PRoC® BLE: CYBL1XXXX Family

Programmable Radio-on-Chip With BLE (PRoC BLE) Architecture Technical Reference Manual (TRM)

Document No. 001-93191 Rev. *D

June 6, 2017

Book 1 of 2

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (Intnl): +1.408.943.2600
www.cypress.com

Copyrights

© Cypress Semiconductor Corporation, 2014-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you under its copyright rights in the Software, a personal, non-exclusive, nontransferable license (without the right to sublicense) (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units. Cypress also grants you a personal, non-exclusive, nontransferable, license (without the right to sublicense) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely to the minimum extent that is necessary for you to exercise your rights under the copyright license granted in the previous sentence. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and Company shall and hereby does release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. Company shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

PRoC and PSoC Creator are trademarks of Cypress Semiconductor Corporation (Cypress).

Contents Overview

Contents

Section A: Overview

This section encompasses the following chapters:

- Introduction chapter on page 19
- Getting Started chapter on page 23
- Document Construction chapter on page 25

Document Revision History

Revision	Issue Date	Origin of Change	Description of Change
**	June 30, 2014	SKUV	New PRoC BLE TRM
*A	December 24, 2014	RAHU	Updated CPU System section on page 31: Updated Interrupts chapter on page 53: Updated “Interrupts and Low-Power Modes” on page 60: Updated description. Updated System-Wide Resources section on page 63: Updated Power Supply and Monitoring chapter on page 83: Updated “How It Works” on page 84: Updated “Regulator Summary” on page 84: Updated “Core Regulators” on page 84: Updated description. Updated Power Modes chapter on page 89: Updated Table 11-1. Updated “Power Mode Summary” on page 91: Updated Table 11-3. Updated Reset System chapter on page 99: Updated “Reset Sources” on page 99: Updated “Hibernate Wakeup Reset” on page 100: Updated description. Updated “Identifying Reset Sources” on page 100: Updated description. Updated Device Security chapter on page 103: Updated description before “Features” on page 103.

Document Revision History *(continued)*

Revision	Issue Date	Origin of Change	Description of Change
*A (cont.)	December 24, 2014	RAHU	Updated Digital System section on page 105: Updated Serial Communications (SCB) chapter on page 107: Updated “Serial Peripheral Interface (SPI)” on page 107: Updated “SPI Registers” on page 116: Updated Table 15-1: Updated details in “Operation” column corresponding to SCB_RX_CTRL register. Updated “Enabling and Initializing SPI” on page 117: Updated Table 15-3: Removed ENABLE bit of the SCB_TX_CTRL and SCB_RX_CTRL registers. Added MEDIAN bit in the SCB_RX_CTRL register. Updated “UART” on page 120: Updated “UART Modes of Operation” on page 121: Updated “IrDA” on page 127: Updated description. Updated Timer, Counter, and PWM chapter on page 147: Replaced GENERIC[10:8] with GENERIC[15:8] in all instances corresponding to TCPWM_CNT_CTRL register. Updated “Block Diagram” on page 148: Updated “Power Modes” on page 151: Updated Table 16-4: Added STOP power mode details. Updated Program and Debug section on page 259: Updated Program and Debug Interface chapter on page 261: Updated “Programming the PRoC Device” on page 264: Updated description. Updated “SWD Programming Mode Entry” on page 265: Updated description. Updated “SWD Programming Routines Executions” on page 265: Updated description. Updated Nonvolatile Memory Programming chapter on page 267: Updated “Functional Description” on page 267: Updated description. Updated “System Calls” on page 269: Updated “Silicon ID” on page 269: Updated description.
*B	June 26, 2015	UDYG	Added support for 256 KB flash family.
*C	February 23, 2016	UDYG	Updated the family part number to be CYBL1XXXX. Updated Figure 1-1. Added information on DMA support in the Introduction chapter and added the DMA chapter. Updated the BLESS chapter with information on support for Bluetooth 4.2-capable devices.
*D	June 06, 2017	SHEA	Updated logo and copyright information

1. Introduction

PRoC® BLE is a 32-bit, 48-MHz ARM® Cortex™-M0 BLE solution with CapSense®, 12-bit ADC, four TCPWM, thirty-six GPIOs, two serial communication blocks (SCBs), LCD, and I2S. PRoC BLE includes a royalty-free BLE stack compatible with Bluetooth 4.2 and provides a complete, programmable, and flexible solution for HID, remote controls, toys, beacons, and wireless chargers. In addition to these applications, PRoC BLE provides a simple, lowcost way to add BLE connectivity to any existing system. PRoC BLE devices have these characteristics:

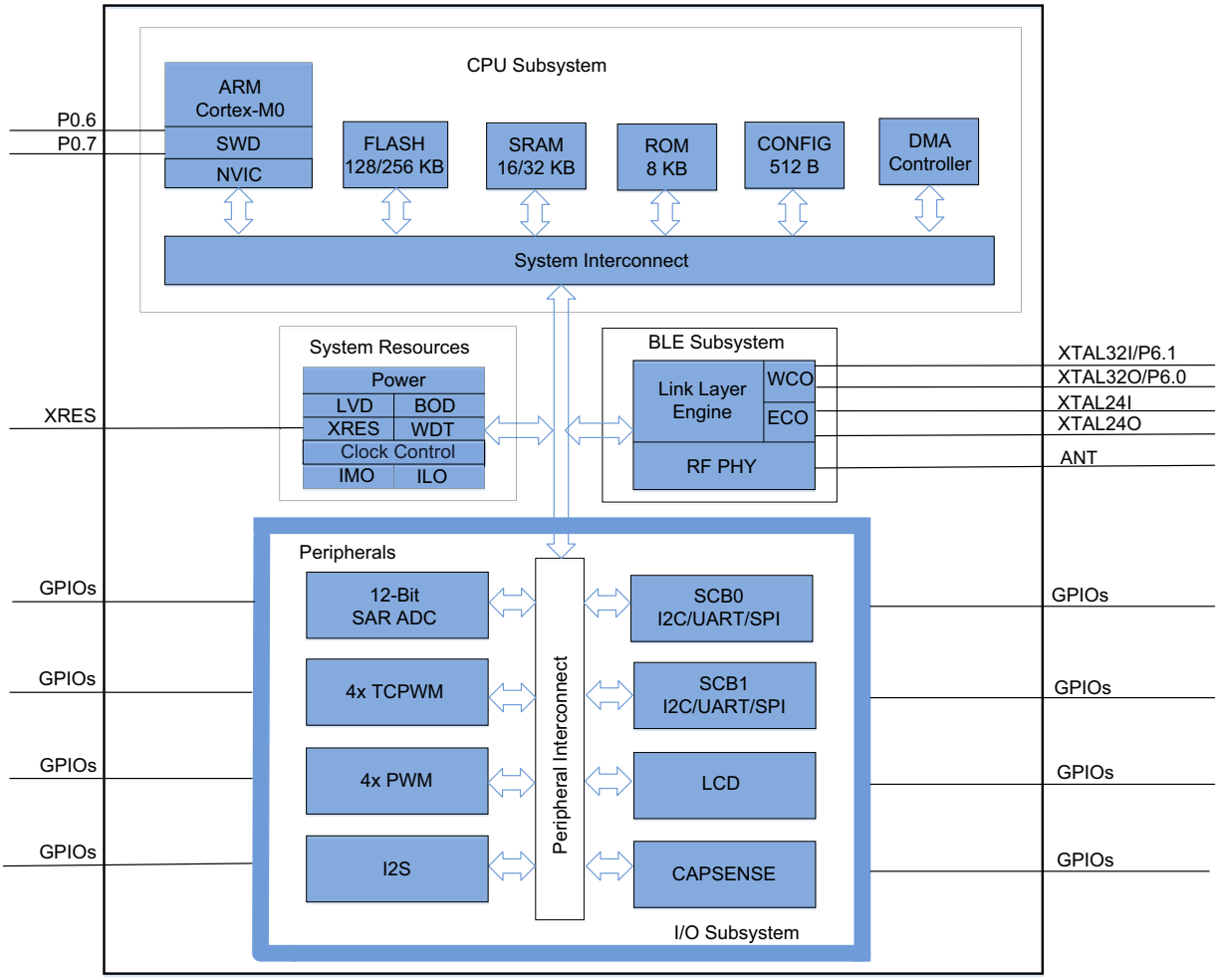
- High-performance, 32-bit single-cycle Cortex-M0 CPU core
- BLE radio and subsystem
  - On-chip BLE transceiver
  - Link layer controller compliant with Bluetooth 4.2 (with all new optional features) and backward-compatible with Bluetooth 4.1 and 4.0
- Fixed-function and configurable digital blocks
- Flexible and programmable interconnect
- Capacitive touch sensing (CapSense®)
- Direct memory access (DMA)

This document describes each function block of the PRoC BLE 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 PRoC BLE architecture.

Figure 1-1. PRoC BLE Family Block Diagram



1.2 Features

PRoC CYBL1XXXX family has these major components:

- BLE radio and subsystem
- 32-bit Cortex-M0 CPU with single-cycle multiply delivering up to 43 DMIPS at 48 MHz
- Up to 256 KB flash and 32 KB SRAM
- Direct memory access (DMA)
- Four independent center-aligned pulse width modulators (PWMs) with complementary dead-band programmable outputs and synchronized analog-to-digital converter (ADC) operation
- 12-bit, 1-Msps SAR ADC with internal reference, sample-and-hold (S/H), and channel sequencer
- Two serial communication blocks (SCB) to work as SPI/ UART/I2C serial communication channels
- CapSense and segment LCD drive
- Low-power operating modes: Sleep, Deep-Sleep, Hibernate, and Stop
- Programming and debug system through serial wire debug (SWD)
- Fully supported PSoC Creator™ IDE tool

1.3 CPU System

1.3.1 Processor

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

The PRoC BLE includes a hardware multiplier that provides a 32-bit result in one cycle.

1.3.2 Interrupt Controller

The CPU subsystem of PRoC BLE includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode. The Cortex-M0 CPU of PRoC BLE implements a non-maskable interrupt (NMI) input, which can be tied to digital routing for general-purpose use.

1.3.3 Direct Memory Access (DMA)

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

1.4 Memory

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

1.4.1 Flash

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

1.4.2 SRAM

The PRoC BLE provides SRAM, which is retained during Hibernate mode.

1.5 System-Wide Resources

1.5.1 Clocking System

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

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the PRoC BLE. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a low-power, less accurate oscillator and is used as a source for LFCLK, to generate clocks for peripheral operation in Deep-Sleep mode. Its clock frequency is 32 kHz with ±60 percent accuracy.

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

The ECO is used to generate a highly accurate 24-MHz clock without any external components. It is primarily used to clock the BLE subsystem, which contains the Link Layer engine, the digital PHY modem, and the RF transceiver. The high-accuracy ECO clock can also be used as a clock source for the PRoC BLE device.

The WCO is used as a source for LFCLK. WCO is used to accurately maintain the time interval of advertising events and connection events during low-power sleep mode. Similar to the ILO, WCO is also available in all modes, except Hibernate and Stop modes.

1.5.2 Power System

The PRoC BLE operates with a single external supply in the range 1.9 V to 5.5 V. PRoC BLE has four low-power modes – Sleep, Deep-Sleep, Hibernate, and Stop – besides the default Active mode.

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

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

1.5.3 GPIO

Every GPIO in PRoC BLE has the following characteristics:

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

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

1.6 Bluetooth Low Energy Subsystem

PRoC CYBL1XXXX Bluetooth Low Energy (BLE) subsystem integrates the RF transceiver, digital PHY modem, and link layer controller.

1.6.1 RF Transceiver

The RF transceiver contains an integrated balun, which provides a single-ended RF port pin to drive a 50-ohm antenna via a matching/filtering network. In the receive direction, this block converts the RF signal from the antenna to a 1-MHz intermediate frequency and digitizes the analog signal to 10-bit digital signal. In the transmit direction, this block takes 1 Mbps GFSK modulated from digital PHY, up-converts it to radio frequency, and transmit it to air through antenna.

1.6.2 Digital PHY Modem

In the transmit direction, this sub-block takes the 1-Mbps serial data from the link layer controller, generates GFSK direct modulated data, and sends it to the BLE analog section. On the receive side, it takes the 1-MHz IF ADC data from the BLE analog section and uses digital demodulator to generate the 1-Mbps serial data.

1.6.3 Link Layer Controller

The link layer controller implements all timing critical functions specified in the Bluetooth Low Energy Link Layer specifications (packet framing/de-framing, CRC generation/checking, encryption/decryption, state machines, and packet transmission); it also provides interface to the digital PHY. The communication between link layer hardware and firmware is done through interrupt, FIFO, and registers.

1.7 Fixed-Function Digital

1.7.1 Timer/Counter/PWM Block

The Timer/Counter/PWM block consists of four16-bit counters with user-programmable period length. The functionality of these counters can be synchronized. Each block has a capture register, period register, and compare registers. The block supports complementary dead-band programmable outputs. It also has a Kill input to force outputs to a predetermined state. Other features of the block include center-aligned PWM, clock pre-scaling, pseudo random PWM, and quadrature decoding.

1.7.2 Serial Communication Blocks

The PRoC CYBL1XXXX has two SCBs, which can each implement a serial communication interface as I2C, universal asynchronous receiver/transmitter (UART), or serial peripheral interface (SPI). The PRoC BLE 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 for each SCB include:

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

1.8 Analog System

1.8.1 SAR ADC

PRoC CYBL1XXXX has a configurable 12-bit 1-Msps SAR ADC. With a gain error of ±0.1 percent, integral nonlinearity (INL) less than 1 LSB, differential nonlinearity (DNL) less than 1 LSB, and signal-to-noise ratio (SNR) better than 68 dB, this convertor addresses a wide variety of analog applications.

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

1.9 Special Function Peripherals

1.9.1 LCD Segment Drive

The PRoC CYBL1XXXX has an LCD controller, which can drive up to four commons and every GPIO can be configured to drive common or segment. It uses full digital methods (digital correlation and PWM) to drive the LCD segments, and does not require generation of internal LCD voltages.

1.9.2 CapSense

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

1.9.2.1 IDACs and Comparator

The CapSense block has two IDACs and a comparator with a 1.2-V reference, which can be used for general purposes, if CapSense is not used.

1.10 Program and Debug

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

2. Getting Started

2.1 Support

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

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

2.2 Product Upgrades

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

2.3 Development Kits

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

2.4 Application Notes

Refer to application note *Getting Started with PRoC BLE* for additional information on PRoC BLE device capabilities and to quickly create a simple BLE application using PSoC Creator and BLE development kit.

3. Document Construction

The following sections in this document include these topics:

- Section B: CPU System on page 31
- Section C: System-Wide Resources on page 63
- Section D: Digital System on page 105
- Section E: Analog System on page 195
- Section F: Program and Debug on page 259

3.1 Major Sections

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

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

3.2 Documentation Conventions

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

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

3.2.1 Register Conventions

Register conventions are detailed in the PRoC BLE Registers Technical Reference Manual.

3.2.2 Numeric Naming

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

3.2.3 Units of Measure

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

Table 3-1. Units of Measure

Symbol	Unit of Measure
bps	bits per second
°C	degrees Celsius
dB	decibels
dBm	decibels-milliwatts
fF	femtofarads
G	Giga
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

Symbol	Unit of Measure
ABUS	analog output bus
AC	alternating current
ADC	analog-to-digital converter
ADV	advertising
AES	Advanced Encryption Standard
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an ARM data transfer bus
API	application programming interface
APOR	analog power-on reset
BC	broadcast clock
BLE	Bluetooth Low Energy (Bluetooth Smart)
BLESS	BLE subsystem
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
DAC	digital-to-analog converter
DC	direct current
DI	digital or data input
DNL	differential nonlinearity
DO	digital or data output
DSI	digital signal interface
DSM	deep-sleep mode
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
GAP	generic access profile
GATT	generic attribute profile
GFSK	Gaussian frequency-shift keying

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
GPIO	general purpose I/O
HCI	host-controller interface (BLE stack)
HFCLK	high-frequency clock
I^2C	inter-integrated circuit
IDE	integrated development environment
ILO	internal low-speed oscillator
IMO	internal main oscillator
INL	integral nonlinearity
I/O	input/output
IOR	I/O read
IOW	I/O write
IRES	initial power on reset
IRA	interrupt request acknowledge
IRQ	interrupt request
ISR	interrupt service routine
IVR	interrupt vector read
L2CAP	logical link control and adaptation protocol
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
PC	program counter
PCH	program counter high
PCL	program counter low
PD	power down
PDU	protocol data unit
PGA	programmable gain amplifier
PHY	physical layer
PM	power management
PMA	PRoC memory arbiter
POR	power-on reset
PPOR	precision power-on reset
PRoC®	Programmable Radio-on-Chip
PRS	pseudo random sequence
PSoC®	Programmable System-on-Chip
PSRR	power supply rejection ratio
PSSDC	power system sleep duty cycle

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
PWM	pulse width modulator
RAM	random-access memory
RETI	return from interrupt
RF	radio frequency
ROM	read only memory
RW	read/write
SAR	successive approximation register
SC	switched capacitor
SCB	serial communication block
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SYSCLK	system clock
SWD	single wire debug
TC	terminal count
TD	transaction descriptors
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
USBIO	USB I/O
WDT	watchdog timer
WDR	watchdog reset
XRES	external reset
XRES_N	external reset, active low

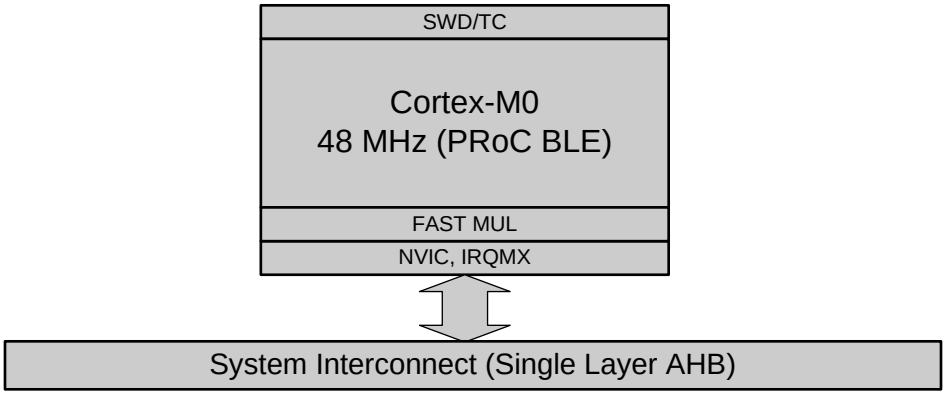
Section B: CPU System

This section encompasses the following chapters:

- Cortex-M0 CPU chapter on page 33
- Interrupts chapter on page 53

Top Level Architecture

CPU System Block Diagram



4. Cortex-M0 CPU

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

The Cortex-M0 processor includes a number of other components that are tightly linked to the CPU core. These include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

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

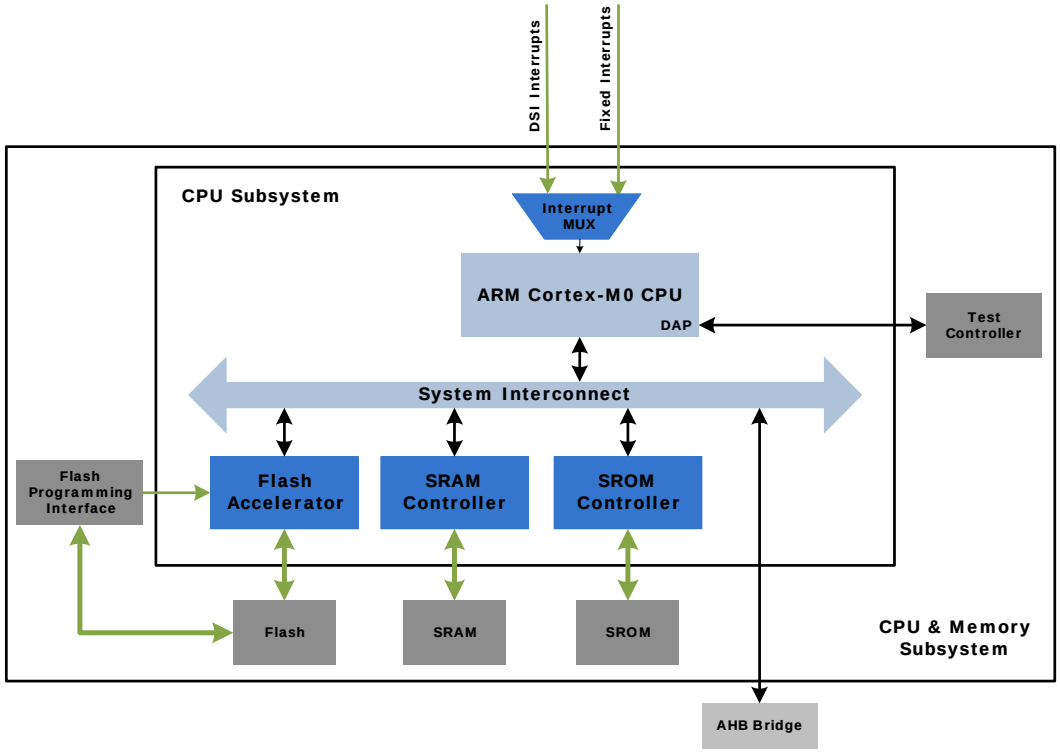
4.1 Features

The PRoC Cortex-M0 has the following features:

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

4.2 Block Diagram

Figure 4-1. PRoC 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 36). 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	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	PRoC 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) PSP (R13)	RW	[0x00000000]	The stack pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use: 0 = Main stack pointer (MSP). This is the reset value. 1 = Process stack pointer (PSP). On reset, the processor loads the MSP with the value from address 0x00000000.
LR (R14)	RW	Undefined	The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004]	The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.
PSR	RW	Undefined	The program status register (PSR) combines: Application Program Status Register (APSR). Execution Program Status Register (EPSR). Interrupt Program Status Register (IPSR).
APSR	RW	Undefined	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	[0x00000004].0	On reset, EPSR is loaded with the value bit[0] of the register [0x00000004].
IPSR	RO	0	The IPSR contains the exception number of the current ISR.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL register controls the stack used when the processor is in Thread mode.

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

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

Table 4-3. Cortex-M0 PSR Bit Assignments

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

Table 4-3. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
27 – 25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23 – 6	–	–	Reserved
5 – 0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = NMI 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 47 = IRQ31

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the Interrupts chapter on page 53 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 the Thread mode, the MSP or PSP can be used. The CONTROL register bit 1 determines which stack pointer is used:
  - 0 = MSP is the current stack pointer
  - 1 = PSP is the current stack pointer
- Handler Mode – used to execute exception handlers. The MSP is always used.

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

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

4.7 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}	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

Table 4-4. Thumb Instruction Set

Mnemonic	Brief Description
LDRB	Load Register with word
LDRH	Load Register with half-word
LDRSB	Load Register with signed byte
LDRSH	Load Register with signed half-word
LSLS	Logical Shift Left
LSRS	Logical Shift Right
MOV{S}	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result
MVNS	Bit wise NOT
NOP	No Operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-Reverse word
REV16	Byte-Reverse packed half-words
REVSH	Byte-Reverse signed half-word
RORS	Rotate Right
RSBS	Reverse Subtract
SBCS	Subtract with Carry
SEV	Send Event
STM	Store Multiple registers, increment after
STR	Store Register as word
STRB	Store Register as byte
STRH	Store Register as half-word
SUB{S}	Subtract
SVC	Supervisor Call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND based test
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait For Event
WFI	Wait For Interrupt

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

4.8 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

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

5. DMA Controller Modes

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

- Supports up to eight DMA channels; consult the device datasheet to determine how many channels are supported for a particular device
- Four levels of priority for each channel
- Byte, half-word (2 bytes), and word (4 bytes) transfers
- Three modes of operation supported for each channel
- Configurable interrupt generation
- Output trigger on completion of transfer
- Transfer sizes up to 65,536 data elements

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

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

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

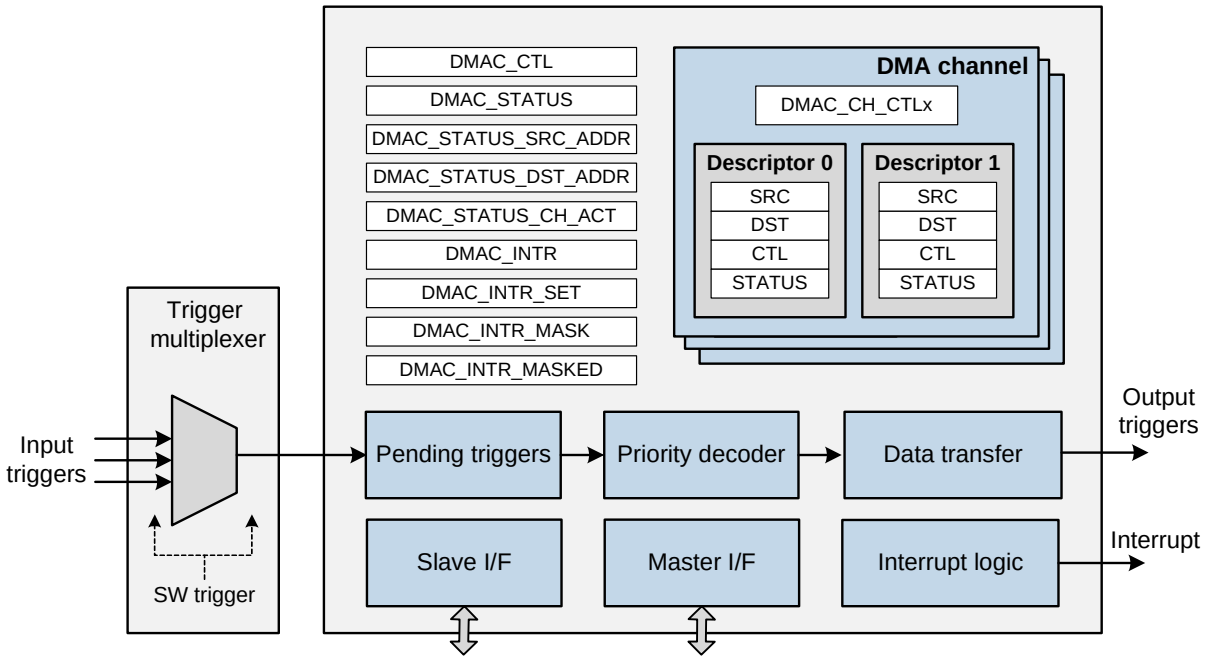
The DMA controller provides Active/Sleep functionality and is not available in the Deep-Sleep and Hibernate power modes.

5.1 Block Diagram Description

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

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

Figure 5-1. DMA Controller Block Diagram



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

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

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

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

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

5.1.1 Trigger Sources and Multiplexing

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

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

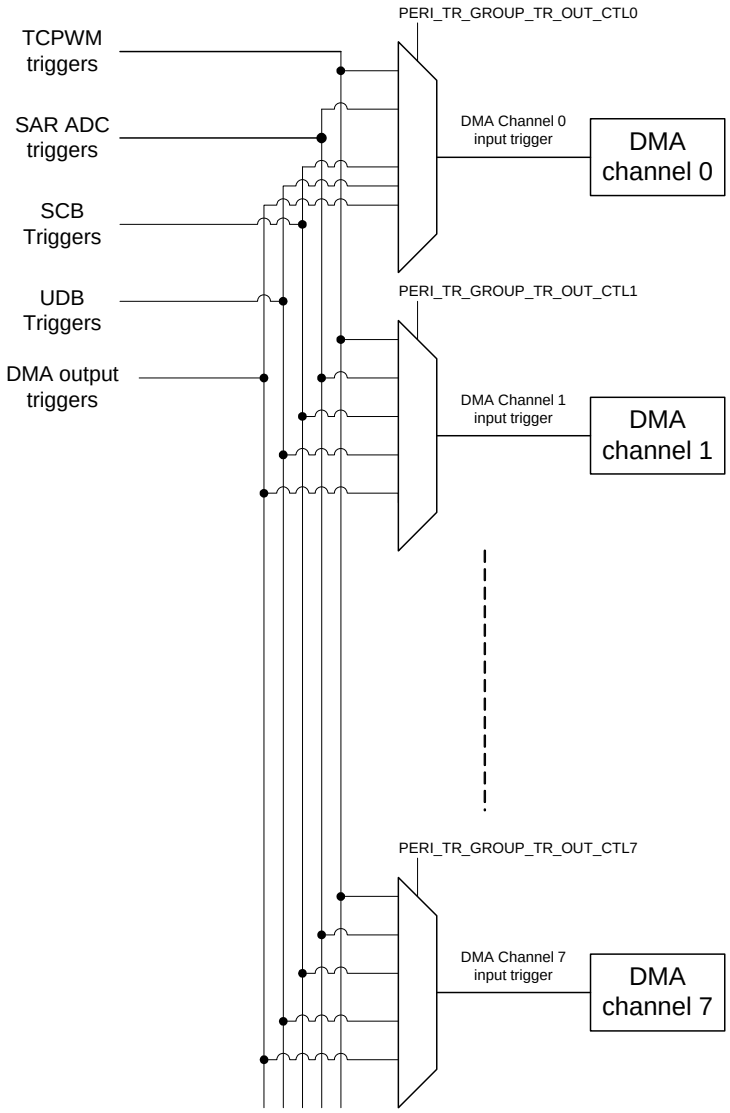
5.1.1.1 Trigger Multiplexer

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

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

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

Figure 5-2. Trigger Multiplexer Implementation



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

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
0	Software trigger hardwired to zero
1	TCPWM 0 overflow
2	TCPWM 1 overflow
3	TCPWM 2 overflow
4	TCPWM 3 overflow
5	TCPWM 0 Compare
6	TCPWM 1 Compare
7	TCPWM 2 Compare

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTL x[5:0]	Trigger Source
8	TCPWM 3 Compare
9	TCPWM 0 Underflow
10	TCPWM 1 Underflow
11	TCPWM 2 Underflow
12	TCPWM 3 Underflow
13	SAR ADC output
14	SCB0 TX
15	SCB0 RX
16	SCB1 TX
17	SCB1 RX
18	UDB DSI request 0

Table 5-1. Trigger Sources

PERI_TR_GROUP_TR_OUT_CTLx[5:0]	Trigger Source
19	UDB DSI request 1
20	DMA channel 0 trigger out
21	DMA channel 1 trigger out
22	DMA channel 2 trigger out
23	DMA channel 3 trigger out
24	DMA channel 4 trigger out
25	DMA channel 5 trigger out
26	DMA channel 6 trigger out
27	DMA channel 7 trigger out

5.1.1.2 Creating Software Triggers

Every DMA channel has a trigger input and output trigger associated with it. This trigger input can come from any trigger group, as described in Trigger Multiplexer on page 40. A software trigger for the DMA channel is implemented using the trigger input option 0 in the trigger multiplexer settings. When PERI_TR_GROUP_TR_OUT_CTLx [5:0] is zero, the DMA trigger is configured for a software trigger. The DMA channel is then triggered using the PERI_TR_CTL register.

