100 )
#define CPUSS_CONFIG_REG           REG( 0x40100000 )
#define CPUSS_SYSREQ_REG           REG( 0x40100004 )
#define CPUSS_SYSARG_REG           REG( 0x40100008 )

#define ROW_SIZE_64       (64)
#define ROW_SIZE        (ROW_SIZE_64)

/*Variable to keep track of how many times SPC ISR is triggered */
__ram int iStatusInt = 0x00;

__flash int main(void)
{
  DoUserStuff();

  /*CM0 interrupt enable bit for spc interrupt enable */
  CM0_ISER_REG |= 0x00000040;

  /*Set CPUSS_CONFIG.VECS_IN_RAM because SPC ISR should be in SRAM */
  CPUSS_CONFIG_REG |= 0x00000001;

  /*Call non-blocking write row API */
  NonBlockingWriteRow();

  /*End Program */
  while(1);
}
__sram void SpcIntHandler(void)
{
  /* Write key1, key2 parameters to SRAM */
  REG( 0x20000000 ) = 0x0000DCB6;

  /*Write the address of key1 to the CPUSS_SYSARG reg */
  CPUSS_SYSARG_REG = 0x20000000;

  /*Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
  * register and assert the sysreq bit
  */
  CPUSS_SYSREQ_REG = 0x80000009;

  /* Number of times the ISR has triggered */
  iStatusInt ++;
}
__sram void NonBlockingWriteRow(void)
{
  int iter;

  /*Load the Flash page latch with data to write*/
  * Write key1, key2, byte address, and macro sel parameters to SRAM
  */
  REG( 0x20000000 ) = 0x0000D7B6;
```

```
  //Write load size param (64 bytes) to SRAM
  REG( 0x20000004 ) = 0x0000003F;


  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 25 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.<br>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.<br>2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic. |
| ***address*** | The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored. |
| ***algorithm*** | A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation. |
| ***ambient temperature*** | The temperature of the air in a designated area, particularly the area surrounding the PSoC device. |
| ***analog*** | See ***analog signals***. |
| ***analog blocks*** | The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more. |
| ***analog output*** | An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0. |
| ***analog signals*** | A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time. |
| ***analog-to-digital (ADC)*** | A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation. |

***AND*** See *Boolean Algebra*.

***API (Application Programming Interface)*** A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications.

***array*** An array, also known as a vector or list, is one of the simplest data structures in computer programming. Arrays hold a fixed number of equally-sized data elements, generally of the same data type. Individual elements are accessed by index using a consecutive range of integers, as opposed to an associative array. Most high-level programming languages have arrays as a built-in data type. Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers; for example, by a group of two integers. One- and two-dimensional arrays are the most common. Also, an array can be a group of capacitors or resistors connected in some common form.

***assembly*** A symbolic representation of the machine language of a specific processor. Assembly language is converted to machine code by an assembler. Usually, each line of assembly code produces one machine instruction, though the use of macros is common. Assembly languages are considered low-level languages; where as C is considered a high-level language.

***asynchronous*** A signal whose data is acknowledged or acted upon immediately, irrespective of any clock signal.

***attenuation*** The decrease in intensity of a signal as a result of absorption of energy and of scattering out of the path to the detector, but not including the reduction due to geometric spreading. Attenuation is usually expressed in dB.

## B

***bandgap reference*** A stable voltage reference design that matches the positive temperature coefficient of $V_T$ with the negative temperature coefficient of $V_{BE}$, to produce a zero temperature coefficient (ideally) reference.

***bandwidth***

1. The frequency range of a message or information processing system measured in hertz.
2. The width of the spectral region over which an amplifier (or absorber) has substantial gain (or loss); it is sometimes represented more specifically as, for example, full width at half maximum.

***bias***

1. A systematic deviation of a value from a reference value.
2. The amount by which the average of a set of values departs from a reference value.
3. The electrical, mechanical, magnetic, or other force (field) applied to a device to establish a reference level to operate the device.