5.1.2 Pending Triggers

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

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

5.1.3 Output Triggers

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

Note that the DMA output triggers also connect to digital system interconnects (DSI) and some DSI signals connect to the trigger multiplexer inputs.

5.1.4 Channel Prioritization

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

5.1.5 Data Transfer Engine

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

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

  Depending on the Transfer mode, State 2 and 3 may be performed multiple times.
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two cycles of the trigger activation. If it was set to ‘wait indefinitely’, the DMA controller will remain in this state until the trigger signal has gone low.
- State 6 - Storing Descriptor Response: In this phase, the data transfer according to the descriptor is completed and an interrupt may be generated if it was configured to do so. The Response field in DMAC_DESCR_PING_STATUS or DMAC_DESCR_PONG_STATUS is also populated and the state transitions to State 0.

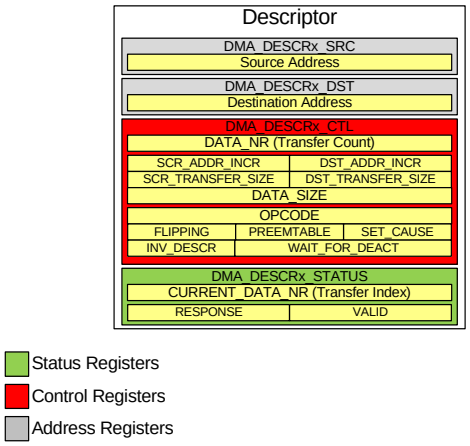
5.2 Descriptors

The data transfer between a source and a destination in a channel is configured using a descriptor. Each channel in

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

Figure 5-3 shows the structure of a descriptor.

Figure 5-3. Descriptor Structure



5.2.1 Address Configuration

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

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

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

Source Address Increment (SCR_ADDR_INC): This is a bit setting in the control register, which determines if a source address is incremented between each data element transfer. This feature is enabled when the source of the data is a buffer and each transfer element needs to be fetched from subsequent locations in the memory. In this case, the Source Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the SCR_TRANSFER_SIZE setting described in 5.2.2 Transfer Size on page 44.

Destination Address Increment (DST_ADDR_INC): This is a bit setting in the control register, which determines if a destination address is incremented between each element transfer. This feature is enabled when the destination of the data is a buffer and each transfer element needs to be transferred to subsequent locations in the memory. In this case, the Destination Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the DST_TRANSFER_SIZE setting described in 5.2.2 Transfer Size on page 44.

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

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

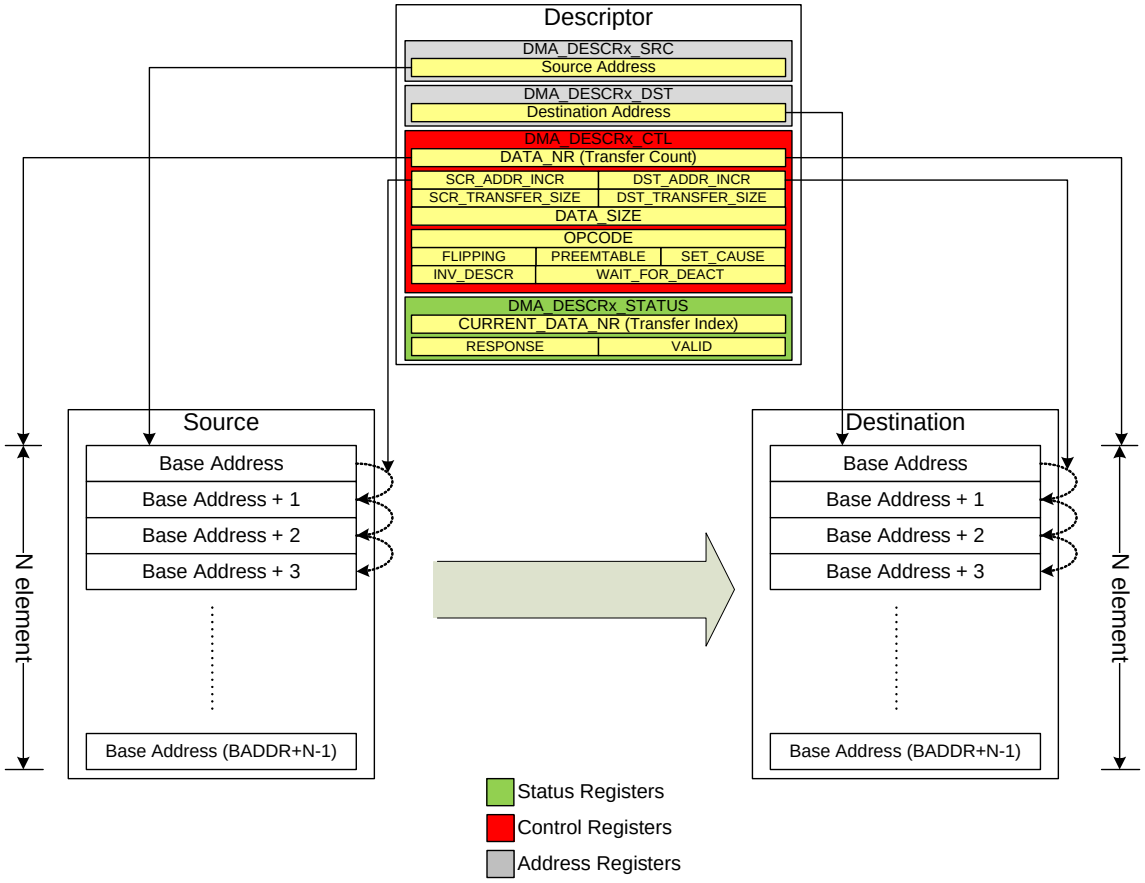
Setting Interrupt Cause (SET_CAUSE): When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

Trigger Type (WAIT_FOR_DEACT): When the DMA transfer based on the descriptor is completed, the data transfer engine checks the state of trigger deactivation. This is corresponding to State 5 of the data transfer engine. See 5.1.5 Data Transfer Engine on page 42. The type of DMA input trigger will determine when the trigger signal is considered deactivated. The DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger state is deactivated. This field is used to synchronize the controller's data transfer(s) with the agent that generated the trigger.

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

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

Figure 5-4. DMA Transfer: Address Configuration



5.2.2 Transfer Size

The transfer word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter (DATA_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes, set by SCR_TRANSFER_SIZE and DST_TRANSFER_SIZE, can have a value of either the DATA_SIZE or 32 bit. DATA_SIZE can be set to a 32-bit, 16-bit, or 8-bit setting.

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

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

Table 5-2. Transfer Size Settings

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

5.2.3 Descriptor Chaining

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

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

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

5.2.4 Transfer Mode

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

5.2.4.1 Single Data Element Per Trigger (OPCODE 0)

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

Figure 5-5 shows a typical use case of this transfer. Here an ADC's data register is the source and the destination is a peripheral register such as a PWM compare. The trigger is from the ADC's end-of-conversion signal. When the trigger is received, the transfer engine will load data from the ADC and store the lower 16 bits to the PWM register. Successive triggers will result in the same behavior because the descriptor is rerun.

Note how the source and destination data widths are assigned as 32 bit. This is because all accesses to peripheral registers in PRoC must be 32 bit. Because the valid data width is only 16 bit, the DATA_SIZE is maintained as 16 bit.

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

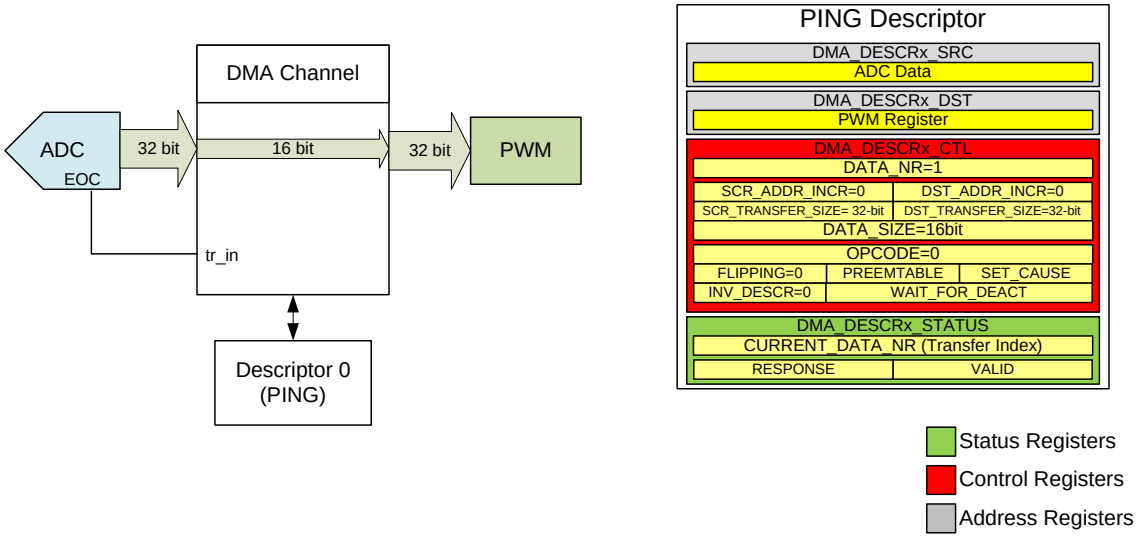
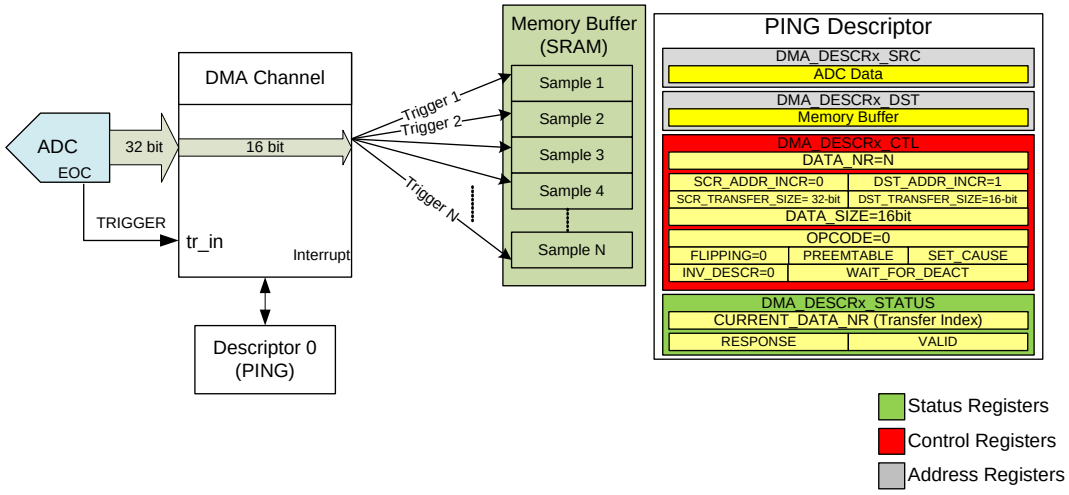


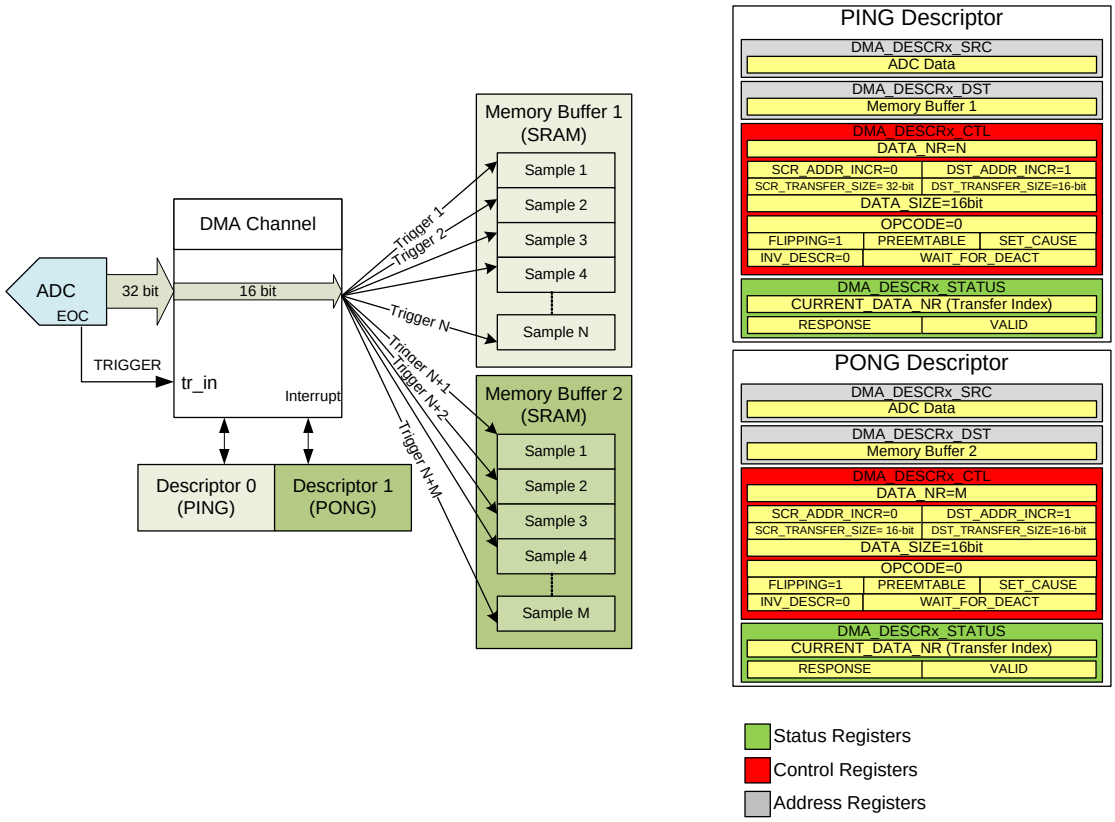
Figure 5-6 describes another use case where the data transfer is between the ADC data register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is an ADC. When the trigger is received, the transfer engine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

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



A similar use case is shown in Figure 5-7. This demonstrates the use of the PING and PONG descriptors. On completion of the PING descriptor, the controller will flip to executing the PONG descriptor. Note that the flipping bit is enabled in the descriptor; this enables the chaining of the descriptors. If the flipping bit was not enabled, the same descriptor will be re-run. Thus, two buffer transfers are achieved in sequence. However, note that the transfers are still done at one element transfer for every trigger.

Figure 5-7. OPCODE 0: Transfer Using Flipping Feature

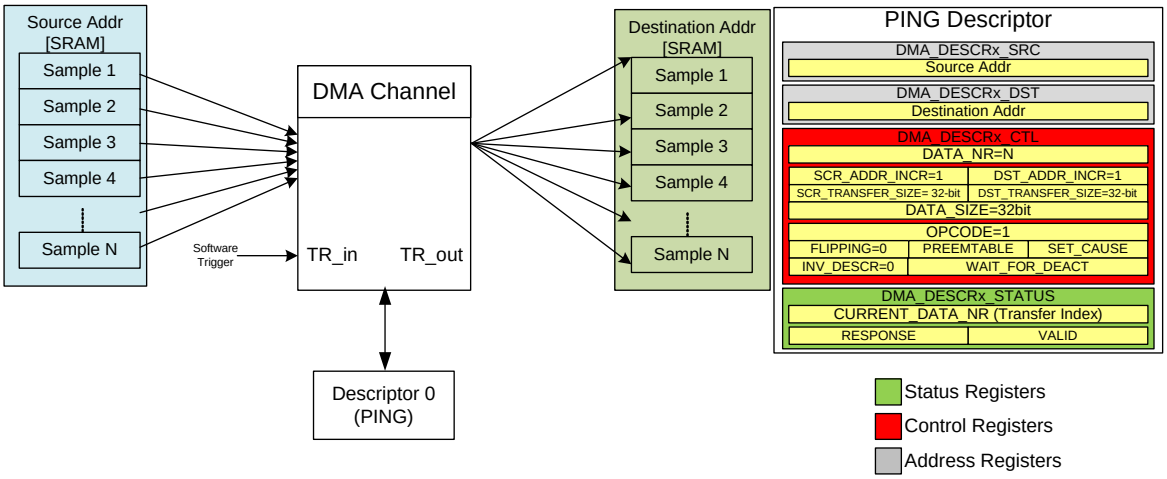


5.2.4.2 *Entire Descriptor Per Trigger (OPCODE 1)*

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

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

Figure 5-8. DMA Transfer Example with OPCODE 1

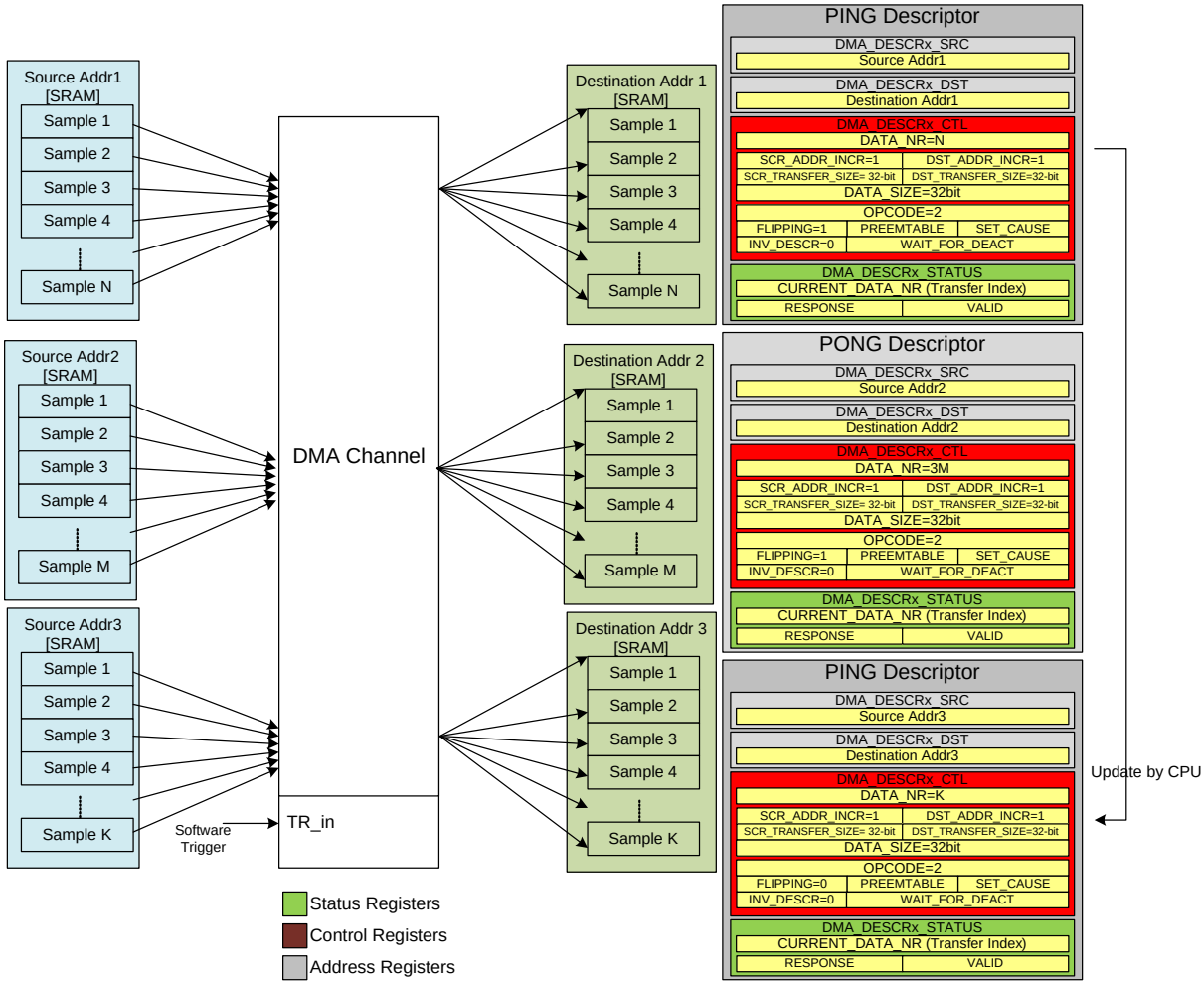


5.2.4.3 Entire Descriptor Chain Per Trigger (OPCODE 2)

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

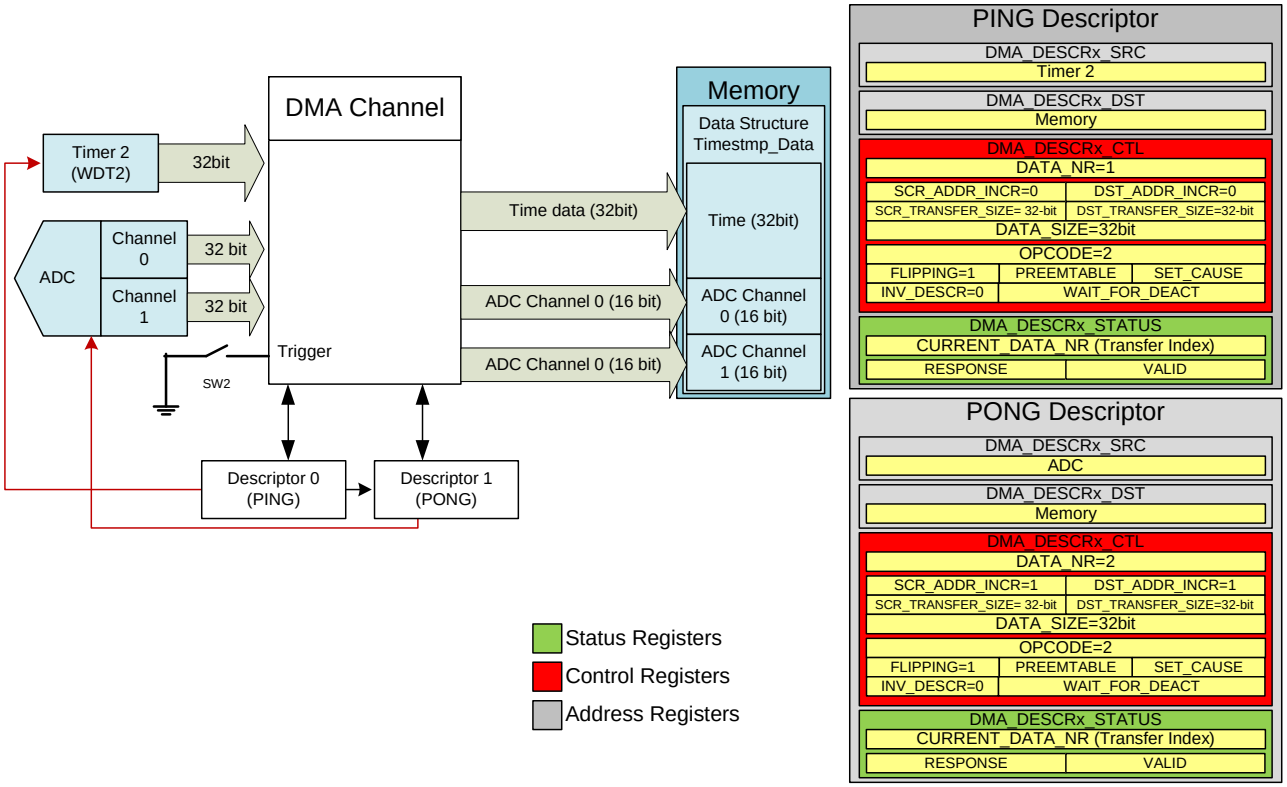
Figure 5-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

Figure 5-9. DMA Transfer Example with OPCODE 2



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

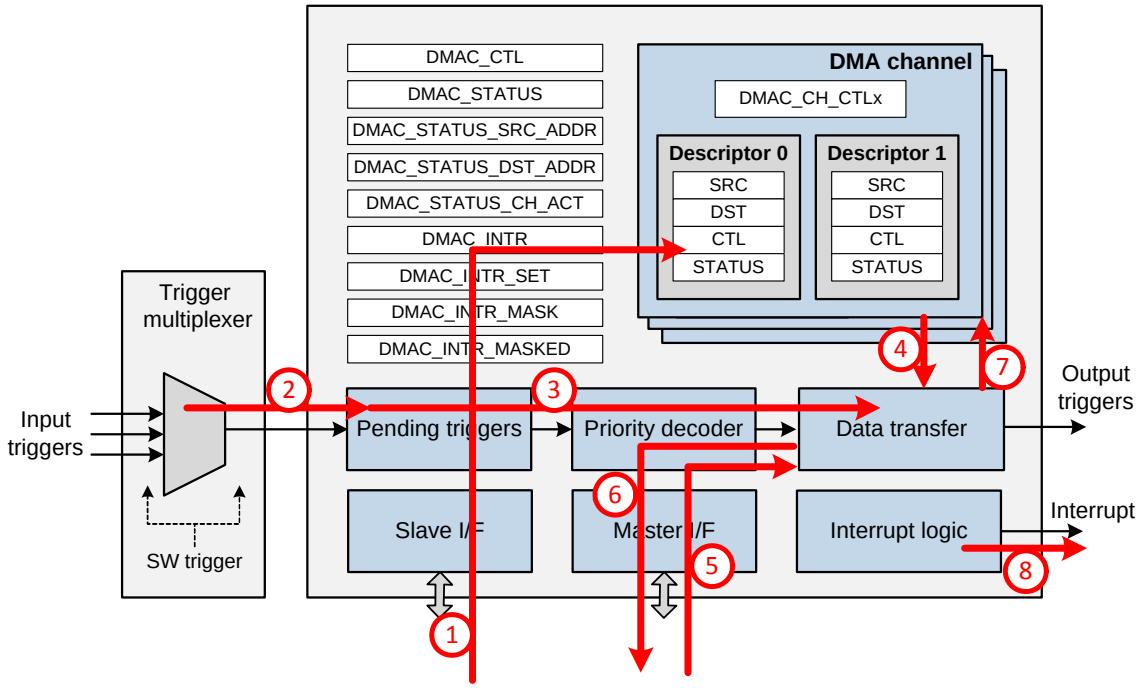
Figure 5-10. OPCODE 2: Multiple Sources to Memory



5.3 Operation and Timing

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

Figure 5-11. Operational Flow



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

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

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

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

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

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

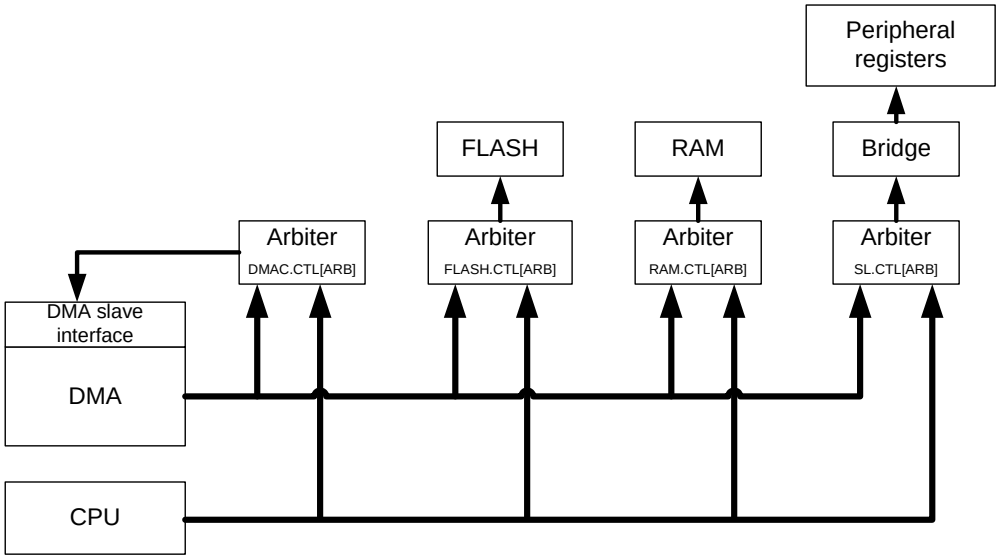
When transferring entire descriptors or chaining descriptor chains, 12 clock cycles are needed for the first data element. Subsequent elements need three cycles. This is also under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to transfer ‘N’ elements is:

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

5.4 Arbitration

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

Figure 5-12. PRoC 4 Bus Architecture

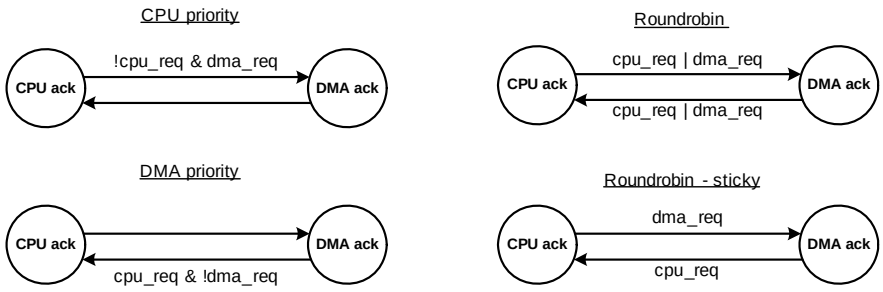


The arbitration policy for each slave can be one of the following:

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

The arbitration models are illustrated using the following diagrams.

Figure 5-13. Arbitration Models



5.5 Register List

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

6. Interrupts

The ARM Cortex-M0 (CM0) CPU in PRoC 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. PRoC provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

6.1 Features

PRoC supports the following interrupt features:

- Supports 32 interrupts
- Nested vectored interrupt controller (NVIC) integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

6.2 How It Works

Figure 6-1. PRoC Interrupts Block Diagram

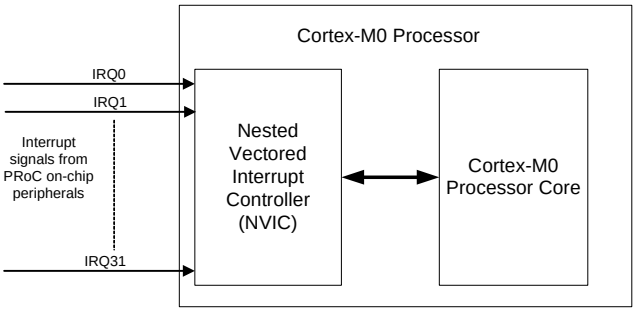


Figure 6-1 shows the interaction between interrupt signals and the Cortex-M0 CPU. PRoC has 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in Figure 6-1 because they are part of CM0 core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

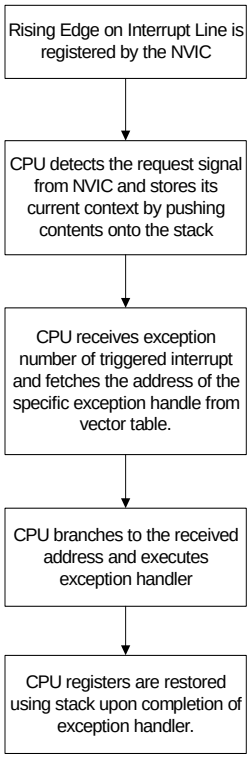
6.3 Interrupts and Exceptions - Operation

6.3.1 Interrupt/Exception Handling in PRoC

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 in PRoC have a unique exception number, as given in Table 6-1. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table.
4. The CPU then branches to this address and executes the exception handler that follows.
5. Upon completion of the exception handler, the CPU registers are restored to their original state using stack pop operations; the CPU resumes the main code execution.

Figure 6-2. Interrupt Handling When Triggered



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

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

6.3.2 Level and Pulse Interrupts

PRoC NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IRQ31). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 6-3. Level Interrupts

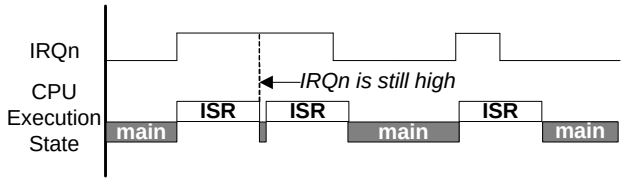


Figure 6-4. Pulse Interrupts

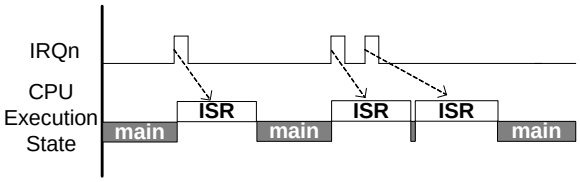


Figure 6-3 and Figure 6-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

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

6.3.3 Exception Vector Table

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

Table 6-1. PRoC Exception Vector Table

Exception Number	Exception	Exception Priority	Vector Address
–	Initial Stack Pointer Value	Not applicable (NA)	Base_Address - Can be 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM)[a]
1	Reset	–3, the highest priority	Base_Address + 0x04
2	Non Maskable Interrupt (NMI)	–2	Base_Address + 0x08
3	HardFault	–1	Base_Address + 0x0C
4-10	Reserved	NA	Base_Address + 0x10 to Base_Address + 0x28
11	Supervisory Call (SVCall)	Configurable (0 - 3)	Base_Address + 0x2C
12-13	Reserved	NA	Base_Address + 0x30 to Base_Address + 0x34
14	PendSupervisory (PendSV)	Configurable (0 - 3)	Base_Address + 0x38
15	System Timer (SysTick)	Configurable (0 - 3)	Base_Address + 0x3C
16	External Interrupt(IRQ0)	Configurable (0 - 3)	Base_Address + 0x40
…	…	Configurable (0 - 3)	…
47	External Interrupt(IRQ31)	Configurable (0 - 3)	Base_Address + 0xBC

a. Note that the reset exception address in SRAM vector table will never be used because the device comes out of reset with the flash vector table selected.

In Table 6-1, the first word (4 bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. In PRoC, 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.

The exception sources (exception numbers 1 to 15) are explained in 6.4 Exception Sources. The exceptions marked as Reserved in Table 6-1 are not used in PRoC, though they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 47) are explained in 6.5 Interrupt Sources.

6.4 Exception Sources

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

6.4.1 Reset Exception

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

6.4.2 Non-Maskable Interrupt (NMI) Exception

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

- **NMI exception due to a hardware signal (user NMI exception):** PRoC provides an option to trigger NMI exception using a digital signal. This digital signal is referred to as irq_out[0] in Table 6-2. The NMI exception triggered due to irq_out[0] will execute the NMI handler

pointed to by the active vector table (flash or SRAM vector table).

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations in PRoC such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

6.4.3 HardFault Exception

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

6.4.4 Supervisor Call (SVCall) Exception

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

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

6.4.5 PendSV Exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, 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.

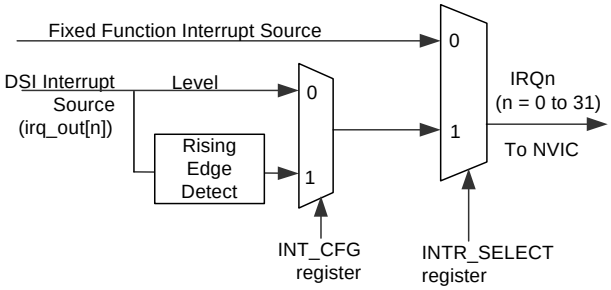
6.4.6 SysTick Exception

CM0 CPU in PRoC supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various time keeping purposes such as 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.

6.5 Interrupt Sources

PRoC supports 32 interrupts (IRQ0 - IRQ31 or exception numbers 16 - 47) from peripherals. The source of each interrupt is listed in Table 6-2. PRoC provides flexible sourcing options for each of the 32 interrupt lines. Figure 6-5 shows the multiplexing options for interrupt source. Each interrupt has two sources: a fixed-function interrupt source and a DSI interrupt source. The CPUSS_INTR_SELECT register is used to select between these sources.

Figure 6-5. Interrupt Source Multiplexing



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

The fixed-function interrupts include standard interrupts from the on-chip peripherals such as TCPWM, serial communication block and CSD block. The fixed-function interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. Fixed-function interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously.

The second category of interrupt sources is the DSI interrupt signals. There are eight DSI channels with each channel demultiplexed to four to spread across 32 interrupt sources for Cortex M0. Any digital signal on the chip, such as digital outputs from UDBs or digital input signals on pins, can be routed as DSI interrupt sources. This provides flexibility in the choice of interrupt sources. You also have the option of routing the DSI signal through a rising edge detect circuit, as shown in Figure 6-5. This edge detect circuit converts a rising edge signal on the DSI line to a pulse signal two system clocks wide. This ensures that the interrupt is triggered once on the rising edge of the signal on the DSI line. It is useful for interrupt sources, which cannot generate proper level interrupt signals to the NVIC. The UDB_INT_CFG register is used to select between the direct DSI path and the edge detect path.

As the DSI interrupt channels are demultiplexed, the maximum number of DSI interrupts, at a time, is limited to eight.

Interrupt	Cortex-M0 Exception No.	Fixed Function Interrupt Source	DSI Interrupt Source
NMI (see Exception Sources on page 55)	2	SYS_REQ	udb.interrupts[0]:4
IRQ0	16	GPIO Interrupt – Port 0	udb.interrupts[0]:0
IRQ1	17	GPIO Interrupt – Port 1	udb.interrupts[1]:0
IRQ2	18	GPIO Interrupt – Port 2	udb.interrupts[2]:0
IRQ3	19	GPIO Interrupt – Port 3	udb.interrupts[3]:0
IRQ4	20	GPIO Interrupt – Port 4	udb.interrupts[4]:0
IRQ5	21	GPIO Interrupt – Port 5	udb.interrupts[5]:0
IRQ6	22	GPIO Interrupt – All Port[a]	udb.interrupts[6]:0
IRQ7	23	LPCOMP (low-power comparator)	udb.interrupts[7]:0
IRQ8	24	WDT (Watchdog timer)	udb.interrupts[0]:1
IRQ9	25	SCB1 (Serial Communication Block 1)	udb.interrupts[1]:1
IRQ10	26	SCB2 (Serial Communication Block 2)	udb.interrupts[2]:1
IRQ11	27	CTBm interrupt (all CTBms)	udb.interrupts[3]:1
IRQ12	28	BLE SubSystem Interrupt	udb.interrupts[4]:1
IRQ13	29	SPCIF Interrupt	udb.interrupts[5]:1
IRQ14	30	SRSS LVD Interrupt	udb.interrupts[6]:1
IRQ15	31	SAR (Successive Approximation ADC)	udb.interrupts[7]:1
IRQ16	32	CSD (CapSense)	udb.interrupts[0]:2
IRQ17	33	TCPWM0 (Timer/Counter/PWM 0)	udb.interrupts[1]:2
IRQ18	34	TCPWM1 (Timer/Counter/PWM 1)	udb.interrupts[2]:2

Interrupt	Cortex-M0 Exception No.	Fixed Function Interrupt Source	DSI Interrupt Source
IRQ19	35	TCPWM2 (Timer/Counter/PWM 2)	udb.interrupts[3]:2
IRQ20	36	TCPWM3 (Timer/Counter/PWM 3)	udb.interrupts[4]:2
IRQ21	37	<DSI-only>	udb.interrupts[5]:2
IRQ22	38	<DSI-only>	udb.interrupts[6]:2
IRQ23	39	<DSI-only>	udb.interrupts[7]:2
IRQ24	40	<DSI-only>	udb.interrupts[0]:3
IRQ25	41	<DSI-only>	udb.interrupts[1]:3
IRQ26	42	<DSI-only>	udb.interrupts[2]:3
IRQ27	43	<DSI-only>	udb.interrupts[3]:3
IRQ28	44	<DSI-only>	udb.interrupts[4]:3
IRQ29	45	<DSI-only>	udb.interrupts[5]:3
IRQ30	46	<DSI-only>	udb.interrupts[6]:3
IRQ31	47	<DSI-only>	udb.interrupts[7]:3

a. Port 6 does not have the dedicated interrupt vector; it uses vector IRQ6.

See the I/O System chapter on page 65 for details on GPIO interrupts.

6.6 Exception Priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. PRoC provides flexibility in choosing priority values for different exceptions. All exceptions except Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In PRoC, 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.

PRoC 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 PRoC 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 55.

The priority of the 32 interrupts (IRQ0 - IRQ31) can be configured by writing to the Interrupt Priority registers (CM0_IPR). This is a group of eight 32-bit registers with each register storing the priority values of four interrupts, as given in Table 6-2. The other bit fields in the register are not used.

Table 6-2. Interrupt Priority Register Bit Definitions

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

6.7 Enabling/Disabling Interrupts

The NVIC provides registers to individually enable and disable the 32 interrupts in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that interrupt line. The Interrupt Set-Enable Register (CM0_ISER) and the Interrupt Clear-Enable Register (CM0_ICER) are used to enable and disable the interrupts respectively. These registers are 32-bit wide and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 6-3 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-3. Interrupt Enable/Disable Registers

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

Table 6-3. Interrupt Enable/Disable Registers

Interrupt Clear Enable Register (CM0_ICER)	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The CM0_ISER and CM0_ICER registers are applicable only for the interrupts (IRQ0 - IRQ31). These registers cannot be used to enable or disable the exception numbers 1 to 11. The 15 exceptions have their own support for enabling and disabling, as explained in Exception Sources on page 55.

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

6.8 Exception States

Each exception can be in one of the following states.

Table 6-4. Exception States

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

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

- The VECTACTIVE bits ([8:0]) in the CM0_ICSR store the exception number for the current executing exception. This value is zero if the CPU is not executing any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0_ICSR 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 IRQ31) is in a pending state.

6.8.1 Pending Exceptions

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

The NVIC allows software pending of the 32 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending Register (CM0_ISPR) and the Interrupt Clear-Pending Register (CM0_ICPR) are used to set and clear the pending status of the interrupt lines. These registers are 32 bits wide, and each bit corresponds to the same numbered interrupt. Table 6-5 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-5. Interrupt Set Pending/Clear Pending Registers

Register	Operation	Bit Value	Comment
Interrupt Set-Pending Register (CM0_ISPR)	Write	1	To put an interrupt to pending state
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending
Interrupt Clear-Pending Register (CM0_ICPR)	Write	1	To clear a pending interrupt
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending

Setting the pending bit when the same bit is already set results in only one execution of the 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 32 peripheral interrupts (exception numbers 16-47). These registers cannot be used for pending the exception numbers 1 to 11. These 15 exceptions have their own support for pending, as explained in Exception Sources on page 55.

6.9 Stack Usage for Exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its general-purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the 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 33 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 done as part of the internal processor architecture (infocenter.arm.com/help/topic/com.arm.doc.ddi0419c/index.html).

6.10 Interrupts and Low-Power Modes

PRoC 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, Deep-Sleep, and Hibernate modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 89. Chip low-power modes have three categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available in the Active, Deep-Sleep, and Hibernate modes (GPIO interrupts, low-power comparators).
- Fixed-function interrupt sources that are available only in the Active and Deep-Sleep modes (watchdog timer interrupt, serial communication block interrupts, and BLE subsystem interrupts)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

6.11 Exception - Initialization and Configuration

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

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 will need to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents. PSoC Creator IDE uses the vector table in SRAM by default.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   b. Define the exception handler function and write the address of the function to the exception vector table. Table 6-1 gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
   c. Set up the exception priority, as explained in Exception Priority on page 58.
   d. Enable the exception, as explained in Enabling/Disabling Interrupts on page 58.

6.12 Registers

Table 6-6. 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
CPUSS_CONFIG	CPU Subsystem Configuration
CPUSS_SYSREQ	System Request Register
CPUSS_INTR_SELECT	Interrupt Multiplexer Select Register
UDB_INT_CFG	UDB Subsystem Interrupt Configuration

6.13 Associated Documents

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

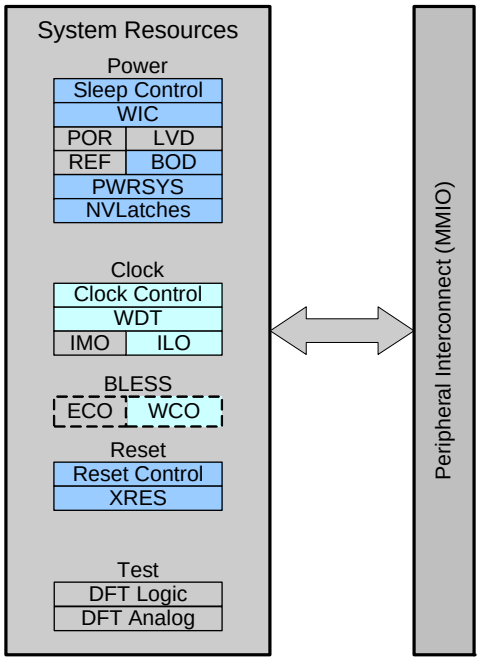
Section C: System-Wide Resources

This section encompasses the following chapters:

- I/O System chapter on page 65
- Clocking System chapter on page 73
- Power Supply and Monitoring chapter on page 83
- Chip Operational Modes chapter on page 87
- Power Modes chapter on page 89
- Watchdog Timer chapter on page 95
- Reset System chapter on page 99
- Device Security chapter on page 103

Top Level Architecture

System-Wide Resources Block Diagram



7. I/O System

This chapter explains the PRoC I/O system, its features, architecture, operating modes, and interrupts. The general-purpose I/O (GPIOs) pins in PRoC are grouped into ports; a port can have a maximum of eight GPIOs. CY8C4xxx_BLE has a maximum of 36 GPIOs, including two over-voltage tolerant pins, arranged in six ports.

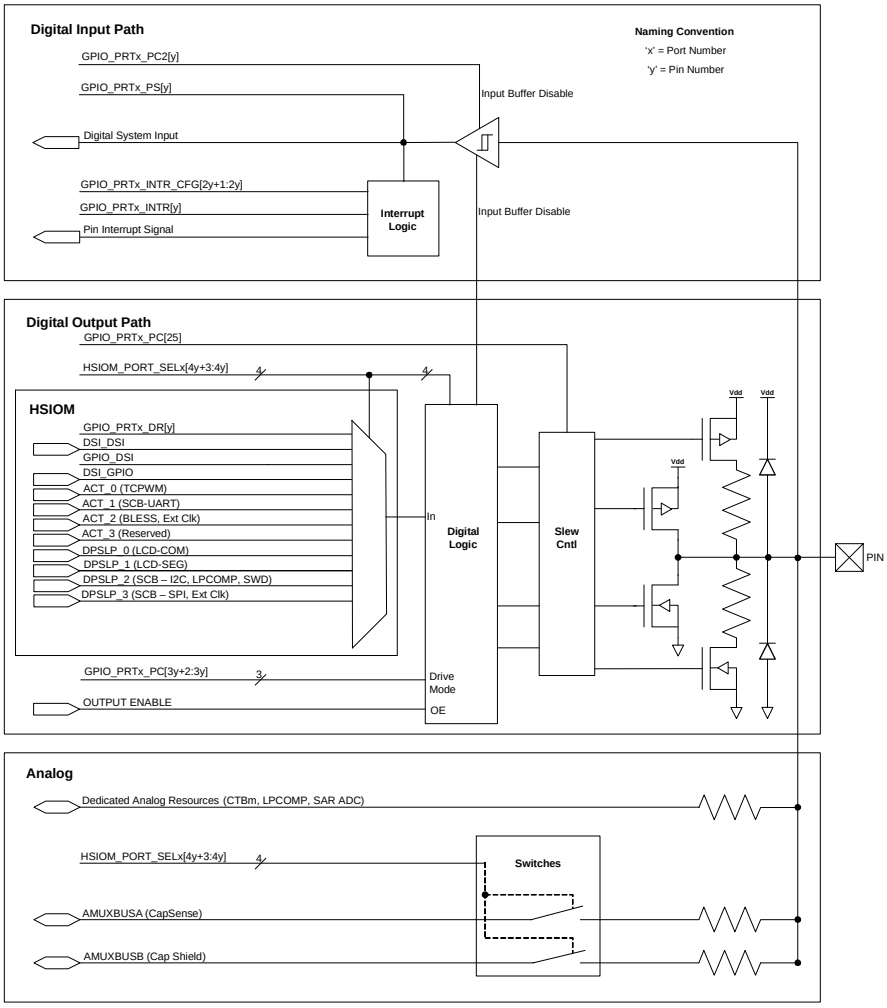
7.1 Features

The PRoC GPIOs have these features:

- Analog and digital input and output capabilities
- Segment LCD drive support
- CapSense support
- 8-mA sink and 4-mA source current in digital mode
- Separate port read (PS) and write (DR) data registers to avoid read-modify-write errors
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Two over-voltage tolerant (OVT-GPIO) pins for I^2C compliance

7.2 Block Diagram

Figure 7-1. GPIO Block Diagram



Note The GPIO features shown in this image may not be available on all the pins. See Table 7-4 for more details.

7.3 GPIO Drive Modes

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

Two port configuration registers are used to configure GPIOs in PRoC: Port Configuration Register (GPIO_PRTx_PC) and Port Secondary Configuration Register (GPIO_PRTx_PC2). All PRoC devices have ports with dedicated GPIO_PRTx_PC and GPIO_PRTx_PC2 registers.

GPIO_PRTx_PC is used to configure the following properties of a port:

- Output drive mode of each pin (three bits select a particular drive mode for a pin)
- Slew rate of the whole port (see Slew Rate Control on page 69)
- Input threshold selection of the whole port (see CMOS LVTTL Level Control on page 69)

GPIO_PRTx_PC2 is used to enable/disable the input buffer of each pin on the port, irrespective of the drive mode configured in GPIO_PRTx_PC. When analog signals are present on the pin, input buffer should be disabled by setting the bit to '1'.

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

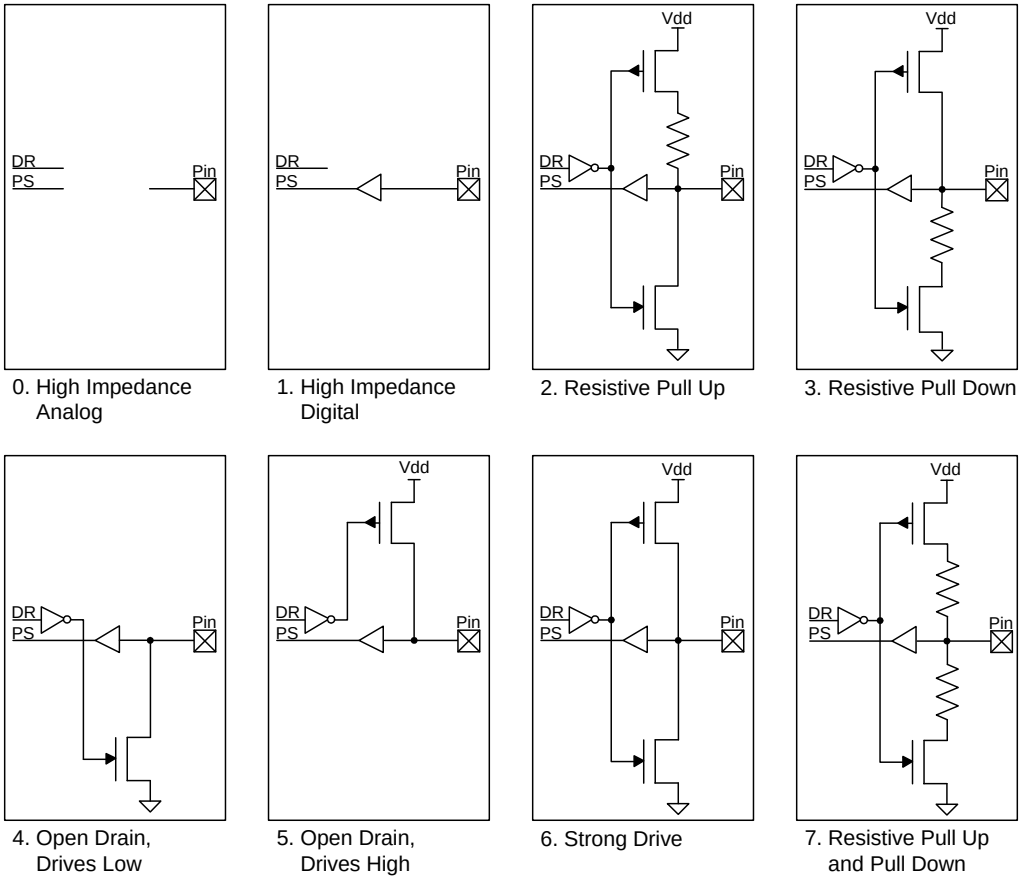


Table 7-1. Drive Mode Settings

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

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

GPIO_PRTx_PC2 ('x' denotes port number and 'y' denotes pin number)		
Bits	**Name**	**Description**
7:0	INP_DIS	Disables the input buffer independent of the port control drive mode. This bit should be set when analog signals are present on the pin.

7.3.1 High-Impedance Analog

High-impedance analog mode is the default reset state; both output driver and digital input buffer are turned off. This state prevents 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.

7.3.2 High-Impedance Digital

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

7.3.3 Resistive Pull-Up or Resistive Pull-Down

Resistive modes provide a series resistance in one of the data states and strong drive in the other. Pins can be used 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 PRoC 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.

7.3.4 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.

7.3.5 Strong Drive

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

7.3.6 Resistive Pull-Up and Resistive Pull-Down

This mode is similar to the drive modes explained in 7.3.3 Resistive Pull-Up or 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.

7.4 Slew Rate Control

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

7.5 CMOS LVTTL Level Control

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

Input level is individually configurable for each port. This bit is cleared by default and the port works in CMOS mode. This bit can be set to reconfigure the port to LVTTL mode.

CMOS mode can be used in most cases, whereas LVTTL can be used for custom interface requirements, which works at lower voltage levels. See the device datasheet for the input voltage thresholds (VIH and VIL) for the modes.

7.6 GPIO-OVT

The PRoC device has two over-voltage tolerant (OVT) pins – P5[0] and P5[1]. It is similar to regular GPIOs with the following additional features:

- Over-voltage tolerant
- Provides better pull-down drive strength
- Serial Communication Block (SCB) when configured as I^2C and its lines routed to GPIO-OVT pins; it meets the following I^2C specifications:
  - Fast Mode Plus IOL Specification
  - Fast Mode and Fast Mode Plus Hysteresis and minimum fall time specifications

See the device datasheet for specifications.

7.7 High-Speed I/O Matrix

High-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the resources inside PRoC. These resources include CapSense, TCPWMs, I2C, and the CPU. The HSIOM selects Active and Deep-Sleep power domain sources for a pin. HSIOM_PORT_SELx are 32-bit wide registers that control the routing of GPIOs. Each register controls one port; four dedicated bits are assigned to each GPIO in the port. This provides up to 16 different options for GPIO routing. This selection provides different pin functions, as listed in Table 7-3.

Table 7-3. 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 regular firmware-controlled I/O or connected to dedicated hardware block.
	DR_DSI	1	Output is firmware controlled, but OE is controlled from DSI.
	DSI_DSI	2	Both output and OE are controlled from DSI.
	DSI_DR	3	Output is controlled from DSI, but OE is firmware controlled.
	CSD_SENSE	4	Pin is a CSD sense pin (analog mode).
	CSD_SHIELD	5	Pin is a CSD shield pin (analog mode).
	AMUXA	6	Pin is connected to AMUXBUS-A.
	AMUXB	7	Pin is connected to AMUXBUS-B. This mode is also used for CSD I/O charging. When CSD I/O charging is enabled in CSD_CONTROL, digital I/O driver is connected to csd_charge signal (pin is still connected to AMUXBUS-B).
	ACT_0	8	Pin-specific Active source #0 (TCPWM).
	ACT_1	9	Pin-specific Active source #1 (SCB UART).
	ACT_2	10	Pin-specific Active source #2 (BLESS, EXT CLK)
	ACT_3	11	Reserved
	DPSLP_0	12	Pin-specific Deep-Sleep source #0 (LCD - COM)
	DPSLP_1	13	Pin-specific Deep-Sleep source #1 (LCD - SEG)
	DPSLP_2	14	Pin-specific Deep-Sleep source #2 (SCB-I2C, SWD, LPCOMP, Wakeup)
	DPSLP_3	15	Pin-specific Deep-Sleep source #3 (SCB SPI, EXT CLK)

Note The active and deep sleep sources are pin dependent. See the “Pinouts” section of the device datasheetfor more details on the features supported by each pin.

Note that when the UART and SPI inputs (SCB mode) are connected to a pin, the SCB controls the digital output buffer drive mode. For example, the SCB block disables the output buffer at the UART Rx pin for high-input impedance. This overrides the drive mode settings, which is done using GPIO_PRTx_PC register.

7.8 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, two other registers - GPIO_PRTx_DR_SET and GPIO_PRTx_DR_CLR - are provided to set or clear the output data of specific GPIOs in port. Additionally, the GPIO_PRTx_INV register can be used to invert the output data of a specific GPIO.

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

See the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM) for the details of these registers.

7.9 Analog I/O

Analog resources, such as LPCOMP, SARMUX, and CTBm, which require low-impedance routing paths have dedicated pins. Dedicated analog pins provide direct connections to specific analog blocks. Dedicated pins help improve performance and should be given priority over other pins when using these analog resources. See the device datasheetfor details on these dedicated pins of PRoC.

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

To configure a GPIO as an analog pin connecting to AMUXBUS, it should be configured in high-impedance analog mode (see Table 7-1) and then routed to AMUXBUS using the HSIOM_PORT_SELx register (Table 7-3).

7.10 LCD Drive

All GPIOs have the capability of driving an LCD common or segment. HSIOM_PORT_SELx registers are used to select the pins for LCD drive. See the LCD Direct Drive chapter on page 233 for details.

7.11 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 7-3 shows the GPIO and HSIOM settings required for CapSense. See the CapSense chapter on page 245 for more information.

Table 7-4. 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 a select GPIO)	High-Impedance Analog	Disable Buffer	AMUXBUS B or CSD_COMP
CSH TANK (GPIO precharge, only available in a select GPIO)	High-Impedance Analog	Disable Buffer	AMUXBUS B or CSD_COMP

7.12 Bluetooth Low Energy Sub-System (BLESS)

The Bluetooth SubSystem in the PRoC device has dedicated pins for the 32-kHz crystal, the 24-MHz crystal, and antenna connections. Refer to the device datasheet for more details.

7.13 I/O Port Reconfiguration

Drive mode and GPIO can be reconfigured in runtime by changing the value of the GPIO_PRTx_PC and HSIOM_PORT_SELx registers. Take care to retain the pin state during reconfiguration of pins when they are connected directly to a digital peripheral. If the ports are driven by the data registers, state maintenance is automatic.During port configuration, the current configuration should be saved as follows:

1. Read the GPIO pin state - GPIO_PRTx_PS in software.

2. Write the GPIO_PRTx_PS value into the data registers - GPIO_PRTx_DR.
3. Change the corresponding field in HSIOM_PORT_SELx to drive the pin by the data register - GPIO_PRTx_DR.