***bias current*** The constant low-level DC current that is used to produce a stable operation in amplifiers. This current can sometimes be changed to alter the bandwidth of an amplifier.

***binary*** The name for the base 2 numbering system. The most common numbering system is the base 10 numbering system. The base of a numbering system indicates the number of values that may exist for a particular positioning within a number for that system. For example, in base 2, binary, each position may have one of two values (0 or 1). In the base 10, decimal, numbering system, each position may have one of ten values (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

***bit*** A single digit of a binary number. Therefore, a bit may only have a value of ‘0’ or ‘1’. A group of 8 bits is called a byte. Because the PSoC's M8CP is an 8-bit microcontroller, the PSoC devices's native data chunk size is a byte.

***bit rate (BR)*** The number of bits occurring per unit of time in a bit stream, usually expressed in bits per second (bps).

***block***
1. A functional unit that performs a single function, such as an oscillator.
2. A functional unit that may be configured to perform one of several functions, such as a digital PSoC block or an analog PSoC block.

***Boolean Algebra*** In mathematics and computer science, Boolean algebras or Boolean lattices, are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the set theoretic operations union, intersection, and complement. Boolean algebra also defines a set of theorems that describe how Boolean equations can be manipulated. For example, these theorems are used to simplify Boolean equations, which will reduce the number of logic elements needed to implement the equation.

The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR, and NOT. In describing circuits, NAND (NOT AND), NOR (NOT OR), XNOR (exclusive NOT OR), and XOR (exclusive OR) may also be used. Mathematicians often use + (for example, A+B) for OR and • for AND (for example, A*B) (in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated (for example, ~A, A_, !A).

***break-before-make*** The elements involved go through a disconnected state entering (‘break”) before the new connected state (“make”).

***broadcast net*** A signal that is routed throughout the microcontroller and is accessible by many blocks or systems.

***buffer***
1. A storage area for data that is used to compensate for a speed difference, when transferring data from one device to another. Usually refers to an area reserved for I/O operations, into which data is read, or from which data is written.
2. A portion of memory set aside to store data, often before it is sent to an external device or as it is received from an external device.
3. An amplifier used to lower the output impedance of a system.

***bus***
1. A named connection of nets. Bundling nets together in a bus makes it easier to route nets with similar routing patterns.
2. A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, address[7:0].
3. One or more conductors that serve as a common connection for a group of related devices.

***byte*** A digital storage unit consisting of 8 bits.

## C

***C*** A high-level programming language.

***capacitance*** A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.

| | |
|---|---|
| ***capture*** | To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file. |
| ***chaining*** | Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another. |
| ***checksum*** | The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a pre-determined value. |
| ***clear*** | To force a bit/register to a value of logic ‘0’. |
| ***clock*** | The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks. |
| ***clock generator*** | A circuit that is used to generate a clock signal. |
| ***CMOS*** | The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor. |
| ***comparator*** | An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements. |
| ***compiler*** | A program that translates a high-level language, such as C, into machine language. |
| ***configuration*** | In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance. |
| ***configuration space*** | In PSoC devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’. |
| ***crowbar*** | A type of over-voltage protection that rapidly places a low-resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value. |
| ***CPUSS*** | CPU subsystem |
| ***crystal oscillator*** | An oscillator in which the frequency is controlled by a piezoelectric crystal. Typically a piezoelectric crystal is less sensitive to ambient temperature than other circuit components. |
| ***cyclic redundancy check (CRC)*** | A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression. |