7.14 I/O State on Power Up

By default, during power up all GPIOs are in high-impedance analog state and input buffers are disabled. When the chip is powered, its GPIOs can be configured according to the required application, by writing to the associated registers.

7.15 Behavior in Low-Power Modes

The GPIOs maintain the current pin state during Sleep mode. In Sleep mode, all the GPIOs are active and can be driven by active peripherals, such as CapSense, TCPWM, and I2C.

To achieve the lowest device current in low-power modes, unused I/Os must be configured in the high-impedance analog mode.

7.16 GPIO Interrupt

This section describes the interrupt functionality of the PRoC GPIOs.

7.16.1 Features

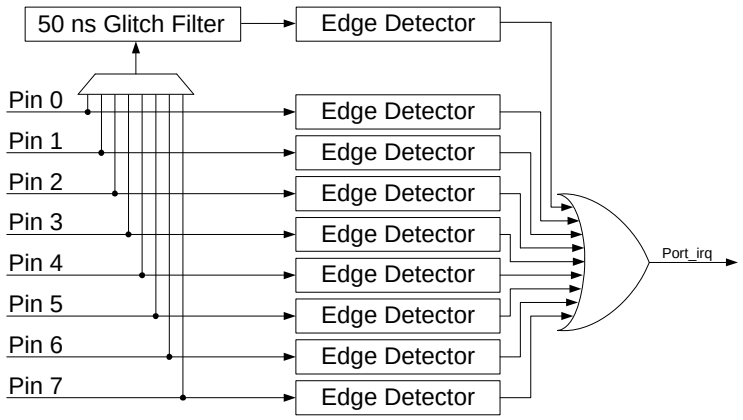
The features of the GPIO interrupt are:

- All eight pins in each port interface have an interrupt and an associated interrupt vector except port 6, which cannot be used to generate interrupts
- Pin status bits provide easy determination of interrupt source down to the pin level
- Rising, falling, or both edge-triggered interrupts are handled
- Pin interrupts can be individually enabled or disabled
- AHB interfaces for read and write into its registers
- Sends out a single port interrupt request (PIRQ) signal, derived from all GPIOs in a port, to the interrupt controller

7.16.2 Interrupt Controller Block Diagram

Each port has its own individual interrupt request and associated interrupt request (IRQ) vector and interrupt service routine (ISR). Additionally, one pin can be selected on each port that is routed through a 50-ns glitch filter to form a glitch-tolerant interrupt for the port. The details are shown in Figure 7-3.

Figure 7-3. Interrupt Generator



7.16.3 Function and Configuration

Each pin of the port, except port 6, can be configured independently to generate an interrupt on the rising edge, falling edge, or on both edges by writing to the GPIO_PRTx_INTR_CFG register. Level-sensitive interrupts are not supported. GPIO_PRTx_INTR_CFG is also used to route a specific channel to the glitch filter and generate a ninth glitch-tolerant interrupt.

When a GPIO interrupt is triggered by a signal on an interrupt-enabled port pin, the GPIO_PRTx_INTR register (Port Interrupt Status Register) is updated. The firmware can read this register to determine which GPIO triggered the interrupt. Firmware can then clear the IRQ bit by writing a '1' to its corresponding bit.

Additionally, when the Port Interrupt Control Status Register is read at the same time an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. 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.

See GPIO_PRTx_INTR_CFG and GPIO_PRTx_INTR in the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Refer-

ence Manual (TRM) for the details of these registers.

7.17 Input and Output Synchronization

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

This feature and other clock and reset features are implemented using a combination of UDB port adapter and I/O blocks.

7.18 Restrictions on Port 4 and Beyond

Port 4 and higher ports do not have a dedicated port-adapter. Therefore, none of the pins on these ports can be routed through the DSI. However, they can still be used as a firmware pin, LCD_COM, LCD_SEG, or CapSense, or can be connected to the SCB block through the HSIOM.

Because DSI is unavailable for these pins, their signals cannot be synchronized with any other clock source.

7.19 Registers

Table 7-5. 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 - Used to read 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 register name denotes the port number. For example, GPIO_PTR1_DR is the port 1 output data register.

8. Clocking System

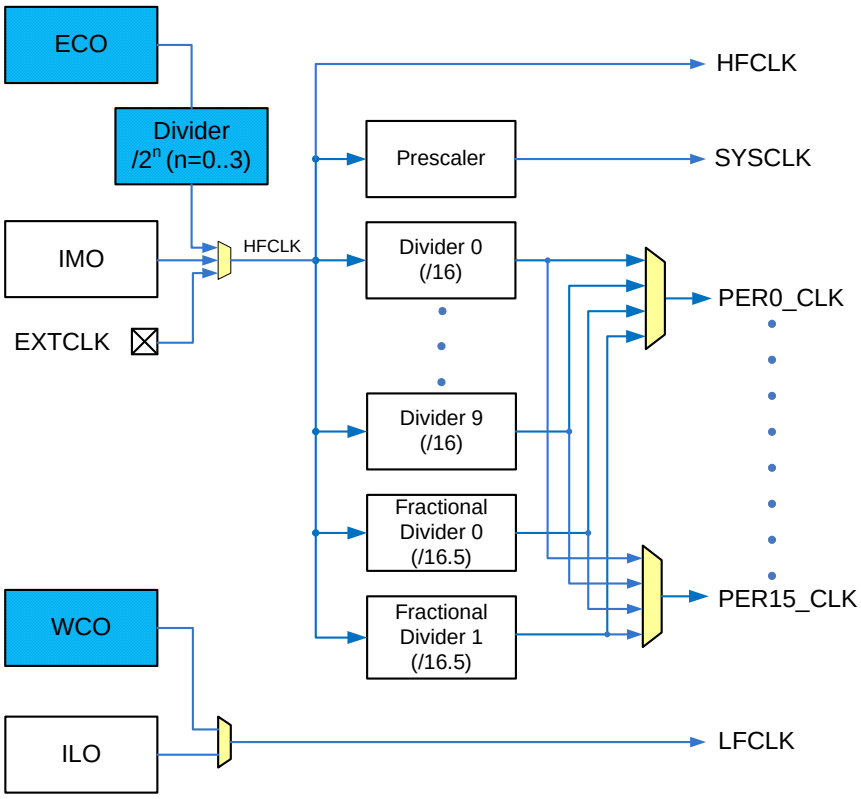
The PRoC clock system includes these clock resources:

- Two internal clock sources:
  - 3–48 MHz internal main oscillator (IMO) ±2 percent across all frequencies with trim
  - 32-kHz internal low-speed oscillator (ILO) ±60 percent with trim
- Three external clock sources:
  - External clock (EXTCLK) generated using a signal from an I/O pin
  - External 24-MHz crystal oscillator (ECO)
  - External 32-kHz watch crystal oscillator (WCO)
- High-frequency clock (HFCLK) of up to 48 MHz selected from IMO, ECO, or external clock
- Low-frequency clock (LFCLK) sourced by ILO or WCO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- Ten 16-bit peripheral clock dividers
- Two fractional dividers for accurate clock generation
- Sixteen digital and analog peripheral clocks

8.1 Block Diagram

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

Figure 8-1. Clocking System Block Diagram



The five clock sources in the device are shown in Figure 8-1, on the left. The ECO divider divides the ECO clock. The HFCLK mux selects the HFCLK source from an external clock source (EXTCLK), ECO, or the IMO. The SYSCLK prescaler generates the SYSCLK and the peripheral dividers generate the individual peripheral clocks.The LFCLK mux selects the LFCLK from the WCO or the ILO.

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 spanning 3–48 MHz in 1-MHz increments. Frequencies are selected by setting the frequency in the CLK_IMO_TRIM2 register, setting the IMO trim in the CLK_IMO_TRIM1 register, and finally setting the bandgap trim in PWR_BG_TRIM4 and PWR_BG_TRIM5 registers. 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, PWR_BG_TRIM4, and PWR_BG_TRIM5. Each PRoC device has IMO trim settings measured 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, PWR_BG_TRIM4, and PWR_BG_TRIM5. 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.

Table 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.

8.2.1.2 IMO Frequency Spread

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

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

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

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

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

The SS_MAX field in the CLK_IMO_SPREAD register sets the maximum count for the spread spectrum counters. Increasing this value increases the entire cycle time of a triangular spread when the SS_MODE is set to a triangular spread.

The IMO spread spectrum logic requires a clock to be routed to it for functionality. The logic uses the peripheral clock 0 as its clock. The IMO spread spectrum needs the peripheral clock 0 to be routed with an appropriate clock from a peripheral clock divider. The frequency of this clock will determine the rate of the spread spectrum logic and hence the rate of change of the frequency.

8.2.1.3 Programming Clock (36-MHz)

The IMO block has a 36-MHz output, which is used as clock for the flash programming block. This clock is only available for the flash programming block and is not available as a clock source into any of the clock dividers or the clock tree.

8.2.2 Internal Low-speed Oscillator

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

8.2.3 External Clock (EXTCLK)

The external clock (EXTCLK) is a MHz range clock that can be generated from a signal on a designated PRoC pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is 0–48 MHz. PRoC 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 65 for more details.

8.2.4 External Crystal Oscillator (ECO)

The external crystal oscillator with no external components is used to generate a highly accurate clock. It is primarily used to clock the BLE subsystem, which contains the Link Layer engine, the digital PHY modem, and the RF transceiver. The high-accuracy ECO clock can also be used as a clock source for the PRoC device. The ECO is enabled and disabled with the BLE_BLERD_DBUS register's XTAL_ENABLE bit.

8.2.4.1 ECO Load Capacitor Tuning

The PRoC device contains a variable integrated load capacitor to tune the clock frequency on a production line. Load capacitors can be independently controlled for X1 and X2 nodes with the BLE_BLERD_BB_XO_CAPTRIM register's

X2 and X1 bits, which are shown in Table 8-3.

Table 8-3. Capacitor Trim Selection Bits X2 and X1

Name	Description
X2[7:0]	Bits 7:0 together control capacitor value on X2 node. Bits 6-0: Fine control Value of 6-0: Cap value 0: 3.6900 pF 1: 3.7911 pF 2: 3.8922 pF . . . 127: 16.4280 pF Bit 7: Coarse control 0: No additional cap turned on 1: Additional cap of 8.1 pF turned on
X1[15:8]	Bits 7:0 together control capacitor value on X1 node. Bits 6-0: Fine control Value of 6-0: Cap value 0: 3.6900 pF 1: 3.7911 pF 2: 3.8922 pF . . . 127: 16.4280 pF Bit 7: Coarse control 0: No additional cap turned on 1: Additional cap of 8.1 pF turned on

8.2.5 Watch Crystal Oscillator (WCO)

The watch crystal oscillator (WCO) is used as one of the sources for LFCLK. WCO is used to accurately maintain the advertising events and connection events time interval during low-power sleep mode. Similar to ILO, WCO is also available in all modes except Hibernate and Stop modes. The WCO is enabled and disabled with the WCO_CONFIG register's ENABLE bit.

8.3 Clock Distribution

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

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

8.3.1 HFCLK Input Selection

HFCLK in PRoC has three input options: IMO, EXTCLK, and ECO. The HFCLK input is selected using the CLK_SELECT register's DIRECT_SEL bits, as described in Table 8-4.

Table 8-4. HFCLK Input Selection Bits DIRECT_SEL

Name	Description
DIRECT_SEL[2:0]	HFCLK input clock selection 0: IMO. Uses the IMO as the source of the HFCLK 1: EXTCLK. Uses the EXTCLK as the source of the HFCLK 2: ECO. Uses ECO as the source of the HFCLK 3-7: Reserved. Do not use

8.3.2 LFCLK Input Selection

The LFCLK in PRoC has two input options: ILO and WCO. THE LFCLK is selected using the WDT_CONFIG register's LFCLK_SEL bits, as shown in Table 8-5.

Table 8-5. LFCLK Input Selection Bits LFCLK_SEL

Name	Description
LFCLK_SEL[1:0]	LFCLK input clock selection 0: ILO. Uses the internal local oscillator as the source of the LFCLK 1: WCO. Uses the Watch Crystal Oscillator as the source of the LFCLK 2-3: Reserved. Do not use

8.3.3 ECO Divider Configuration

The ECO divider allows the device to divide the ECO clock before use as HFCLK. The divider is capable of dividing the ECO clock by powers of two between 0 and 3. The divider value is set using the BLE_BLERD_XTAL_CLK_DIV_CONFIG register's SYSCLK_DIV bits, as described in Table 8-6. By default, the ECO divider is set to 0 (HFCLK = XTAL_CLK).

Table 8-6. ECO Divider Selection Bits SYSCLK_DIV

Name	Description
SYSCLK_DIV[1:0]	ECO Clock divider. The 24-MHz crystal clock is divided to generate the HFCLK. 0: NO_DIV: HFCLK = XTAL_CLK/1 1: DIV_BY_2: HFCLK = XTAL_CLK/2 2: DIV_BY_4: HFCLK = XTAL_CLK/4 3: DIV_BY_8: HFCLK = XTAL_CLK/8

8.3.4 SYSCLK Prescaler Configuration

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

8.3.5 Peripheral Clock Divider Configuration

PRoC has 12 clock dividers, which include ten 16-bit clock dividers and two 16.5-bit fractional clock dividers. Fractional clock dividers allows the clock divisor to include a fraction of 0..31/32. For example, a 16.5-divider with an integer divide value of 3, produces signals to generate a 16-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 4, produces signals to generate a 12-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 3 and a fractional divider of 16 produces signals to generate a 48 / (3 + 16/32) = 48 / 3.5 = 13.7 MHz clock from a 48-MHz HFCLK. Not all 13.7-MHz clock periods are equal in size; half of them will be 3 HFCLK cycles and half of them will be 2 HFCLK cycles.

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

The ten non-fractional clock dividers are configured with the PERI_DIV_16_CTLx registers and the two fractional clock dividers are configured with the PERI_DIV_16_5_CTLx registers. Table 8-7 and Table 8-8 describe the configurations for these

registers.

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 [2, 65,536].

Table 8-8. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_5_CTLx

Bits	Name	Description
0	ENABLE_x	Divider enabled. HW sets this field to '1' as a result of an ENABLE command. HW sets this field to '0' as a result on a DISABLE command.
7:3	FRAC5_DIV_x	Fractional division by (FRAC5_DIV/32). Allows for fractional divisions in the range [0, 31/32]. Note that fractional division results in clock jitter as some clock periods may be 1 "clk_hf" cycle longer than other clock periods.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65,536].

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

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

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

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

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

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

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

shows the mapping of the peripheral output to the corresponding peripheral blocks (shown in Figure 8-1). Any of the 12 digital peripheral clocks can be mapped to a specific digital peripheral by using their respective PERI_PCLK_CTLx register, as described in Table 8-10.

Table 8-9. Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
0	IMO (Spread Spectrum)
1	SCB0
2	SCB1
3	CLOCK_PUMP
4	CSD (1)
5	CSD (2)
6	SAR
7	TCPWM0

Table 8-9. Peripheral Clock Multiplexer Output Mapping

PERI#_CLK	Peripheral
8	TCPWM1
9	TCPWM2
10	TCPWM3
15	LCD

Table 8-10. Programmable Clock Control Register - PERI_PCLK_CTLx

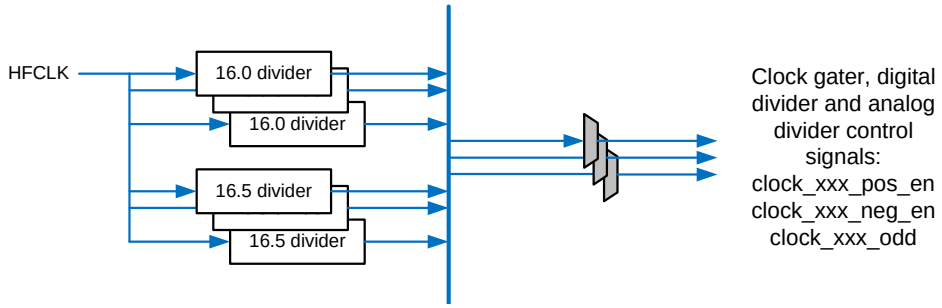
Bits	Name	Description
5:0	SEL_DIV	Specifies one of the dividers of the divider type specified by SEL_TYPE. If SEL_DIV is "4" and SEL_TYPE is "1", then the fifth (zero being first) 16-bit clock divider will be routed to the mux output for peripheral clock_x. Similarly, if SEL_DIV is "0" and SEL_TYPE is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

8.3.6 Peripheral Clock Configuration

Peripheral clock dividers along with the clock gater, digital clock divider, and analog clock divider generate clocks that are used by the end peripherals.

Peripheral clock divider generates three control signals: clock_xxx_pos_en, clock_xxx_neg_en and clock_xxx_odd. These signals along with HFCLK are used by the clock gater, digital clock divider, and analog clock divider to generate the desired clock signal. Figure 8-2 gives an overview of the peripheral clock divider.

Figure 8-2. Peripheral Clock Divider



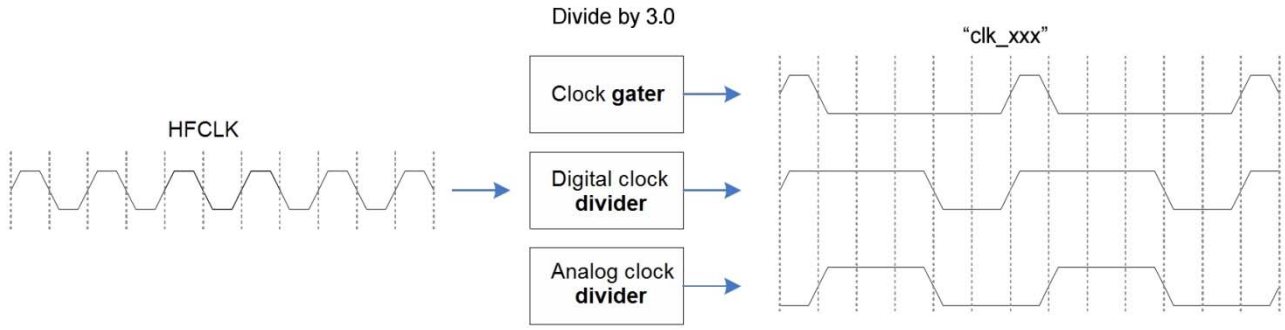
The peripheral clock divider consists of two layers:

- Clock dividers produce clock signals based on an integer divider and an optionally fractional clock divider value.
- Clock muxes select the signals from a specific clock divider and provide the three clock signals to a specific end peripheral.

8.3.6.1 Clock Generation

Figure 8-3 gives an overview of the clock generation via the clock gater, digital clock divider, and analog clock divider.

Figure 8-3. Clock Generation Using Clock Gater, Digital Clock Divider, and Analog Clock Divider



- Clock gater: This component gates the input clock source. The "Clock_xxx_pos_en" signal is used to enable clock gating. This component generates a single high pulse that lasts for as long as the HFCLK high pulse. This logic is typically used for digital logic that has positive edge flip flops only.
- Digital clock divider: This component uses "clock_xxx_pos_en" and "clock_xxx_neg_en" to generate the clock source. For an even integer divide, the generated clock has 50 percent duty cycle. For an odd integer divide, the generated clock does not have a 50 percent duty cycle; instead, the low time is one cycle longer than high time. This component is used for digital logic that has both positive and negative edge flip flops, such as an UDB array.
- Analog clock divider: This component uses "clock_xxx_pos_en", "clock_xxx_neg_en", and "clock_xxx_odd" signals to generate the clock source. For an integer divider (both even and odd dividers), the generated clock has 50 percent duty cycle.

8.4 Low-Power Mode Operation

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

8.5 Register List

Table 8-11. 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.
PWR_BG_TRIM4	Bandgap Trim Registers - These registers control the trim of the bandgap reference, allowing manipulation of the voltage references in the device.
PWR_BG_TRIM5	
CLK_IMO_SPREAD	IMO Spread Spectrum Control Register - This register controls the IMO spread spectrum functionality.
BLE_BLERD_DBUS	ECO Enable. This register contains the bit to enable or disable the ECO.
BLE_BLERD_BB_XO_CAP TRIM	X1 and X2 Capacitor Trim Register - This register controls the capacitor values at X1 and X2 node of the ECO.
CLK_ILO_CONFIG	ILO Configuration Register - This register controls the ILO configuration.
CLK_SELECT	Clock Select - This register controls clock tree configuration, selecting different sources for the system clocks.
WDT_CONFIG	This register selects the source for LFCLK.
WCO_CONFIG	WCO Enable. This register enables or disables the external watch crystal oscillator.
BLE_BLERD_XTAL_CLK_D IV_CONFIG	ECO Divider. This register selects the divider value for the ECO clock.
PERI_DIV_16_CTLx	Peripheral Clock Divider Control Registers - These registers configure the peripheral clock dividers, setting integer divide value, and enabling or disabling the divider.
PERI_DIV_16_5_CTLx	Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting fractional divide value, and enabling or disabling the divider.
PERI_PCLK_CTLx	Programmable Clock Control Registers - These registers are used to select the input clocks to peripherals.

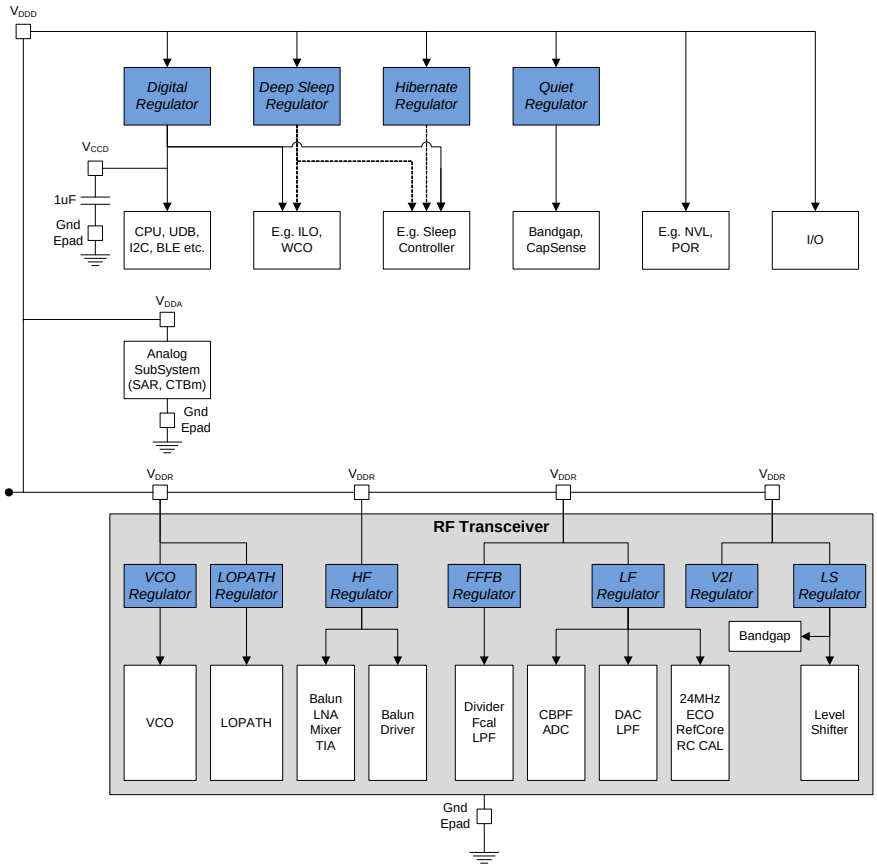
9. Power Supply and Monitoring

PRoC can be operated by directly connecting externally supplied voltage of 1.9 V to 5.5 V to the digital supply (VDDD), analog supply (VDDA), and radio supply (VDDR) pins.

PRoC devices have different internal regulators to support various power modes and BLE states. These include Active digital regulator, Quiet regulator, Deep-Sleep regulator, and Hibernate regulator to support core functionality. Moreover, the VCO regulator, LOPATH regulator, HF regulator, FFFB regulator, LF regulator, V2I regulator, and LS regulator support different BLE power states.

9.1 Block Diagram

Figure 9-1. Power System Block Diagram



The power system has separate digital, analog, and radio supply pins labeled V_{DDD}, V_{DDA}, and V_{DDR} as shown in Figure 9-1. All the grounds are connected to the E-PAD of the chip. Digital ground and CapSense ground are also brought out to different pins labeled V_{SS} and V_{SSA}. The digital and analog supply pins share ESD resources. To avoid turning on the ESD devices, the digital supply cannot exceed analog supply by more than 300 mV, not even on a transient basis. Thus, the V_{DDA} supply must ramp up before or concurrently with the V_{DDD} and V_{DDR} supply.

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

9.2 How It Works

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

9.2.1 Regulator Summary

9.2.1.1 Core Regulators

The Active digital regulator and Quiet regulator are enabled during the Active or Sleep power modes. They are turned off in the Deep-Sleep and Hibernate power modes (see Table 9-1 and Figure 9-1). The Deep-Sleep and Hibernate regulators are designed to fulfill power requirements in the low-power modes of the device.

Table 9-1. Core Regulator Status in Different Power Modes

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

Active Digital Regulator

The Active digital regulator provides the main digital logic in Active and Sleep modes. It regulates the external VDDD supply ranging from 1.9 V to 5.5 V to the nominal 1.8 V required for the digital core. This regulator has its output connected to a pin (VCCD) and requires an external decoupling capacitor (1 µF X5R).

The Active digital regulator is available only in Active and Sleep power modes.

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.

Deep-Sleep Regulator

This regulator supplies the circuits that remain powered in Deep-Sleep mode, such as the ILO, WCO, and SCB. The Deep-Sleep regulator is available in all power modes except the Hibernate mode. In Active and Sleep power modes, the main output of this regulator is connected to the output of the Digital regulator (VCCD). This regulator also has a separate replica output that provides a stable voltage for the ILO and WCO oscillators. This output is not connected to VCCD in Active and Sleep modes.

Hibernate Regulator

This regulator supplies the circuits that remain powered in Hibernate mode, such as the Sleep controller, low-power comparator, and SRAM. The Hibernate regulator is available in all power modes. In Active and Sleep modes, the output of this regulator is connected to the output of the Digital regulator. In Deep-Sleep mode, the output of this regulator is connected to the output of the Deep-Sleep regulator.

9.2.1.2 RF Transceiver Regulators

RF Transceiver regulators provide power to the different blocks inside the RF transceiver, as shown in Figure 9-1, VDDR provides the input to different regulators with a voltage ranging from 1.9 V to 5.5 V. RF Transceiver regulators generate the regulated voltage of 1.8 V from the VDDR.

Three out of the seven RF Transceiver regulators are available in all BLE modes except BLE_DeepSleep mode. The remaining regulators are available only in BLE_RX and BLE_TX modes (see Table 9-2). For more details on BLE modes, see the Bluetooth Low Energy Subsystem (BLESS) chapter on page 177.

Table 9-2. RF Transceiver Regulator Status in Different BLE Modes

Mode	LDO_LS	LDO_HF	LDO_VCO	LDO_LOPATH	LDO_FFFB	LDO_V2I	LDO_LF
BLE_DeepSleep	Off	Off	Off	Off	Off	Off	Off
BLE_Sleep	On	Off	Off	Off	Off	On	On
BLE_IDLE	On	Off	Off	Off	Off	On	On
BLE_TX	On	On	On	On	On	On	On
BLE_RX	On	On	On	On	On	On	On

Level Shifter Regulator (LDO_LS)

This regulator supplies power to the level shifter and the bandgap generator inside the RF transceiver. The Level Shifter regulator is one of the three regulators that are available in all modes (BLE_RX, BLE_TX, BLE_Idle, and BLE_Sleep) except the BLE_DeepSleep mode.

High Frequency Regulator (LDO_HF)

This regulator supplies power to the high frequency radio circuits such as low noise amplifier (LNA), mixer and balun circuit. This regulator is available only in the BLE_RX and BLE_TX modes.

LDO_FFFB

LDO_FFFB supplies power to the circuits in the synthesizer block such as dividers and VCO-calibration block. Similar to LDO_HF, this regulator is available only in the BLE_RX and BLE_TX modes.

VCO Regulator (LDO_VCO)

The LDO_VCO regulator supplies power to the VCO section of the RF transceiver. This regulator is available only in the BLE_RX and BLE_TX modes.

High-Speed Latches Regulator (LDO_LOPATH)

This regulator is used to supply power to the high-speed latches inside the RF transceiver. LDO_LOPATH is available only in the BLE_RX and BLE_TX modes.

V-to-I Regulator (LDO_V2I)

LDO_V2I is used to power the voltage current converter circuit in the RF transceiver. This regulator is available in all modes except the BLE_DeepSleep mode.

Low-Frequency Regulator (LDO_LS)

LDO_LS is used to power the low-frequency circuits in the RF transceiver. The low-frequency circuit consists of ADC, complementary band pass filter (CBPF), DAC, ECO, and so on. This regulator is available in all modes except the BLE_DeepSleep mode.

9.3 Voltage Monitoring

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

9.3.1 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.

9.3.1.1 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 device datasheet for details). The system will not come out of RESET until the supply is detected to be valid again.

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

9.3.1.2 Low-Voltage-Detect (LVD)

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

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

using the LVD_SEL field in the PWR_VMON_CONFIG register.

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

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

9.4 Register List

Table 9-3. 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.
PWR_INTR	Power System Interrupt Register – This register indicates the power system interrupt status.
PWR_INTR_MASK	Power System Interrupt Mask Register – This register controls which interrupts are propagated to the interrupt controller of the CPU.
PWR_VMON_CONFIG	Power System Voltage Monitoring Trim and Configuration – This register contains trim and configuration bits for the voltage monitoring system.

10. Chip Operational Modes

PRoC 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.

PRoC's 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 the power-up phase, hardware trim settings are loaded from nonvolatile (NV) latches 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 268. Exit from this mode returns the device to user mode.

10.4 Debug

Debug mode is an operational mode that allows observation of the PRoC 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 261.

For more details on protection modes, see the Device Security chapter on page 103.

11. Power Modes

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

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

Active, Sleep, and Deep-Sleep are standard ARM-defined power modes, supported by the ARM CPUs and instruction set architecture (ISA). Hibernate and Stop modes are additional low-power modes supported by PRoC. These modes are entered from firmware similar to Deep-Sleep, but on wakeup, the CPU and all peripherals go through a reset.

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

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

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

Figure 11-1. Power Mode Transitions State Diagram

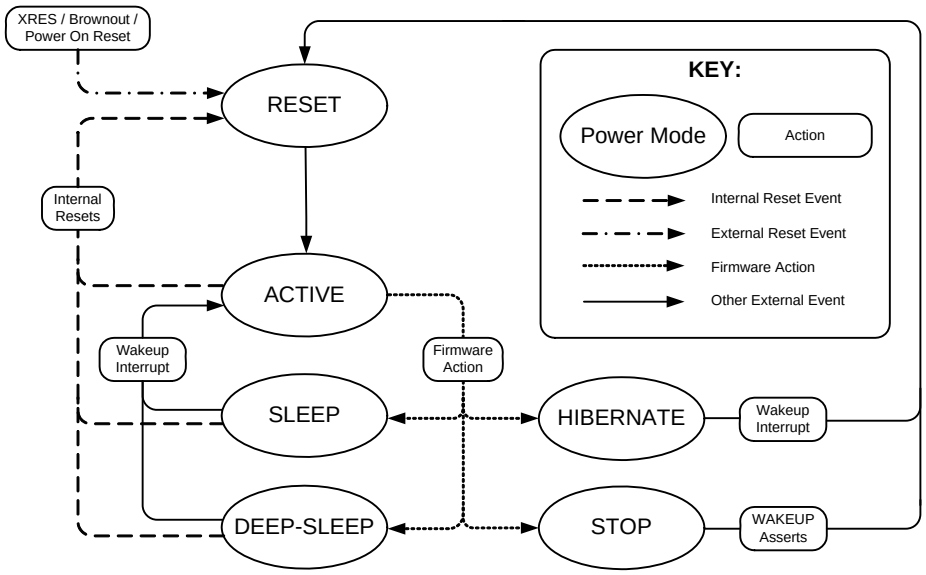


Table 11-1 illustrates the power modes offered by PRoC

Table 11-1. PRoC 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	Any (programmable)	Interrupt	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	Any (programmable)	Interrupt	All regulators are available. The active digital regulator can be disabled if external regulation is used.
Deep-Sleep	All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency (32 kHz) clock is available. Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup.	Manual register write	GPIO interrupt, low-power comparator, SCB, watchdog timer	ILO (32 kHz), WCO (32 kHz)	Interrupt	Deep-Sleep regulator and Hibernate regulator
Hibernate	Only SRAM and UDBs are retained; all internal supplies, except the hibernate supply are off. Wakeup is possible from a pin interrupt or a low-power comparator.	Manual register write	GPIO interrupt, low-power comparator	None	Reset (with interrupt state retention)	Hibernate regulator
Stop	All internal supplies are off. Only GPIO states are retained. Wakeup is possible from XRES or WAKEUP pins only.	Manual register write	WAKEUP pin	None	Reset	None

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. For details on BLESS state power modes, see the Bluetooth Low Energy Subsystem (BLESS) chapter on page 177.

11.1 Active Mode

Active mode is the primary power mode of the PRoC 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 PRoC 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, UDB, and high-speed logic are in retention. The MHz range clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (32 kHz) oscillator and watch crystal oscillator (WCO) remain 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, I2C slave) continue to operate. Interrupts from low-speed, asynchronous or low-power analog peripherals can cause a wakeup from Deep-Sleep mode. CTBm can also operate in this mode with reduced power and bandwidth. For details on power consumption and CTBm bandwidth, refer to the device datasheet.

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

11.4 Hibernate Mode

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

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

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

Wakeup from Hibernate mode is possible from a pin interrupt or a low-power comparator only. Wakeup from hibernate incurs a reset rather than a wakeup from interrupt. When waking up from hibernate, the CPU and most peripherals are in their reset state and firmware will start at the reset vector. I/O pins will be tri-stated after reset, unless they are explicitly frozen by firmware before entry into Hibernate mode. To know the cause of interrupt, use the TOKEN bits in PWR_STOP register, as described in Low-Power Mode Entry and Exit on page 92.

External reset (XRES) triggers a full system restart. In this case, the cause is not readable after the device restarts, and I/O pins will not retain their “frozen” state.

11.5 Stop Mode

In the Stop mode, the CPU, all internal regulators, and all peripherals are switched off. Wakeup from Stop mode is a system reset and it is possible from XRES or WAKEUP pins only. I/O pins will be tri-stated after reset, unless they are explicitly frozen by firmware before entry into Stop mode. To know the cause of interrupt, use the TOKEN bits in PWR_STOP register, as described in Low-Power Mode Entry and Exit on page 92.

External reset (XRES) triggers a full system restart. In this case, the cause is not readable after the device restarts, and I/O pins will not retain their "frozen" state.

11.6 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	Hibernate	Stop
CPU	On	Retention[a]	Retention	Off	Off
SRAM	On	Retention	Retention	Retention	Off
High-speed peripherals (peripherals that operate from HFCLK)	On	On	Retention	Off	Off
CapSense	On	On	Retention	Off	Off
Universal digital block (UDB)	On	On	Retention	Off	Off
Low-speed peripherals (peripherals that operate from ILO)	On	On	On (optional)	Off	Off
Internal main oscillator (IMO)	On	On	Off	Off	Off
Internal low-speed oscillator (ILO/WCO 32 kHz)	On	On	On (optional)	Off	Off
Asynchronous peripherals	On	On	On	Off	Off
Power-on-reset, Brownout detection	On	On	On	Off	Off
Regular analog peripherals except CTBm	On	On	Off	Off	Off
Hibernate analog peripheral (LP comparator)	On	On	On	On	Off
CTBm	On	On	On	Off	Off
GPIO output state	On	On	On/Frozen	Frozen[b]	Frozen

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

Table 11-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any interrupt source	Interrupt
	Any reset source	Reset
Deep-Sleep	GPIO interrupt	Interrupt
	Low-power comparator	Interrupt
	I2C address match	Interrupt
	Watchdog timer	Interrupt / Reset
	XRES (external reset pin)[a], Brownout	Reset
	CTBm	Interrupt
	BLESS	Interrupt
Hibernate	GPIO Interrupt	Reset
	Low-power comparator	Reset
	XRES (external reset pin)[a], Brownout	Reset
Stop	WAKEUP pin	Reset
	XRES (external reset pin)[a], Brownout	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.

11.7 Low-Power Mode Entry and Exit

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

The transition to Sleep, Deep-Sleep, and Hibernate modes are controlled by the flags SLEEPDEEP in the CM0 System Control Register (CM0_SCR) and HIBERNATE in the System Resources Power subsystem (PWR_CONTROL)

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

The LPM READY bit in the PWR_CONTROL register shows the status of Deep-Sleep and Hibernate regulators. If the firmware tries to enter Deep-Sleep or Hibernate mode before the regulators are ready, then PRoC goes to Sleep mode first, and when the regulators are ready, the device enters Deep-Sleep or Hibernate 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.

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

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

The fields in PWR_STOP register are:

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

The recommended procedure to enter Stop mode is:

1. Write TOKEN = <any application-specified value>
2. Write UNLOCK = 0x3A
3. Write POLARITY = <application-specified polarity>
4. Write FREEZE = 1
5. Write STOP = 1

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

The recommended firmware procedure on wakeup from Stop or Hibernate mode is as follows:

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

11.8 Register List

Table 11-4. Power Mode Register List

Register Name	Description
CM0_SCR	System Control Register - Sets or returns system control data.
PWR_CONTROL	Power Mode Control - Controls the device power mode options and allows observation of current state.
PWR_STOP	This register controls entry/exit from the Stop power mode.

12. Watchdog Timer

The watchdog timer (WDT) is used to automatically reset the device in the event of an unexpected firmware execution path. The WDT runs from the LFCLK, generated by the ILO or WCO. The timer, if enabled, 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
- Supports two 16-bit and one 32-bit independent counters, which can be cascaded to increase the interval

12.2 Block Diagram

Figure 12-1. Watchdog Timer Block Diagram

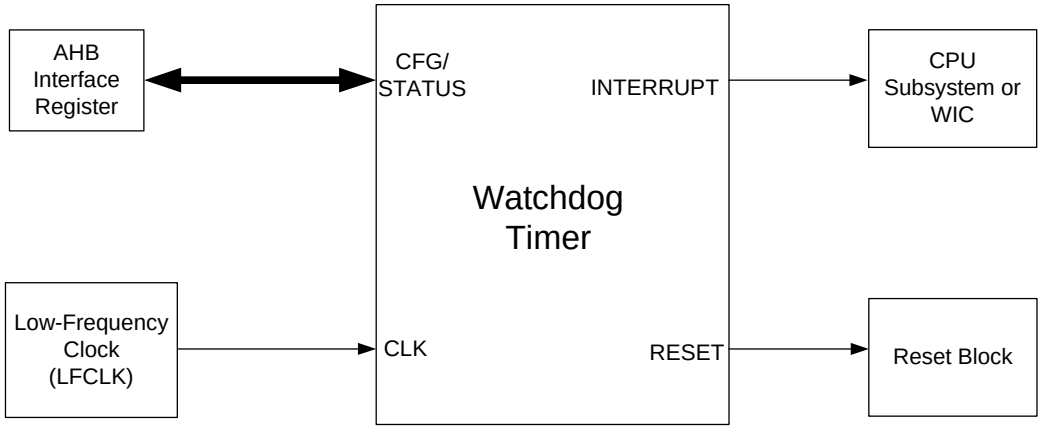
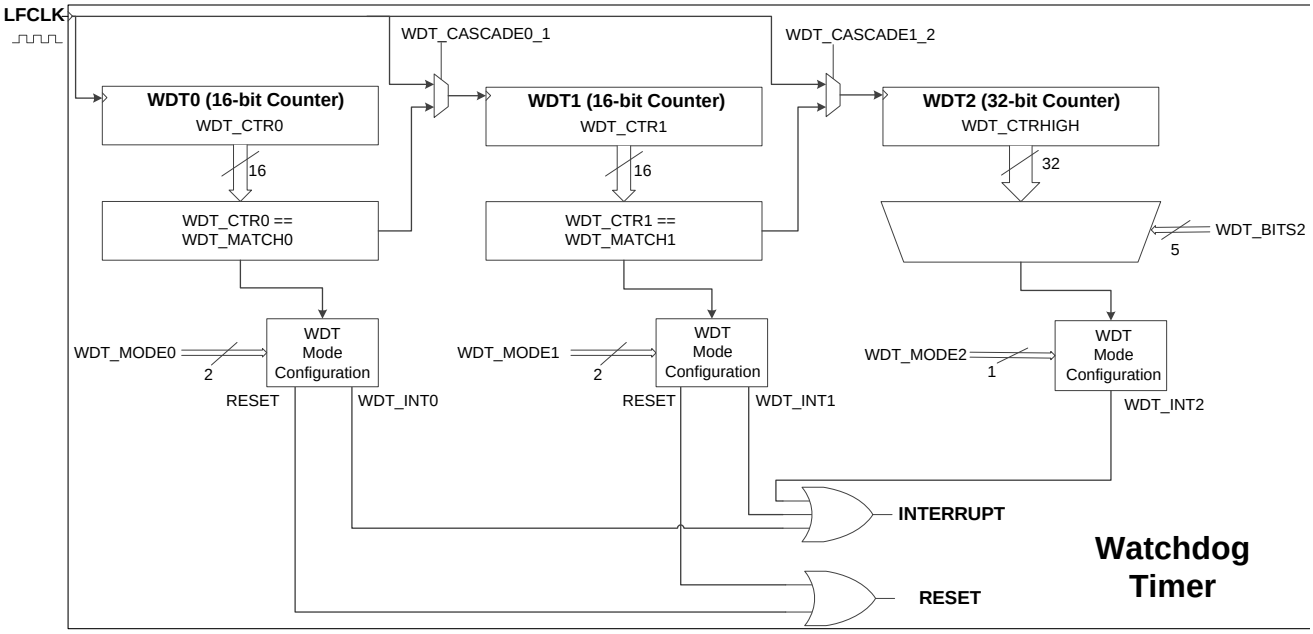


Figure 12-2. Watchdog Timer Internal Block Diagram



12.3 How It Works

The WDT asserts an interrupt or a hardware reset to the device after a programmable interval, unless it is periodically serviced in firmware. The WDT has two 16-bit counters (WDT0 and WDT1) and one 32-bit counter (WDT2). These counters can be configured to work independently or in cascade.

WDT0 and WDT1 can be configured to generate an interrupt on a match event, that is, when the counter value equals the match value. The WDT0 and WDT1 counters can also be configured to generate a reset on a match event or after three successive match events that are not handled (match event interrupt not cleared).

WDT2 can be configured to generate an interrupt based on the value stored in the WDT_BITS2[4:0] bits in the WDT_CONFIG register. WDT2 cannot generate a system reset or an interrupt with any match value similar to WDT1 or WDT0. WDT2 can generate an interrupt only on a rising edge on one of the 32 bits present in the counter. The WDT_BITS2[4:0] bits in the WDT_CONFIG register control the bit that generates the interrupt. See the WDT_CONFIG register in the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM) for details.

WDT_MODEx bits are used to configure the watchdog counters as described above.

The cascade configuration shown in Figure 12-2 provides an option to increase the reset or interrupt interval. Note that cascading two 16-bit counters will not provide a 32-bit counter; instead, you will obtain a 16-bit period counter with a 16-bit prescaler. For example, when cascading WDT0 and WDT1, WDT0 acts as a prescaler for WDT1 and the prescaler value will be defined by the WDT_MATCH0[15:0] bits in the WDT_MATCH register. The WDT1 will have a period defined by WDT_MATCH1[31:16] bits in the WDT_MATCH register. The same logic applies to WDT1 and WDT2 cascading.

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. If configured to generate a reset on a match event, reset the watchdog by clearing the WDTx counter. The WDTx counter can be cleared by setting the WDT_RESETx bit in the

WDT_CONFIG register. For details, refer to the WDT_CONFIG register in the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

- 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 the same watchdog counter for generating system reset and interrupt. For example, if WDT0 is used for generating system reset against crashes, then WDT1 or WDT2 should be used for periodic interrupt generations.

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

1. Set the WDT_CLEAR0 or WDT_CLEAR1 bit in the WDT_CONFIG register for WDT0 or WDT1 to reset the corresponding watchdog counter to '0' on a match event.
2. Write the desired match value to the WDT_MATCH register for WDT0/WDT1 and the WDT_BITS2 value to the WDT_CONFIG register for WDT2.
3. Clear the WDT_INTx bit in WDT_CONTROL to clear any pending interrupt.
4. Enable the WDT interrupt by configuring the WDT_MODEx bits in WDT_CONFIG. Configure the WDT_MODE0 or WDT_MODE1 bits in WDT_CONFIG for WDT0 or WDT1 to '1' (interrupt on match) or '3' (interrupt on match and reset on third unhandled match). For WDT2, set the WDT_MODE2 bit in the WDT_CONFIG register.
5. Enable global WDT interrupt in the CM0_ISER register (See the Interrupts chapter on page 53 for details).
6. In the ISR, clear the WDT interrupt.

Changing WDT_MATCH requires three LFCLK cycles to come into effect. After changing WDT_MATCH, do not enter the Deep-Sleep mode for one LFCLK cycle to ensure the WDT updates to the new setting.

12.3.1 Enabling and Disabling WDT

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

After WDT is enabled, it is not recommended to write to the WDT configuration (WDT_CONFIG) and control the (WDT_CONTROL) registers. Accidental corruption of WDT registers can be prevented by setting the WDT_LOCK[15:14] bits of the CLK_SELECT register. If the application requires updating the match value (WDT_MATCH) when the WDT is running, the WDT_LOCK bits must be cleared. The WDT_LOCK bits require two different writes to clear both the bits. Writing a '1' to the bits clears bit 0. Writing a '2' clears bit 1. Writing a '3' sets both the bits and writing '0' does not have any effect. For details, refer to the CLK_SELECT register in the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

12.3.2 WDT Operating Modes

The WDT0 and WDT1 can be used to generate a reset to stop the system from going into the unresponsive state or to generate an interrupt to wake up the system from Sleep or Deep-Sleep power modes. The bit field WDT_MODEx[1:0] in the WDT_CONFIG register can be configured to select the required action when the count value stored in the WDT_CTRx register bits equals the match values (WDT_MATCHx) stored in the WDT_MATCH register. See the WDT_CTRHIGH, WDT_CTRLOW, and WDT_MATCH registers in the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM)for details.

Table 12-1. WDT0 and WDT1 Modes

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

The WDT2 can be used to generate interrupts based on the status of the WDT_BITS2[4:0] register bits.

Table 12-2. WDT2 Mode

Bit-field Name	Description
WDT_MODE2	0: Free-running counter with no interrupt requests 1: Free-running counter with interrupt request on the rising edge of the bit specified by WDT_BITS2 bits in the WDT_CONFIG register

Note: When the watchdog counters are configured to generate an interrupt every LFCLK cycle, make sure you read the WDT_CONTROL register after clearing the watchdog interrupt (setting the WDT_INTx bit in the WDT_CONTROL register). Failure to do this may result in missing the next interrupt; it will also result in an interrupt cycle of LFCLK/2.

12.3.3 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, see the Power Modes chapter on page 89.

12.3.4 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 99.

12.4 Register List

Table 12-3. WDT Registers

Register Name	Description
WDT_CTRLOW	Watchdog counters 0 and 1
WDT_CTRHIGH	Watchdog counter 2
WDT_MATCH	Match value for watchdog counters 0 and 1
WDT_CONFIG	Contains WDT configuration bits
WDT_CONTROL	Controls the behavior of WDT counters

13. Reset System

PRoC 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. PRoC 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 electrical signal external to the PRoC
- Protection fault reset (PROT_FAULT) to reset the device if unauthorized operating conditions occur
- Hibernate wakeup reset to bring the device out of the Hibernate low-power mode
- Stop wakeup reset to bring the device out of the Stop low-power mode

13.1 Reset Sources

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

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

BOD events do not set a reset cause status bit, but in some cases they can be detected. In some BOD events, V_{CCD} will fall below the minimum logic operating voltage, but remain above the minimum logic retention voltage. Thus, some BOD events may be distinguished from POR events by checking for logic retention. This is explained further in Identifying Reset Sources on page 100.

13.1.3 Watchdog Reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. This feature is enabled by setting the WDT_ENABLEx bit in the WDT_CONTROL 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 or undetectable 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 95

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 or undetectable BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.1.5 External Reset

External reset (XRES) is a user-supplied reset that causes immediate system reset when asserted. The XRES_N pin is ***active low*** – a high voltage on the pin causes no behavior and a low voltage causes a reset. The pin is pulled high inside the device. XRES_N 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, undetectable BOD, or XRES.

13.1.6 Protection Fault Reset

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

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

13.1.7 Hibernate Wakeup Reset

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

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

For more details, see Hibernate Mode on page 91.

13.1.8 Stop Wakeup Reset

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

Some stop wakeup resets can be detected by examining the TOKEN bit-field (bits 0:7) in the PWR_STOP register. This bit-field will be filled with a key when Stop mode is entered. Its contents will be retained if the device is woken up using the WAKEUP pin. If the device is woken up with the XRES pin, the wakeup source cannot be detected. For more details, see Stop Mode on page 91.

13.2 Identifying Reset Sources

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

Hibernate wakeup resets can be detected by examining the comparator and pin interrupt registers that were configured to wake the device from Hibernate mode. Stop wakeup resets that occur as a result of a WAKEUP pin event can be detected by examining the PWR_STOP register, as described previously. Stop wakeup resets that occur as a result of an XRES cannot be detected. The other reset sources can be inferred to some extent by the status of RES_CAUSE shown in Table 13-1.

Table 13-1. Reset Cause Bits to Detect Reset Source

Bits	Name	Description
0	RESET_WDT	A WatchDog Timer reset has occurred since last power cycle.
3	RESET_PROT_FAULT	A protection violation occurred that requires a RESET.
4	RESET_SOFT	Cortex-M0 requested a system reset through it's SYSRESETREQ.

Brownout events can be subdivided into two categories: retention resets and non-retention resets. If V_{CCD} dips below the minimum logic operating voltage, but not below the minimum logic retention voltage, then a BOD reset occurs; but retention of registers is maintained. If V_{CCD} dips below both minimum operating and minimum retention voltage, then a BOD reset occurs

without retention of registers. This register retention can be detected using a special register, PWR_BOD_KEY. The PWR_BOD_KEY register only changes value when written by firmware or when a non-retention reset such as a non-retention BOD, XRES, or POR event. This register may be initialized by firmware, and then checked in subsequent executions of startup code to determine if a retention BOD occurred.

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

13.3 Register List

Table 13-2. Reset System Register List

Register Name	Description
WDT_CONTROL	Watchdog Timer Control Register - This register allows configuration of the device watchdog timer.
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.
PWR_STOP	This register controls entry/exit from the Stop power mode.

14. Device Security

PRoC offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and robust flash protection provide a high level of security. Additional security can be gained by implementing custom functionality in the universal digital blocks (UDBs) instead of in firmware. It is more difficult to reverse-engineer a hardware design implemented in the UDBs than it is to reverse-engineer object code.

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 of flash row and chip protection, see the PRoC BLE Programming Specifications.

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

14.1 Features

The PRoC 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, PRoC 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 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 removes all debug access to user code or memory, and the flash cannot be erased. Access to most registers is still available; debug access to registers to reprogram flash is

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

14.2.2 Flash Security

The PRoC 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.

Table 14-1. Flash Protection Levels

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

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

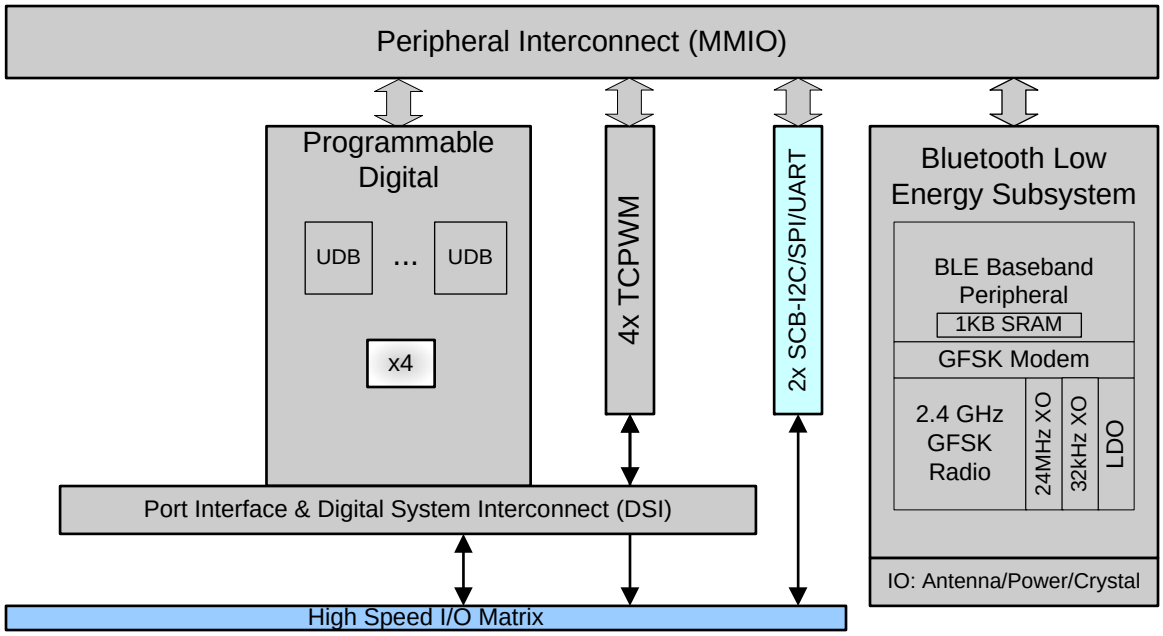
Section D: Digital System

This section encompasses the following chapters:

- Serial Communications (SCB) chapter on page 107
- Universal Digital Blocks (UDB) chapter on page 133
- Timer, Counter, and PWM chapter on page 147
- Bluetooth Low Energy Subsystem (BLESS) chapter on page 177

Top Level Architecture

Digital System Block Diagram



15. Serial Communications (SCB)

The Serial Communications Block (SCB) of PRoC supports three serial interface protocols: SPI, UART, and I2C. Only one of the protocols is supported by an SCB at any given time. PRoC devices have two SCBs. Additional instances of the serial peripheral interface (SPI) and UART protocols can be implemented using universal digital blocks (UDBs).

15.1 Features

This block supports the following features:

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

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

15.2 Serial Peripheral Interface (SPI)

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

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

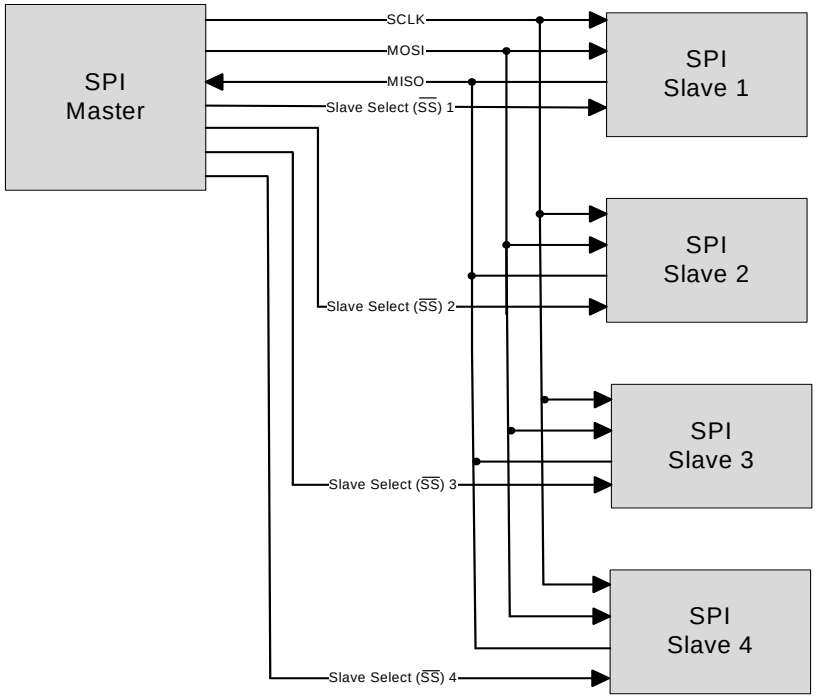
15.2.1 Features

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

15.2.2 General Description

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

Figure 15-1. SPI Example



A standard SPI interface consists of four signals as follows.