## D

| | |
|---|---|
| ***data bus*** | A bi-directional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions. |
| ***data stream*** | A sequence of digitally encoded signals used to represent information in transmission. |
| ***data transmission*** | Sending data from one place to another by means of signals over a channel. |
| ***debugger*** | A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory. |
| ***dead band*** | A period of time when neither of two or more signals are in their active state or in transition. |
| ***decimal*** | A base-10 numbering system, which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 (called digits) together with the decimal point and the sign symbols + (plus) and - (minus) to represent numbers. |
| ***default value*** | Pertaining to the pre-defined initial, original, or specific setting, condition, value, or action a system will assume, use, or take in the absence of instructions from the user. |
| ***device*** | The device referred to in this manual is the PSoC device, unless otherwise specified. |
| ***die*** | An non-packaged integrated circuit (IC), normally cut from a wafer. |
| ***digital*** | A signal or function, the amplitude of which is characterized by one of two discrete values: ‘0’ or ‘1’. |
| ***digital blocks*** | The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI. |
| ***digital logic*** | A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system. |
| ***digital-to-analog (DAC)*** | A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation. |
| ***direct access*** | The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data. |
| ***duty cycle*** | The relationship of a clock period *high time* to its *low time*, expressed as a percent. |

## E

| | |
|---|---|
| ***External Reset (XRES_N)*** | An active high signal that is driven into the PSoC device. It causes all operation of the CPU and blocks to stop and return to a pre-defined state. |

## F

***falling edge*** A transition from a logic 1 to a logic 0. Also known as a negative edge.

***feedback*** The return of a portion of the output, or processed portion of the output, of a (usually active) device to the input.

***filter*** A device or process by which certain frequency components of a signal are attenuated.

***firmware*** The software that is embedded in a hardware device and executed by the CPU. The software may be executed by the end user, but it may not be modified.

***flag*** Any of various types of indicators used for identification of a condition or event (for example, a character that signals the termination of a transmission).

***Flash*** An electrically programmable and erasable, *volatile* technology that provides users with the programmability and data storage of EPROMs, plus in-system erasability. Nonvolatile means that the data is retained when power is off.

***Flash bank*** A group of flash ROM blocks where flash block numbers always begin with ‘0’ in an individual flash bank. A flash bank also has its own block level protection information.

***Flash block*** The smallest amount of flash ROM space that may be programmed at one time and the smallest amount of flash space that may be protected. A flash block holds 64 bytes.

***flip-flop*** A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until it is made to change to the other state by application of the corresponding signal.

***frequency*** The number of cycles or events per unit of time, for a periodic function.

## G

***gain*** The ratio of output current, voltage, or power to input current, voltage, or power, respectively. Gain is usually expressed in dB.

***gate***

1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients.
2. One of many types of combinational logic elements having at least two inputs (for example, AND, OR, NAND, and NOR (also see *Boolean Algebra*)).

***ground***

1. The electrical neutral line having the same potential as the surrounding earth.
2. The negative side of DC power supply.
3. The reference point for an electrical system.
4. The conducting paths between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

## H

***hardware*** A comprehensive term for all of the physical parts of a computer or embedded system, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.

***hardware reset*** A reset that is caused by a circuit, such as a POR, watchdog reset, or external reset. A hardware reset restores the state of the device as it was when it was first powered up. Therefore, all registers are set to the POR value as indicated in register tables throughout this document.

***hexadecimal*** A base 16 numeral system (often abbreviated and called hex), usually written using the symbols 0-9 and A-F. It is a useful system in computers because there is an easy mapping from four bits to a single hex digit. Thus, one can represent every byte as two consecutive hexadecimal digits. Compare the binary, hex, and decimal representations:

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

So the decimal numeral 79 whose binary representation is 0100 1111b can be written as 4Fh in hexadecimal (0x4F).

***high time*** The amount of time the signal has a value of '1' in one period, for a periodic digital signal.

## I

***$I^2C$*** A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). $I^2C$ is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. $I^2C$ uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.

***idle state*** A condition that exists whenever user messages are not being transmitted, but the service is immediately available for use.

| | |
|---|---|
| ***impedance*** | 1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.<br>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.<br>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.<br>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.<br>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.<br>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.<br>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.<br>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

## R



## S



## W