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

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

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

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

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

15.2.3 SPI Modes of Operation

15.2.3.1 Motorola SPI

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

Modes of Motorola SPI

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

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

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

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

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

Figure 15-2. SPI Motorola, 4 Modes

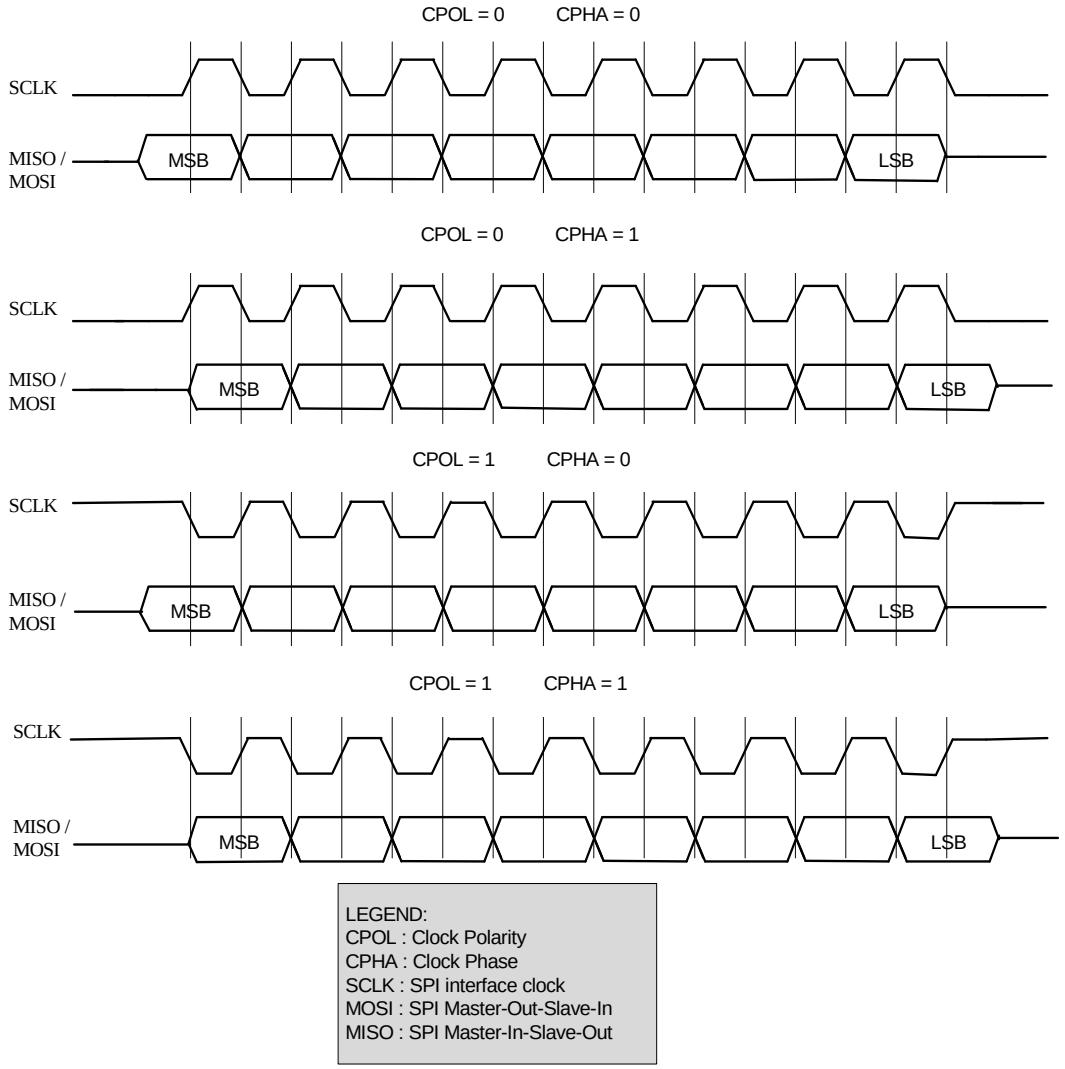
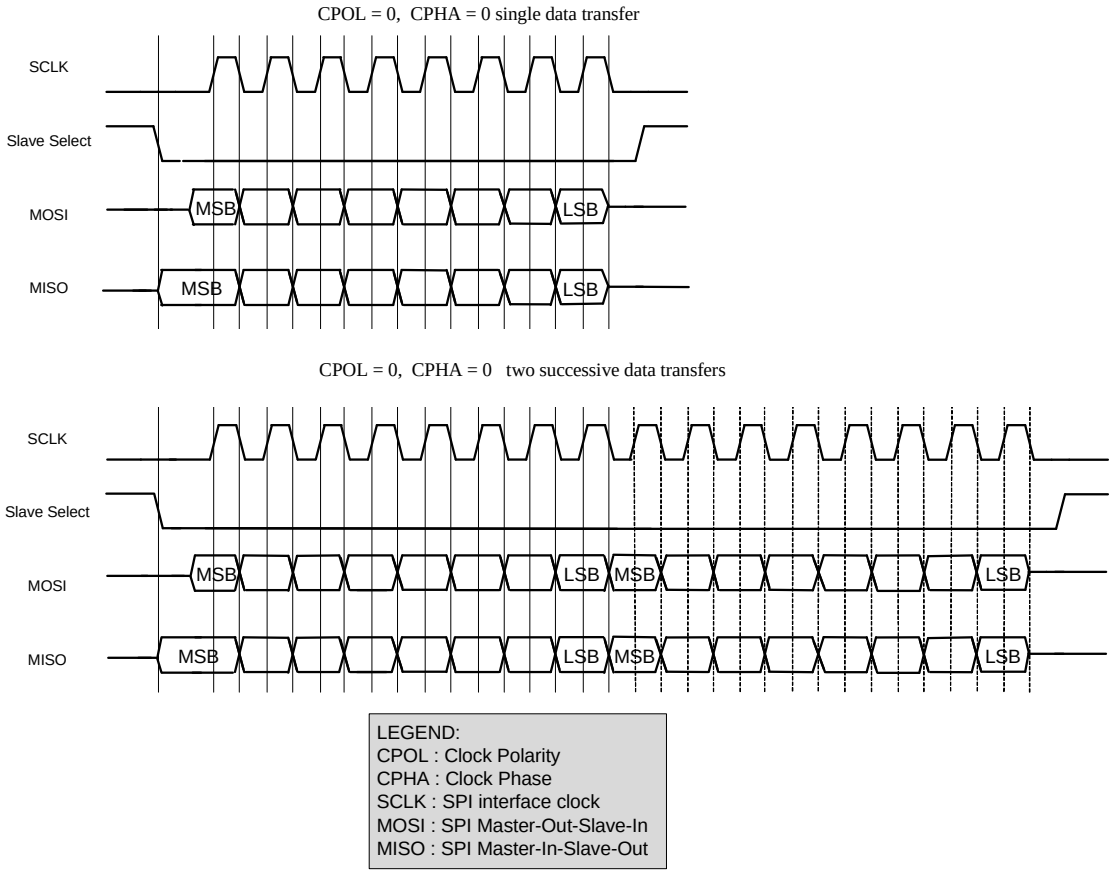


Figure 15-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 15-3. SPI Motorola Data Transfer Example



Configuring SCB for SPI Motorola Mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI Motorola mode by writing '00' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the CPHA and CPOL fields (bits 2 and 3 respectively) of the SCB_SPI_CTRL register.
4. Follow steps 2 to 4 mentioned in Enabling and Initializing SPI on page 117.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.2.3.2 Texas Instruments SPI

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

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

Figure 15-4. SPI TI Data Transfer Example

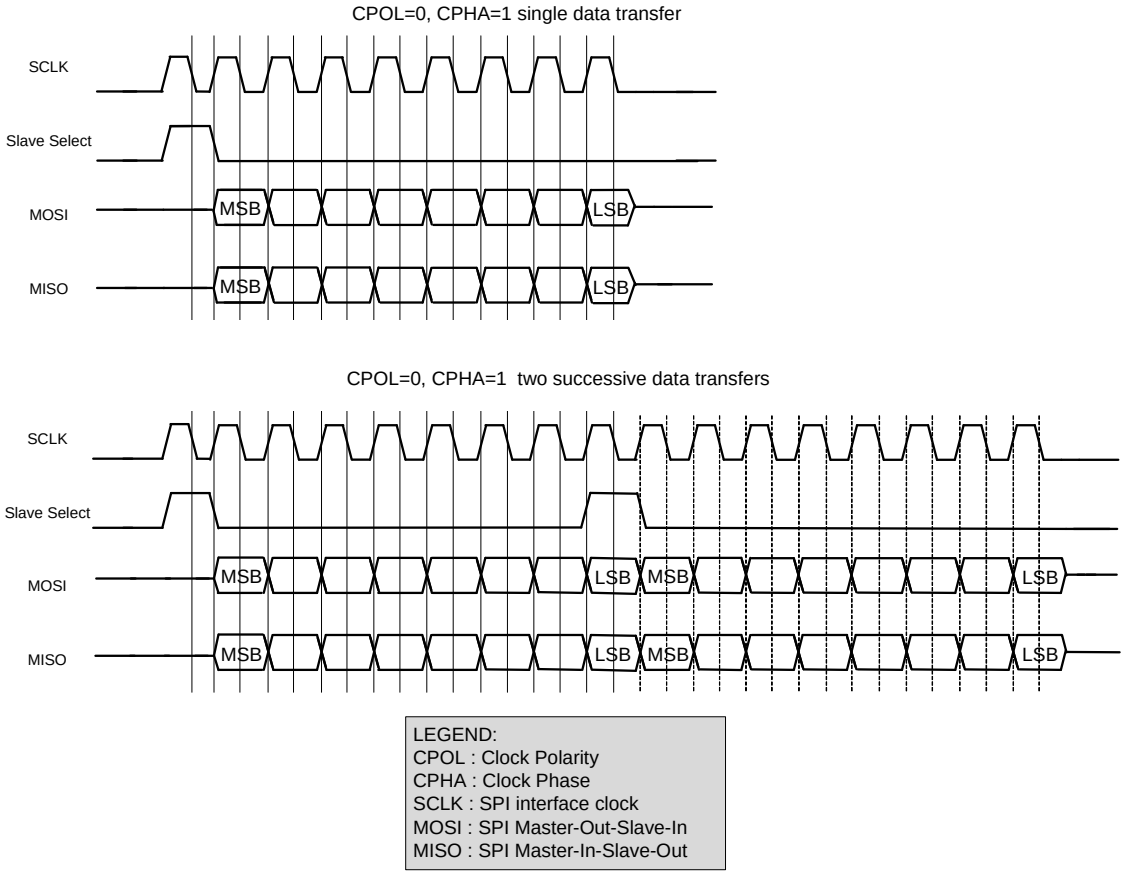
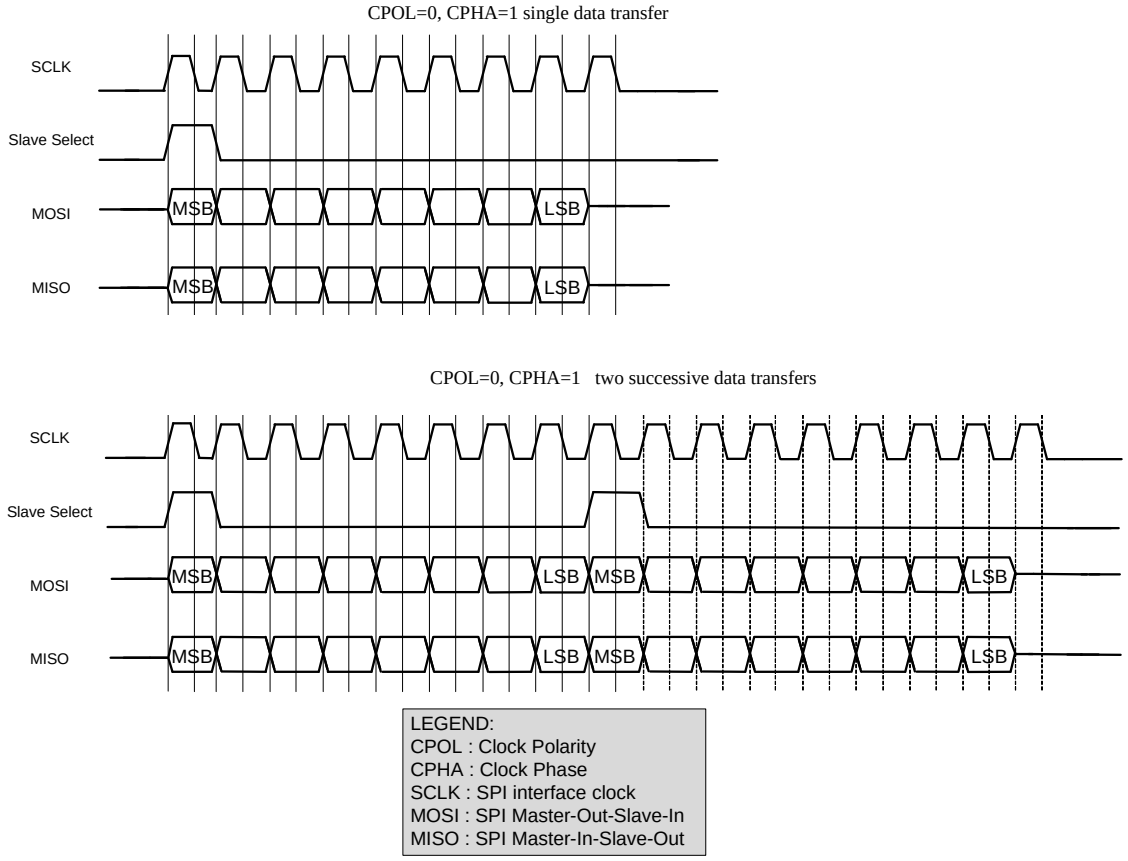


Figure 15-5 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

Figure 15-5. SPI TI Data Transfer Example



Configuring SCB for SPI TI Mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI TI mode by writing '01' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in TI by writing to the SELECT_PRECEDE field (bit 1) of the SCB_SPI_CTRL register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 5 mentioned in Enabling and Initializing SPI on page 117.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

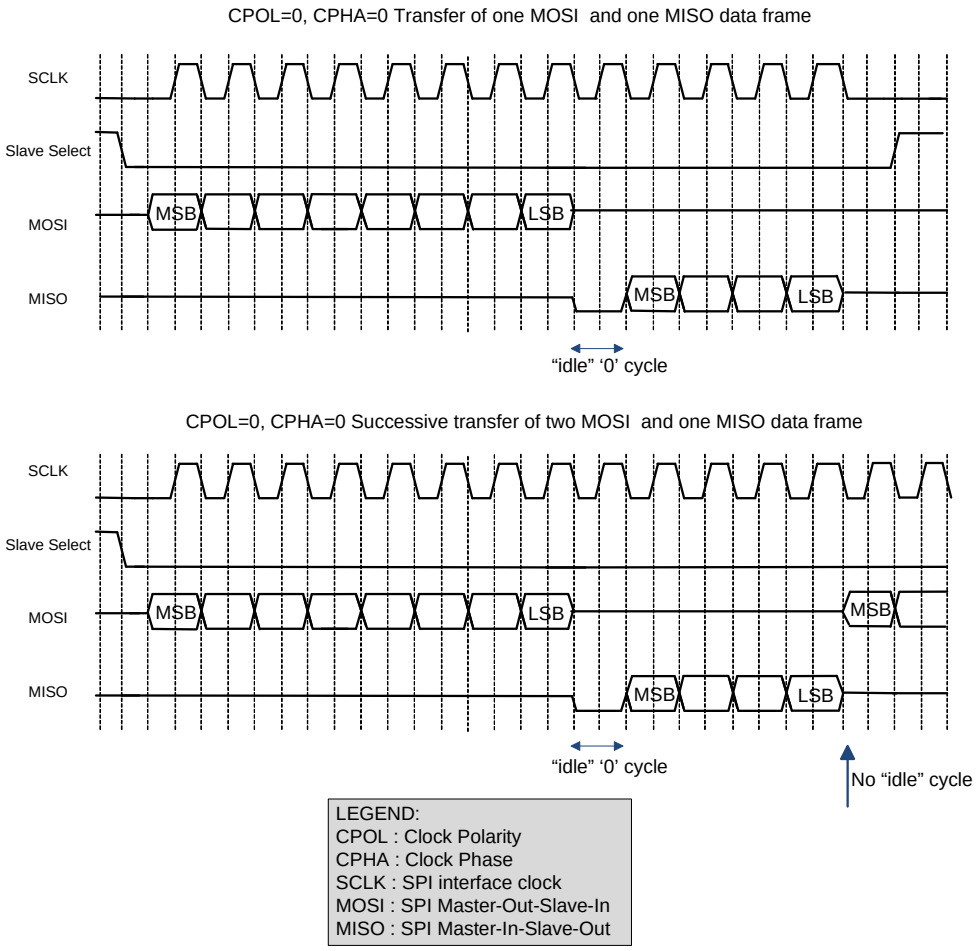
15.2.3.3 National Semiconductors SPI

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

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

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

Figure 15-6. SPI NS Data Transfer Example



Configuring SCB for SPI NS Mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in Enabling and Initializing SPI on page 117.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.2.4 Using SPI Master to Clock Slave

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

15.2.5 Easy SPI Protocol

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

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

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

used as a single 32x8 bit EZFIFO because only a fixed 8-bit width data is used in EZ mode.

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

15.2.5.1 EZ Address Write

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

15.2.5.2 Memory Array Write

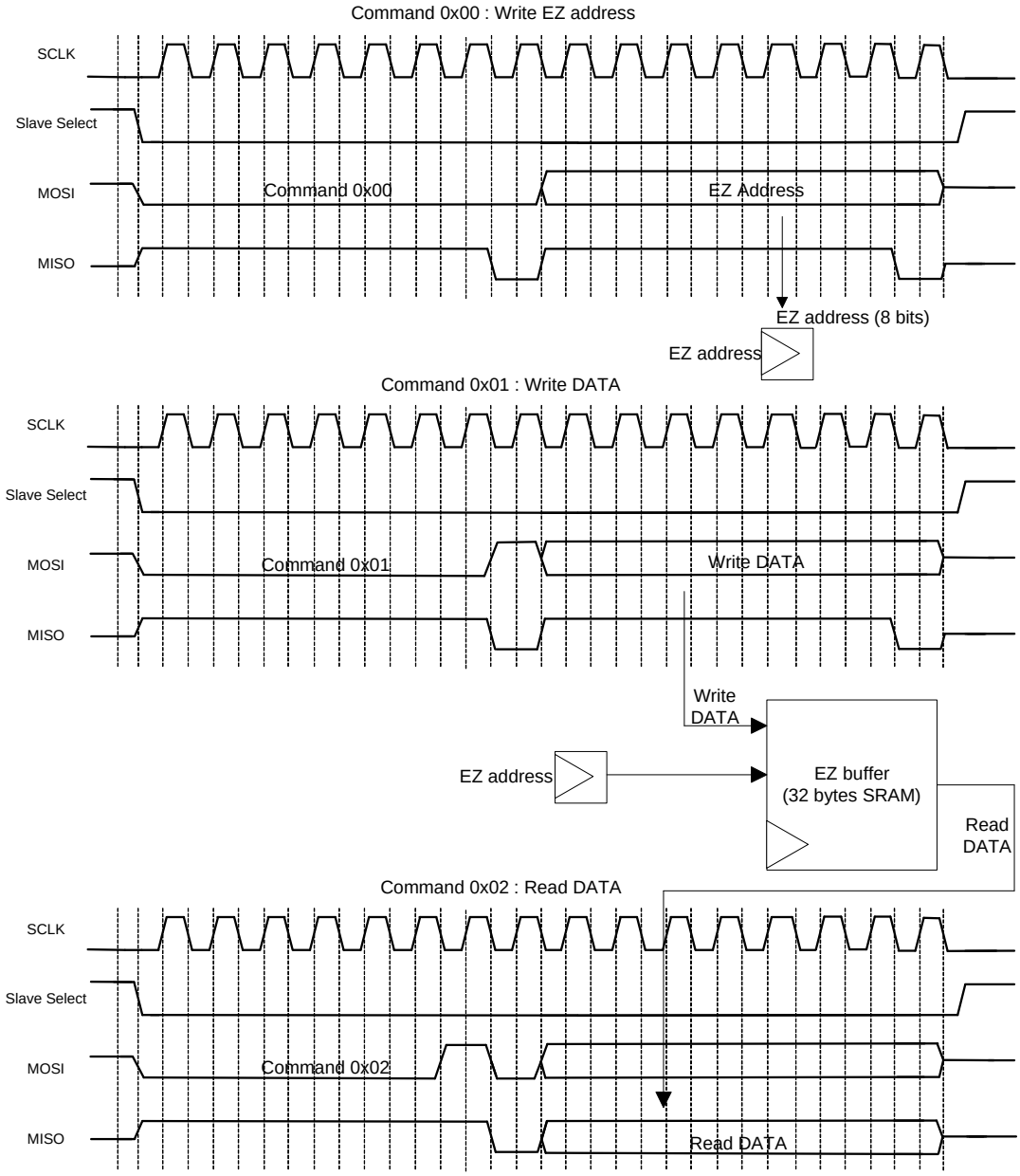
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was observed (0xFE) or not (0xFF). Any additional write data bytes on the MOSI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

15.2.5.3 Memory Array Read

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

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

Figure 15-7. EZSPI Example



15.2.5.4 Configuring SCB for EZSPI Mode

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

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Use continuous transmission mode for the transmitter by writing '1' to the CONTINUOUS bit of SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in Enabling and Initializing SPI on page 117.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.2.6 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 15-1. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

Table 15-1. SPI Registers

Register Name	Operation
SCB_CTRL	Enables the SCB, selects the type of serial interface (SPI, UART, I2C), and selects internally and externally clocked operation, EZ and non-EZ modes of operation.
SCB_STATUS	In EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory.
SCB_SPI_CTRL	Configures the SPI as either a master or a slave, selects SPI protocols (Motorola, TI, National) and clock-based submodes in Motorola SPI (modes 0,1,2,3), selects the type of SELECT signal in TI SPI.
SCB_SPI_STATUS	Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode.
SCB_TX_CTRL	Specifies the data frame width and specifies whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCB_RX_FIFO_RD	Holds the data frame read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO - behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data frame read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Holds the slave device address and mask values.
SCB_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_EZ_DATA	Holds the data in EZ memory location

15.2.7 SPI Interrupts

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

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

The SPI supports interrupts on the following events:

- SPI master transfer done
- SPI Bus Error - Slave deselected at an unexpected time in the SPI transfer
- SPI slave deselected after any EZSPI transfer occurred

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

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

15.2.8 Enabling and Initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCB_SPI_CTRL register, according to Table 15-2. This includes selecting the submodes of the protocol and selecting master-slave functionality. EZSPI can be used with slave mode only.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-3:
   a. Specify the data frame width. This should always be 8 for EZSPI.
   b. Specify whether MSB or LSB is the first bit to be transmitted/received. This should always be MSB first for EZSPI.
3. Program the transmitter and receiver FIFOs using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers respectively, as shown in Table 15-4:
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFO.
4. Program SCB_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in Table 15-5
5. Enable the block (write a '1' to the ENABLED bit of the SCB_CTRL register). After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally clocked to internally clocked operation. The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

Table 15-2. SCB_SPI_CTRL Register

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

Table 15-3. SCB_TX_CTRL/SCB_RX_CTRL Registers

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

Table 15-4. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

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

Table 15-5. SCB_CTRL Register

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

15.2.9 Internally and Externally Clocked SPI Operations

The SCB supports both internally and externally clocked operations for SPI and I2C functions. An internally clocked operation uses a clock provided by the chip. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation enables operation in the Deep-Sleep system power mode.

Internally clocked operation uses the high-frequency clock (HFCLK) of the system. For more information on system clocking, see the Clocking System chapter on page 73. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

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

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

Table 15-6. SPI Slave Maximum Data Rates

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

Externally clocked operation is limited to:

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

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

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

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

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

describe the settings for SPI (in non-EZ and EZ modes).

15.2.9.1 Non-EZ Mode of Operation

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

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

SPI (non-EZ) Mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
System Power Mode	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 0	EC_AM_MODE = 1
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). In Sleep mode, the MASK bit can be configured by the user.	Not supported	Not supported
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		
Hibernate	The SCB is not available in these modes (see the Power Modes chapter on page 89)			
Stop				

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

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

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

15.2.9.2 EZ Mode of Operation

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

Table 15-8. SPI Operation in EZ Mode

SPI, EZ Mode				
	EC_OP_MODE = 0		EC_OP_MODE = 1	
System Power Mode	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 0	EC_AM_MODE = 1
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock. Operation using internal clock. In Active mode, the Wakeup interrupt cause is disabled (MASK = 0). In Sleep mode, the MASK bit can be configured by the user.	Invalid	Selection using external clock. Operation using external clock.
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Send 0xFF.		Selection using external clock. Operation using external clock.
Hibernate	The SCB is not available in these modes (see the Power Modes chapter on page 89)			
Stop				

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

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

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

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

15.3 UART

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

- TX: Transmitter output
- RX: Receiver input

15.3.1 Features

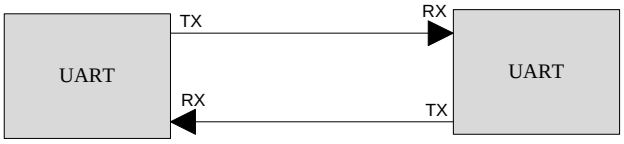
- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 1 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports Local Interconnect Network (LIN)
  - Break detection

  - Baud rate detection
  - Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus)
- Multi-processor mode
- Data frame size programmable from 4 to 9 bits
- Programmable number of STOP bits, which can be set in terms of half bit periods between 1 and 4
- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

15.3.2 General Description

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

Figure 15-8. UART Example



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

Three different serial interface protocols are supported:

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

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

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

15.3.3 UART Modes of Operation

15.3.3.1 Standard Protocol

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

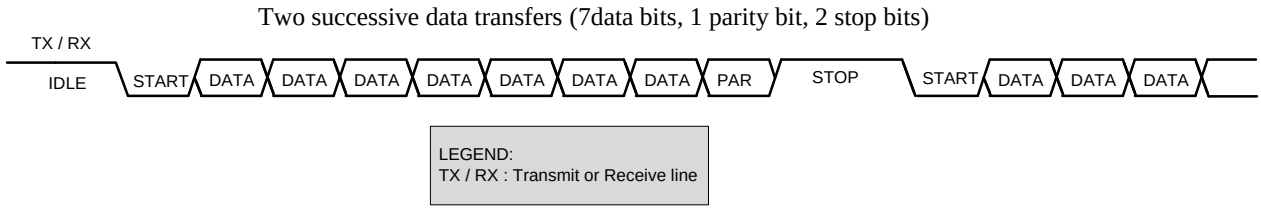
Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample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

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

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

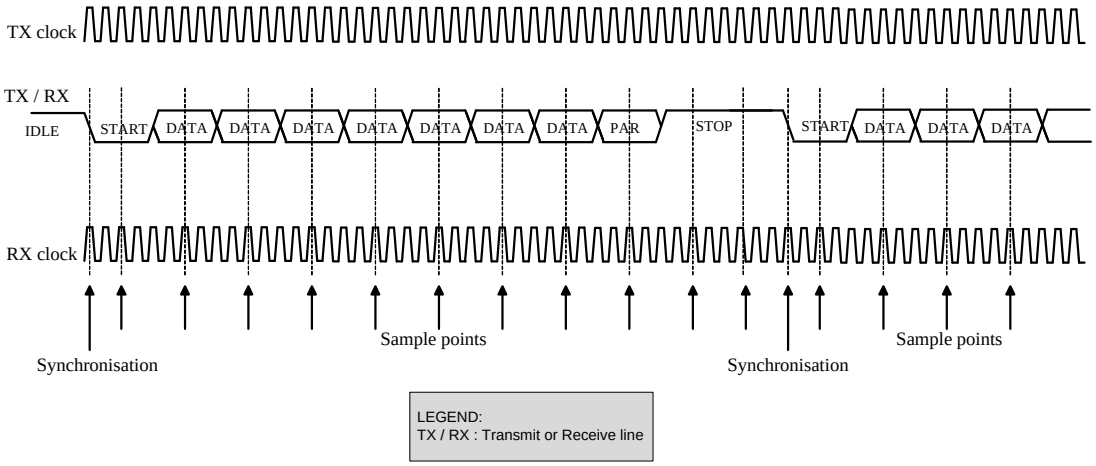
Figure 15-9 illustrates the UART protocol.

Figure 15-9. UART, Standard Protocol Example



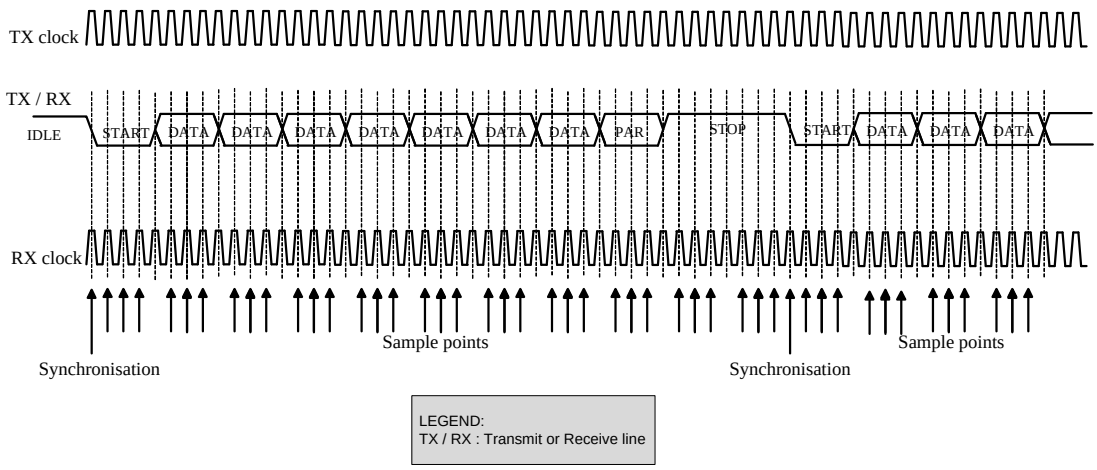
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. Figure 15-10 illustrates this.

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



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

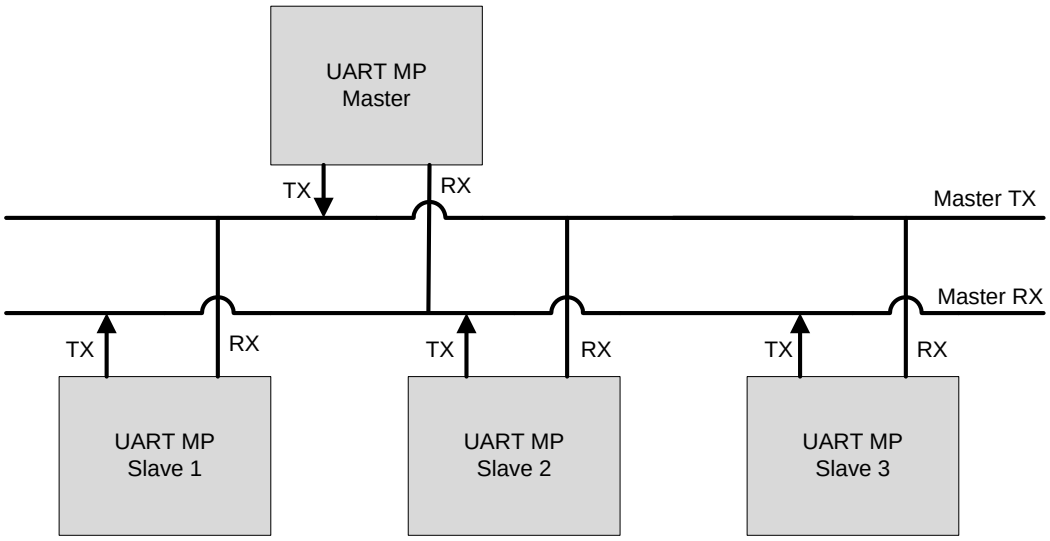
Figure 15-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor Mode

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

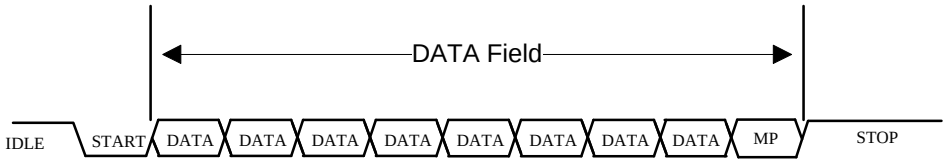
Figure 15-12. UART MP Mode Bus Connections



The main properties of UART_MP mode are:

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

Figure 15-13. UART MP Data Frame

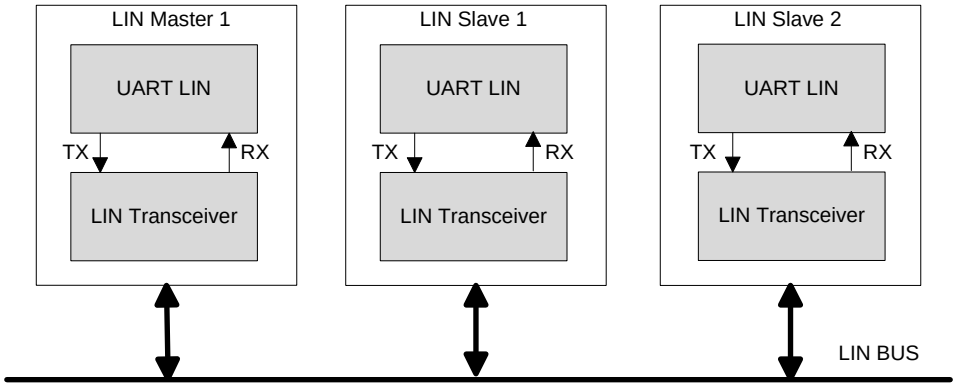


The SCB can be used as either master or slave device in UART_MP mode. Both SCB_TX_CTRL and SCB_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device,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

UART Local Interconnect Network (LIN) Mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with single master-multi slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports both LIN master and slave functionality. The LIN specification defines both physical layer (layer1) and data link layer (layer 2). Figure 15-14 illustrates the UART_LIN and LIN Transceiver.

Figure 15-14. UART_LIN and LIN Transceiver

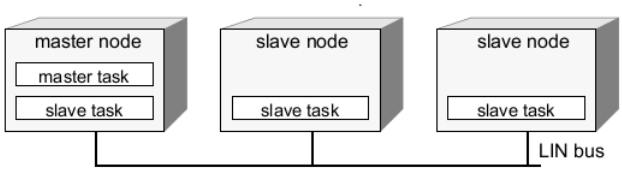


LIN protocol defines two tasks:

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

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

Figure 15-15. LIN Bus Nodes and Tasks

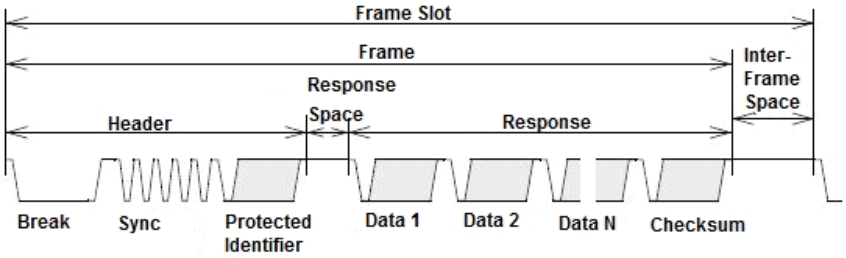


LIN Frame Structure

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

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

Figure 15-16. LIN Frame Structure



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

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

Figure 15-17. LIN Break Field

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

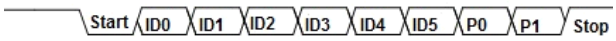
Figure 15-18. LIN Sync Field

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

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

Figure 15-19 shows the PID field structure.

Figure 15-19. PID Field



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

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

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

LIN Frame Types

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

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

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

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

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

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

The frame identifier for diagnostic frame is:

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

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

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

LIN Go-To-Sleep and Wake-Up

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

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

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

Configuring the SCB as Standard UART interface

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

1. Configure the SCB as UART interface by writing '10' to the MODE field (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCB_UART_CTRL register.
3. To enable the UART MP Mode or UART LIN Mode, write '1' to the MP_MODE (bit 10) or LIN_MODE (bit 12) respectively of the SCB_UART_RX_CTRL register.
4. Follow steps 2 to 5 described in Enabling and Initializing UART on page 128.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.3.3.2 SmartCard (ISO7816)

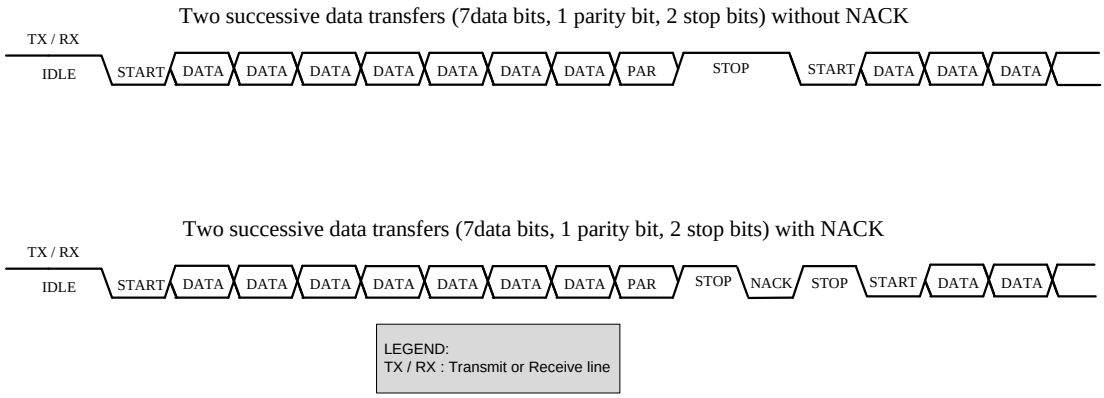
ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic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

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

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

Figure 15-20 illustrates the SmartCard protocol.

Figure 15-20. SmartCard Example



The communication Baud rate for ISO7816 is given as:

Baud rate= f_{7816} × (D/F)

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

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

Configuring SCB as UART SmartCard Interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PRoC® BLE:

CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a Smart-Card protocol by writing '01' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Follow steps 2 to 5 described in Enabling and Initializing UART on page 128.

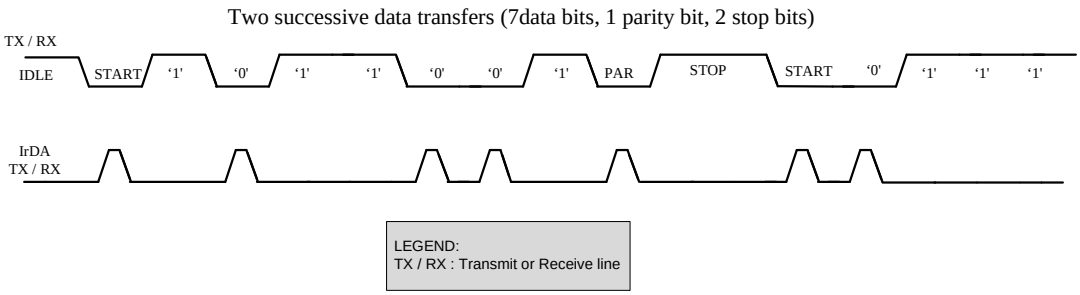
15.3.3.3 IrDA

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

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

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

Figure 15-21. IrDA Example



Configuring the SCB as UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Enable the Median filter on the input interface line by writing '1' to MEDIAN (bit 9) of the SCB_RX_CTRL register.
4. Configure the SCB as described in Enabling and Initializing UART on page 128.

15.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 15-9. For more information on these registers, see PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

Table 15-9. UART Registers

Register Name	Operation
SCB_CTRL	Enables the SCB; selects the type of serial interface (SPI, UART, I2C)
SCB_UART_CTRL	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCB_UART_RX_STATUS	Used to specify the BR_COUNTER value that determines the bit period. This is used to set the accuracy of the SCB clock. This value provides more granularity than the OVS bit in SCB_CTRL register.
SCB_UART_TX_CTRL	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.

Table 15-9. UART Registers

Register Name	Operation
SCB_UART_RX_CTRL	Performs same function as SCB_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error.
SCB_TX_CTRL	Used to specify the data frame width and to specify whether MSB or LSB is the first bit in transmission.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_UART_FLOW_CONTROL	Configures flow control for UART transmitter.

15.3.5 UART Interrupts

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

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

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

15.3.6 Enabling and Initializing UART

The UART must be programmed in the following order:

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

Table 15-10. SCB_UART_CTRL Register

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

Table 15-11. SCB_TX_CTRL/SCB_RX_CTRL Registers

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

Table 15-12. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

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

Table 15-13. SCB_CTRL Register

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

15.4 Inter Integrated Circuit (I2C)

This section explains the I2C implementation in PRoC. For more information on the I2C protocol specification, refer to the I2C-bus specification available on the NXP website.

15.4.1 Features

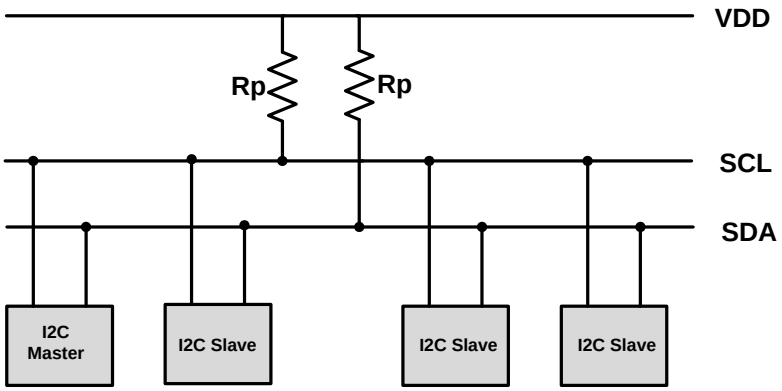
This block supports the following features:

- Master, slave, and master/slave mode
- Slow-mode (50 kbps), standard-mode (100 kbps), fast-mode (400 kbps), and fast-mode plus (1000 kbps) data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of 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

15.4.2 General Description

Figure 15-22 illustrates an example of an I2C communication network.

Figure 15-22. 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. PRoC also supports 10-bit address matching for I2C with firmware support.

15.4.3 Terms and Definitions

Table 15-14 explains the commonly used terms in an I2C communication network.

Table 15-14. Definition of I2C Bus Terminology

Term	Description
Transmitter	The device that sends data to the bus
Receiver	The device that receives data from the bus
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer

Table 15-14. Definition of I2C Bus Terminology

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

15.4.3.1 Bus Stalling (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 postponing the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock.

15.4.3.2 Bus Arbitration

The 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.

15.4.4 I2C Modes of Operation

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

Table 15-15 illustrates the I2C modes of operation.

Table 15-15. I2C Modes

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

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

Table 15-16. I2C Bus Events Terminology

Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH

Table 15-16. I2C Bus Events Terminology

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

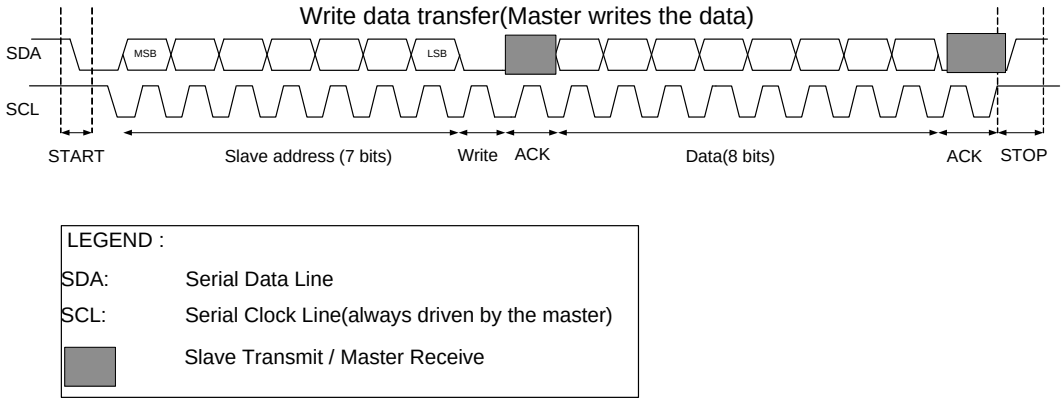
When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see Table 15-16, Figure 15-23, and Figure 15-24).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

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt, so that the device is ready to respond to any other master on the bus.

With all of these modes, there are two types of transfer - read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in Master Mode Transfer Examples on page 139, Slave Mode Transfer Examples on page 141, and Multi-Master Mode Transfer Example on page 145.

15.4.4.1 Write Transfer

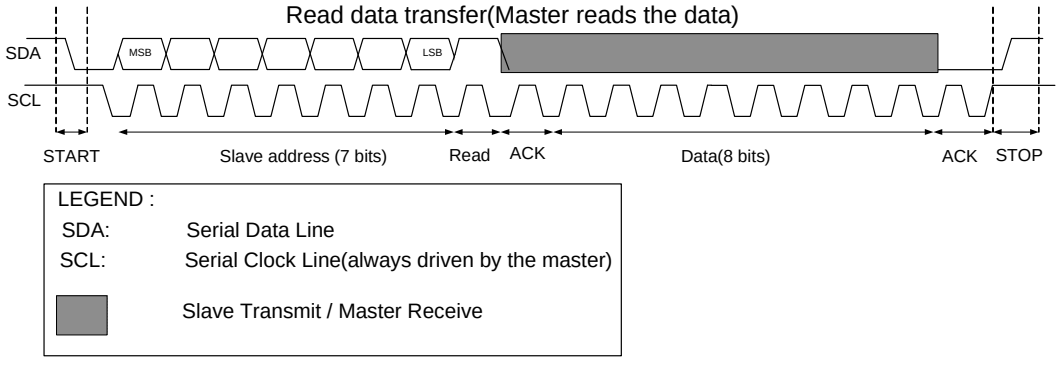
Figure 15-23. Master Write Data Transfer



- A typical write transfer begins with the master generating a START condition on the I2C bus. The master then writes a 7-bit I2C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match any of the slave devices or if the addressed device does not want to acknowledge the request, it transmits a no acknowledgement (NACK) by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- The master may transmit data to the bus if it receives an acknowledgement. The addressed slave transmits an acknowledgement to confirm the receipt of every byte of data written. Upon receipt of this acknowledgement, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

15.4.4.2 Read Transfer

Figure 15-24. Master Read Data Transfer



- A typical read transfer begins with the master generating a START condition on the I2C bus. The master then writes a 7-bit I2C slave address and a read indicator ('1') after the START condition. The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match with that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgement (NACK) is transmitted by not pulling the SDA line low. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgement signal. The master transmits an acknowledgement to confirm the receipt of

each data byte sent by the slave. Upon receipt of this acknowledgement, the addressed slave may transmit another data byte.

- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.

15.4.5 Easy I2C (EZI2C) Protocol

The Easy I2C (EZI2C) protocol is a unique communication scheme built on top of the I2C protocol by Cypress. It uses a software wrapper around the standard I2C protocol to communicate to an I2C 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 EZ 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 I2C 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: an EZ write of data from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

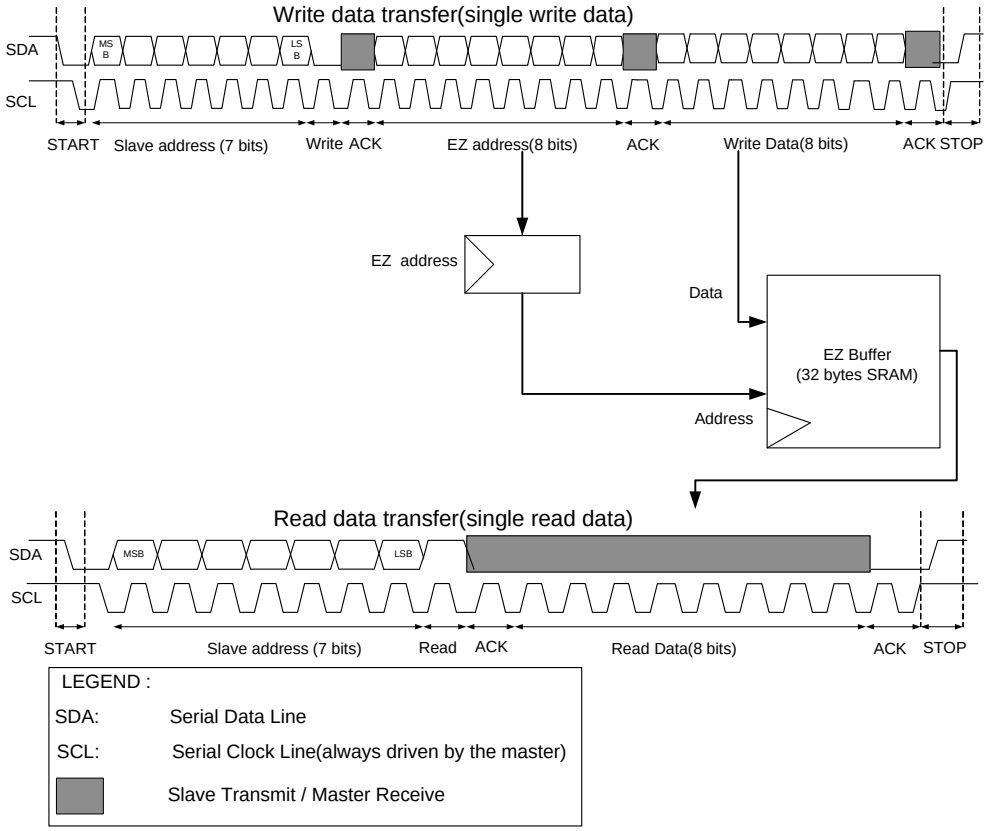
15.4.5.1 Memory Array Write

An EZ write to a memory array index is by means of an I2C write transfer. The first transmitted write data is used to send an EZ address from the master to the slave. The five lowest significant bits of the write data are used as the "new" EZ address at the slave. Any additional write data elements in the write transfer are bytes that are written to the memory array. The EZ address is automatically incremented by the slave as bytes are written into the memory array. If the number of continuous data bytes written to the EZI2C buffer exceeds EZI2C buffer boundary, it overwrites the last location for every subsequent byte.

15.4.5.2 Memory Array Read

An EZ read from a memory array index is by means of an I2C read transfer. The EZ read relies on an earlier EZ write to have set the EZ address at the slave. The first received read data is the byte from the memory array at the EZ address memory location. The EZ address is automatically incremented as bytes are read from the memory array.

Figure 15-25. EZI2C Write and Read Data Transfer



15.4.6 I2C Registers

The I2C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in Table 15-17.

Table 15-17. I2C Registers

Register	Function
SCB_CTRL	Enables the SCBI2C block and selects the type of serial interface (SPI, UART, I2C). Also used to select internally and externally clocked operation and EZ and non-EZ modes of operation.
SCB_I2C_CTRL	Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status.
SCB_I2C_STATUS	Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address.
SCB_I2C_M_CMD	Enables the master to generate START, STOP, and ACK/NACK signals.
SCB_I2C_S_CMD	Enables the slave to generate ACK/NACK signals.
SCB_STATUS	Indicates whether the externally clocked logic is using the EZ memory. This bit can be used by software to determine whether it is safe to issue a software access to the EZ memory.
SCB_I2C_CFG	Configures filters, which remove glitches from the SDA and SCL lines.
SCB_TX_CTRL	Specifies the data frame width; also used to specify whether MSB or LSB is the first bit in transmission.
SCB_TX_FIFO_CTRL	Specifies the trigger level, clearing of the transmitter FIFO and shift registers, and FREEZE operation of the transmitter FIFO.
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.

Table 15-17. I2C Registers

Register	Function
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 PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.4.7 I2C Interrupts

The fixed-function I2C 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 PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

15.4.8 Enabling and Initializing the I2C

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

15.4.8.1 Configuring for I2C Standard (Non-EZ) Mode

The I2C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to Table 15-18. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-19.
   a. Specify the data frame width.

b. Specify that MSB is the first bit to be transmitted/ received.

3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in Table 15-20.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.

Program the SCB_CTRL register to enable the I2C block and select the I2C mode. These register bits are shown in Table 15-21. For a complete description of the I2C registers, see PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

Table 15-18. SCB_I2C_CTRL Register

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

Table 15-19. SCB_TX_CTRL/SCB_RX_CTRL Register

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

Table 15-20. SCB_TX_FIFO_CTRL/ SCB_RX_FIFO_CTRL

Bits	Name	Description
[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 15-21. SCB_CTRL Registers

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

15.4.8.2 Configuring for EZI2C Mode

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 Configuring for I2C Standard (Non-EZ) Mode.
3. Set the S_READY_ADDR_ACK (bit 12) and S_READY_DATA_ACK (bit 13) bits of the SCB_I2C_CTRL register.

15.4.9 Internal and External Clock Operation in I2C

The I2C block supports both internally and externally clocked operation for data-rate generation. Internally

clocked operations use a clock signal derived from the PRoC 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 73.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality.

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

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

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

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

15.4.9.1 I2C Non-EZ Mode of Operation

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

EC_AM_MODE is '0' and EC_OP_MODE is '0'. This setting only works in Active and Sleep system power modes. All the functionality of the I2C is provided in the internally clocked domain.

EC_AM_MODE is '1' and EC_OP_MODE is '0'. This setting works in Active, Sleep, and Deep-Sleep system power modes. I2C address matching is performed by the externally clocked logic in Active, Sleep, and Deep-Sleep system power modes. When the externally clocked logic matches the address, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wakeup the CPU.

- 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').

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

I2C (Non-EZ) Mode				
System Power Mode	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 0	EC_AM_MODE = 1
Active and Sleep	Address match using internal clock. Operation using internal clock.	Address match using external clock. Operation using internal clock.	Not supported	
Deep-Sleep	Not supported	Address match using external clock. Operation using internal clock.		
Hibernate	The SCB is not available in these modes (see the Power Modes chapter on page 89).			
Stop				

15.4.9.2 I2C EZ Operation Mode

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

Table 15-23. I2C Operation in EZ Mode

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

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

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

15.4.10 Wake up from Sleep

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

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

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

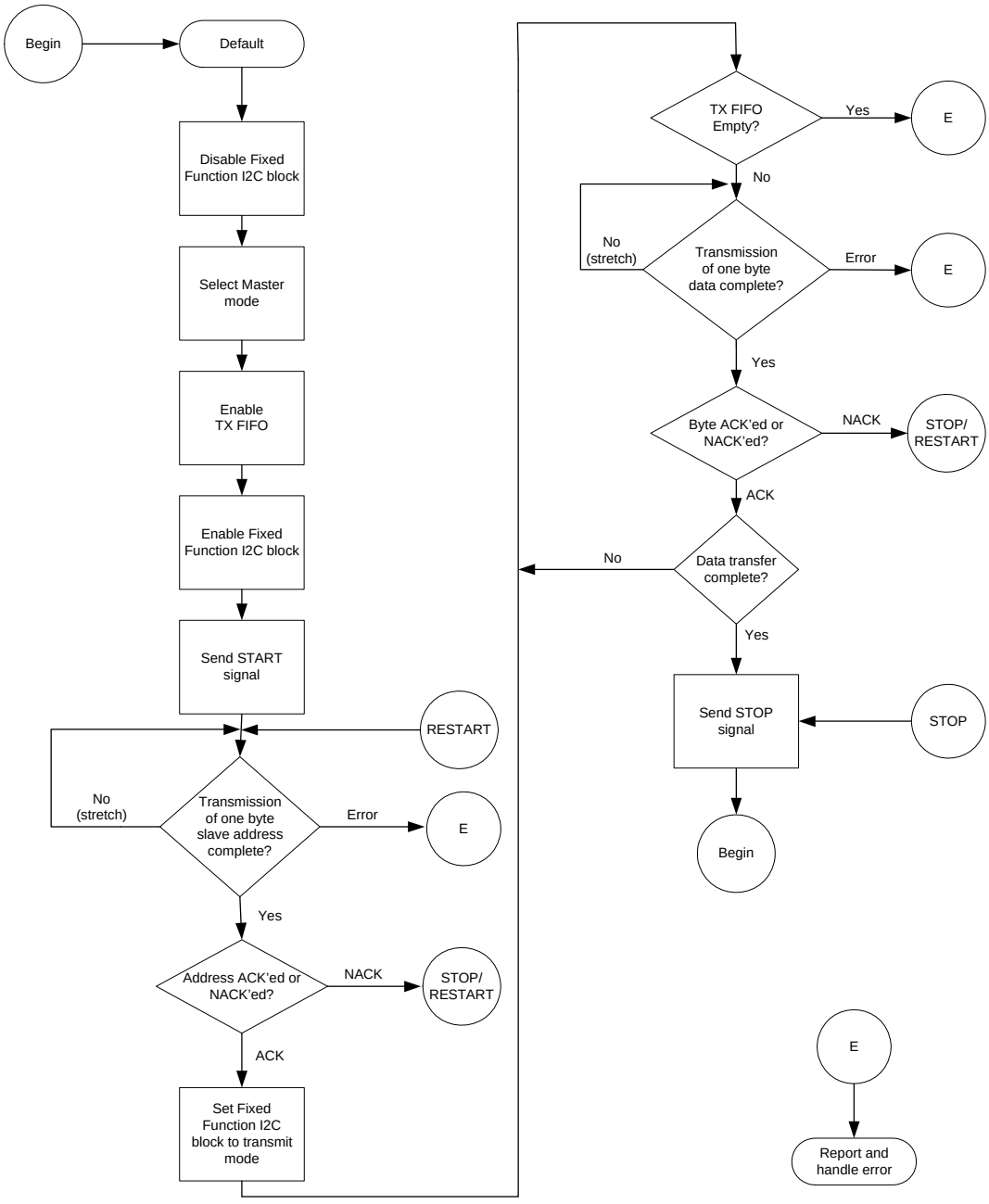
Note The interrupt bit WAKE_UP (bit 0) of the SCB_INTR_I2C_EC register must be enabled for the I2C to wake up the device on slave address match while switching to the Sleep mode.

15.4.11 Master Mode Transfer Examples

Master mode transmits or receives data.

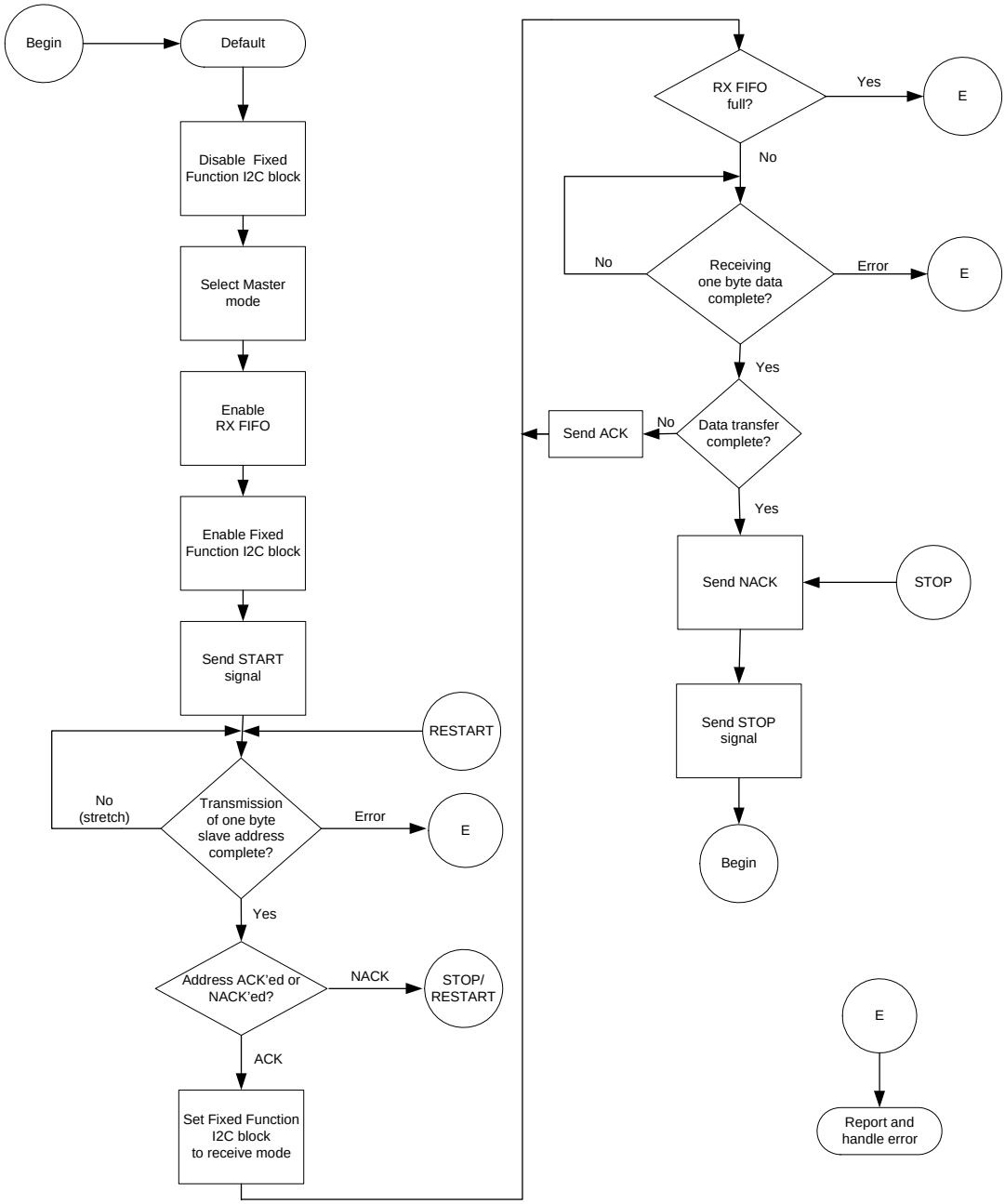
15.4.11.1 Master Transmit

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



15.4.11.2 Master Receive

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

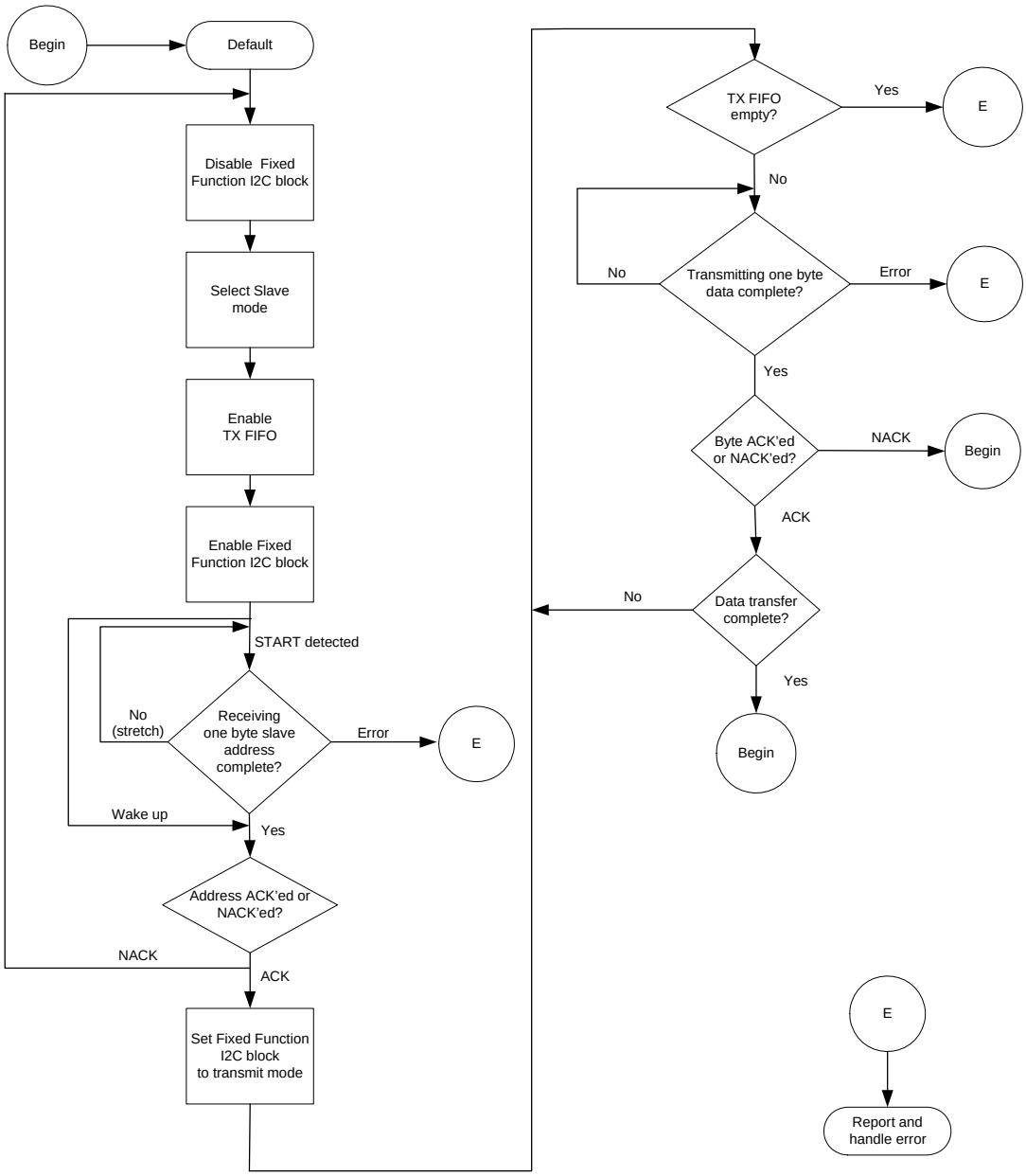


15.4.12 Slave Mode Transfer Examples

Slave mode transmits or receives data.

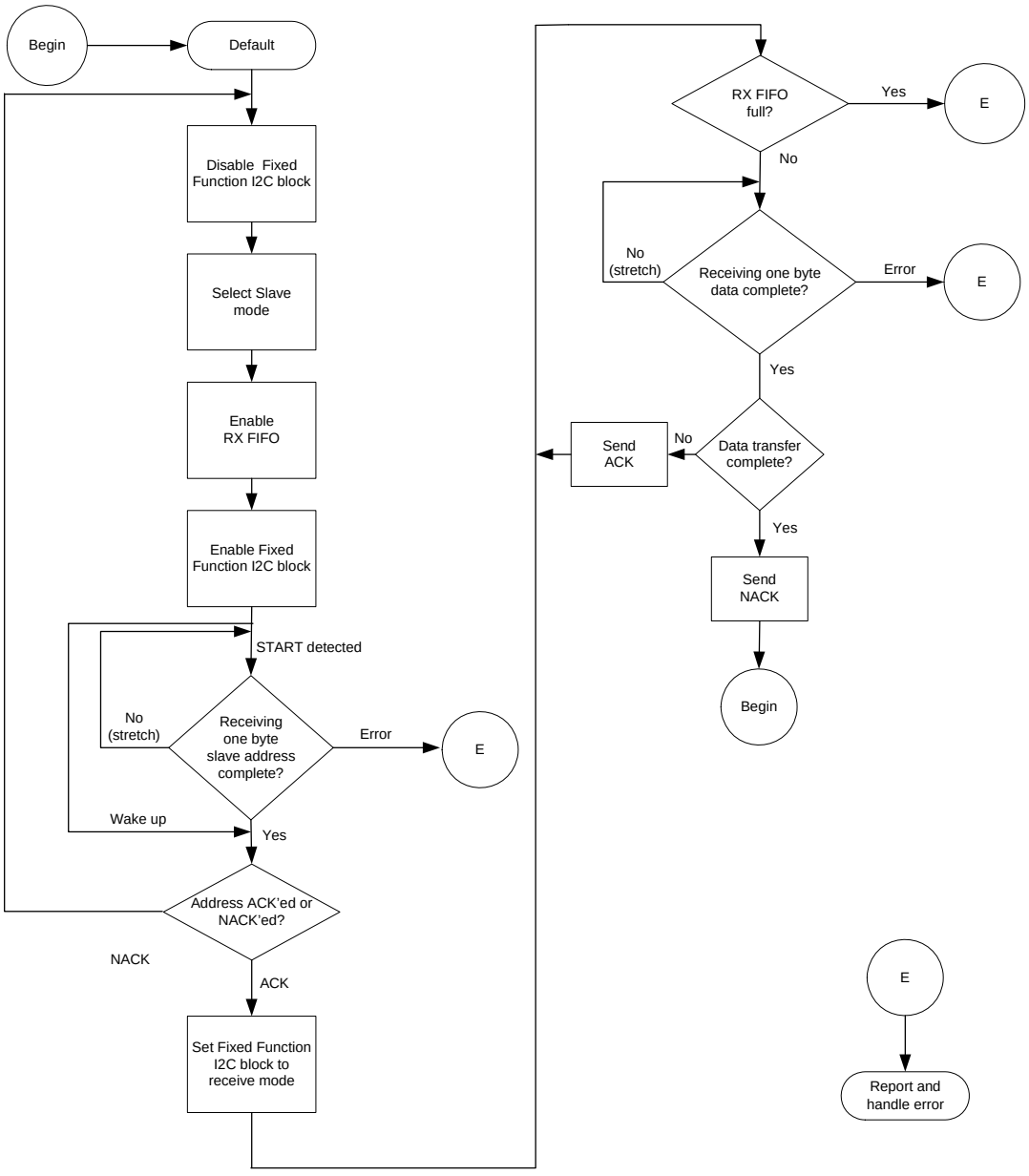
15.4.12.1 Slave Transmit

Figure 15-28. Slave Mode Write Operation Flow Chart



15.4.12.2 Slave Receive

Figure 15-29. Slave Mode Read Operation Flow Chart

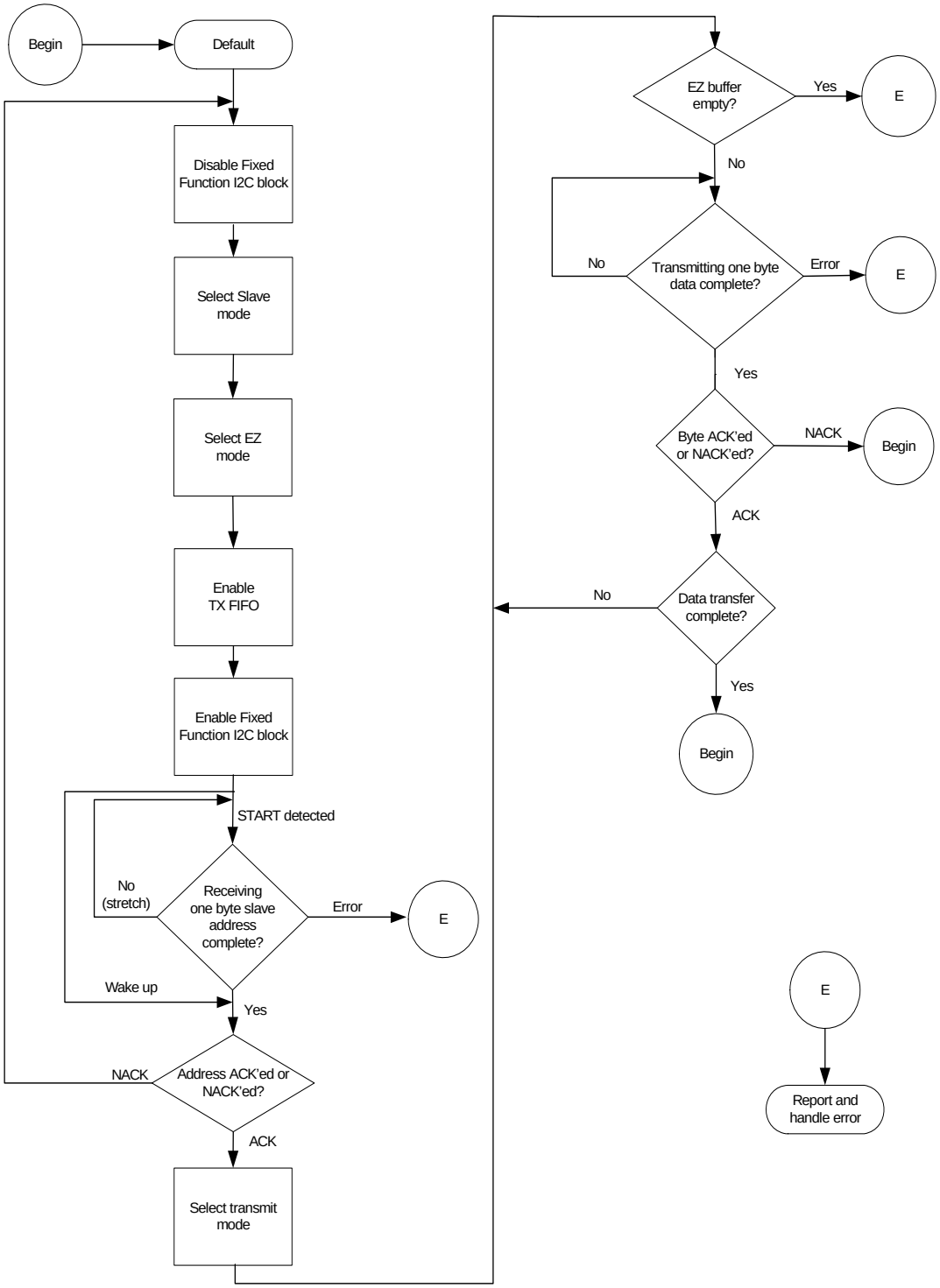


15.4.13 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

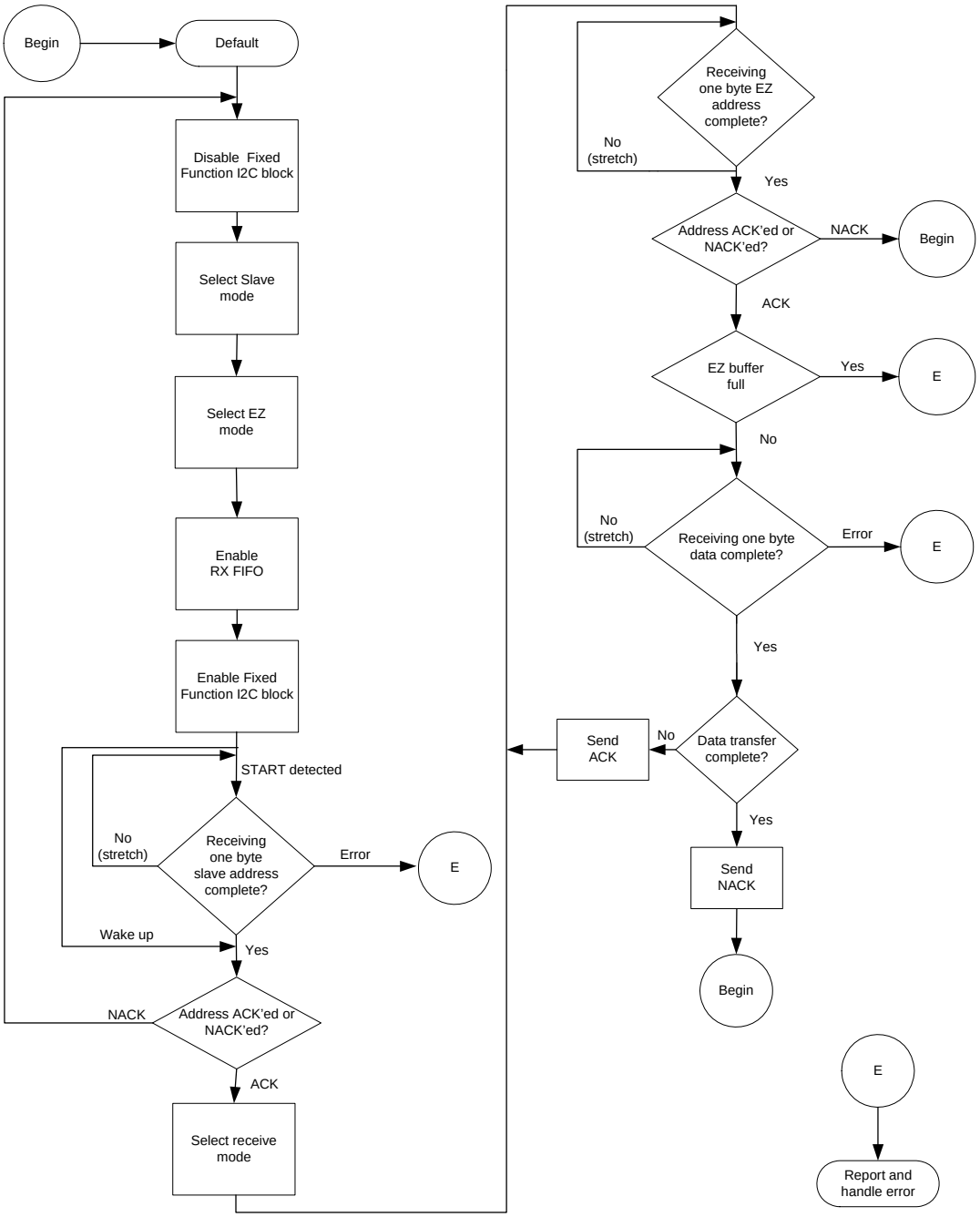
15.4.13.1 EZ Slave Transmit

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



15.4.13.2 EZ Slave Receive

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

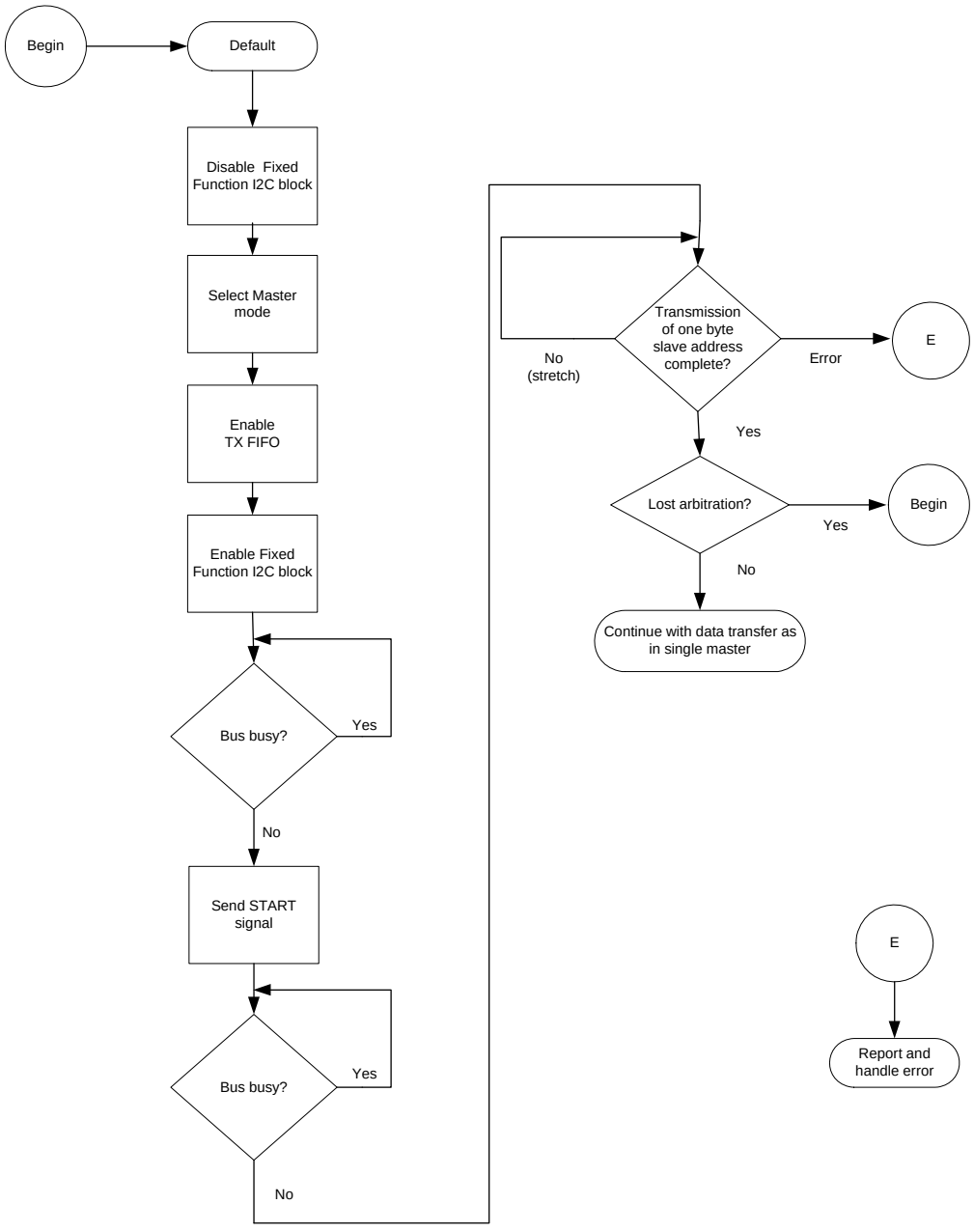


15.4.14 Multi-Master Mode Transfer Example

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

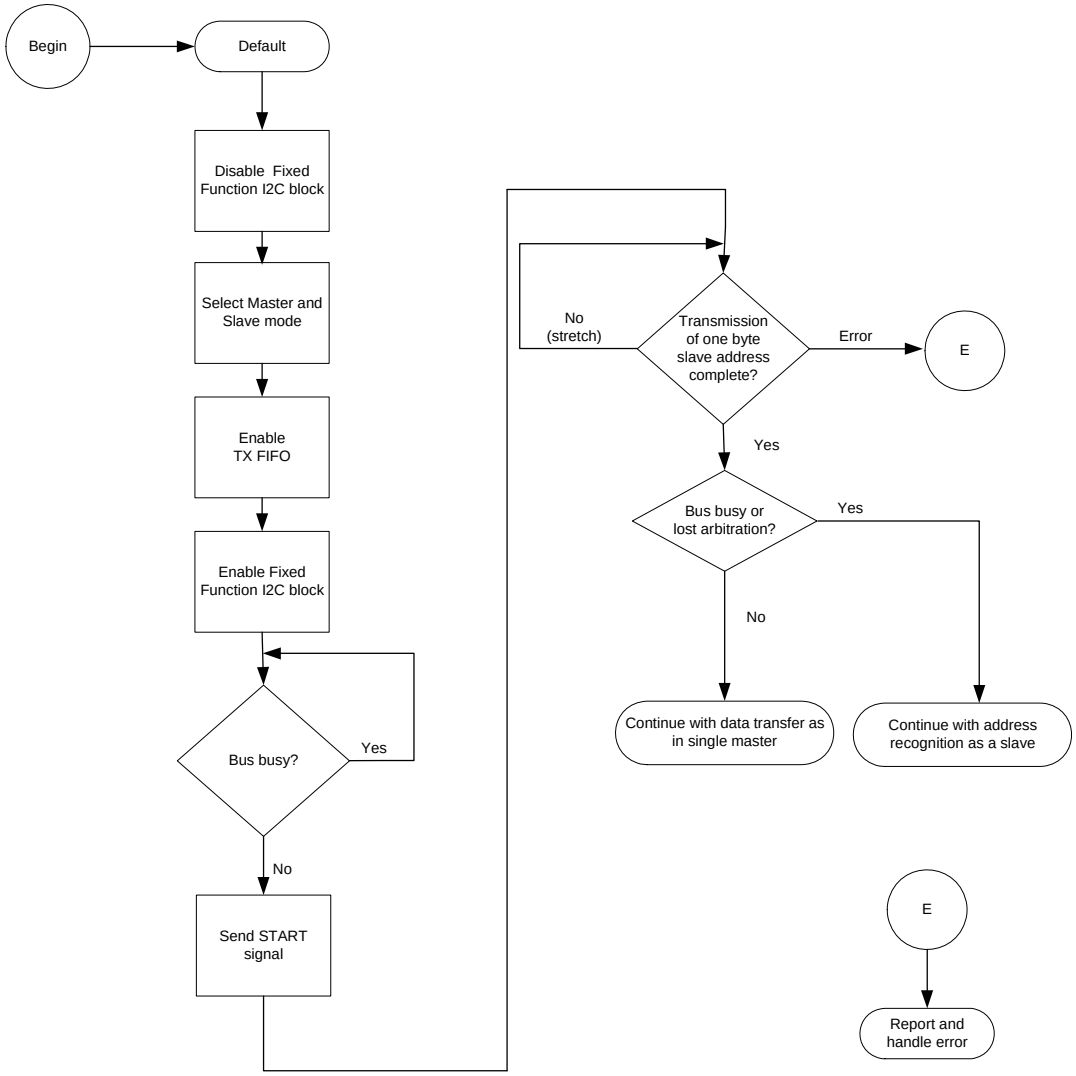
15.4.14.1 Multi-Master - Slave Not Enabled

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



15.4.14.2 Multi-Master - Slave Enabled

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



16. Timer, Counter, and PWM

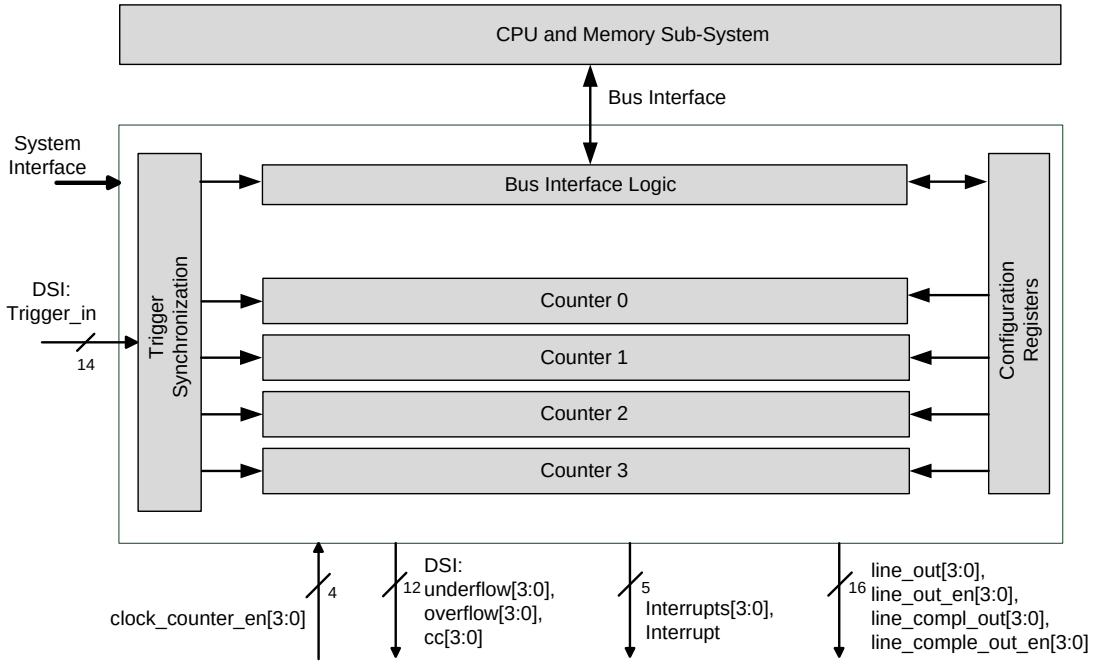
The Timer, Counter, and Pulse Width Modulator (TCPWM) block in PRoC implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular event 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

- Four 16-bit timers, counters, or pulse width modulators (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 pre-scaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count – The final value in the counter register is reached
  - Capture/Compare – The count is captured to the capture/compare register or the counter value equals the compare value
- Synchronized counters – The counters can reload, start, stop, and count at the same time
- DSI output signals for each counter to indicate underflow, overflow, and capture/compare match events
- Complementary line output for PWMs
- Selectable start, reload, stop, count, and capture event signals for each TCPWM from up to 14 DSI signals 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 with DSI: Routes signals to or from the universal digital block (UDB) and TCPWM block. It consists of input triggers (such as reload, start, stop, count, and capture) and output signals (such as overflow (OV), underflow (UN), and capture/compare (CC)). Any GPIO can be used as the input trigger signal.
- Interrupts: Provides interrupt request signals from each counter, based on terminal count (TC) or CC conditions, and a combined interrupt signal generated by the logical OR of all four interrupt request signals.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem.

This TCPWM block can be configured by writing to the TCPWM registers. See TCPWM Registers on page 167 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).Status registers are cleared after the counter is disabled.

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 Pre-Scaling: counter_clock can be pre-scaled, with divider values of 1, 2, 4… 64, 128. This 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 Pre-Scale Counter Clock

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

Note Clock pre-scaling cannot be done in quadrature mode and pulse width modulation mode with dead time (PWM-DT).

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. TCPWM Trigger Selection and Event Detection

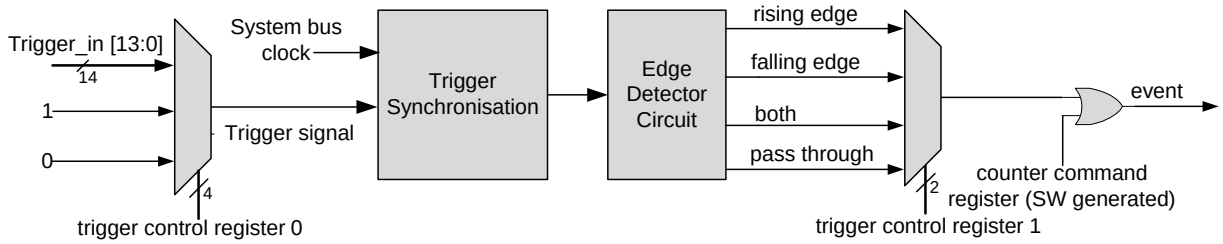


Figure 16-2 shows the trigger selection and event detection in the TCPWM block. The trigger control register 0 (TCPWM_CNT_TR_CTRL0) selects one of the 14 trigger inputs as the event signal. Additionally, a constant '0' and '1' signals are available to be used as the event signal.

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

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

- **Reload:** A reload event initializes and starts the counter.
  - In up counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with ‘0’.
  - In down counting mode, the counter is initialized with the period value stored in the TCPWM_CNT_PERIOD register.
  - In up/down counting mode, the count register is initialized with ‘1’.
  - In quadrature mode, the reload event acts as a quadrature index event. An index/reload event indicates a completed rotation and can be used to synchronize quadrature decoding.
- **Start:** A start event is used to start counting; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
  - In quadrature mode, the start event acts as quadrature phase input phiB, which is explained in detail in Quadrature Decoder Mode on page 157.
- **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.

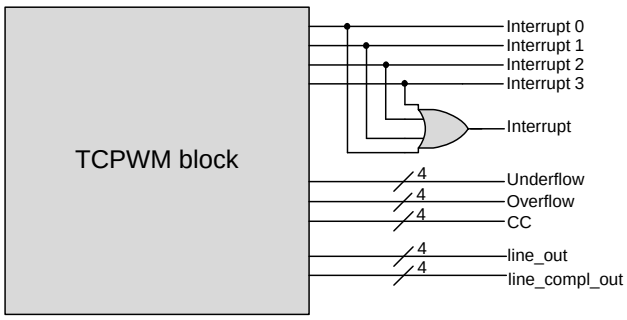
Notes

- All trigger inputs are synchronized to the HFCLK.
- In the Quadrature mode, edge detection is performed with the counter clock. In the other five modes, the edge detection is done using the gated version of the HFCLK.

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 one cycle of the HFCLK. For reliable operation, the condition that causes this trigger should be less than a quarter of the HFCLK. For example, if the HFCLK is running at 24 MHz, the condition causing the trigger should occur at a frequency less than 6 MHz.

16.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific. All four interrupt output signals from the four 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 (see 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.
Hibernate	In this mode, the power to this block is switched off. Configuration registers will lose their state.
Stop	In this mode, the power to this block is switched off. Configuration registers will lose 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.

Table 16-5. Operational Mode Configuration

Mode	MODE Field [26:24]	Description
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

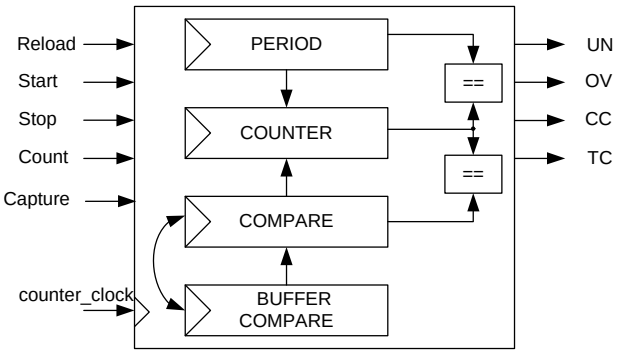
Counting Modes	UP_DOWN_MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until 0 is reached. A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

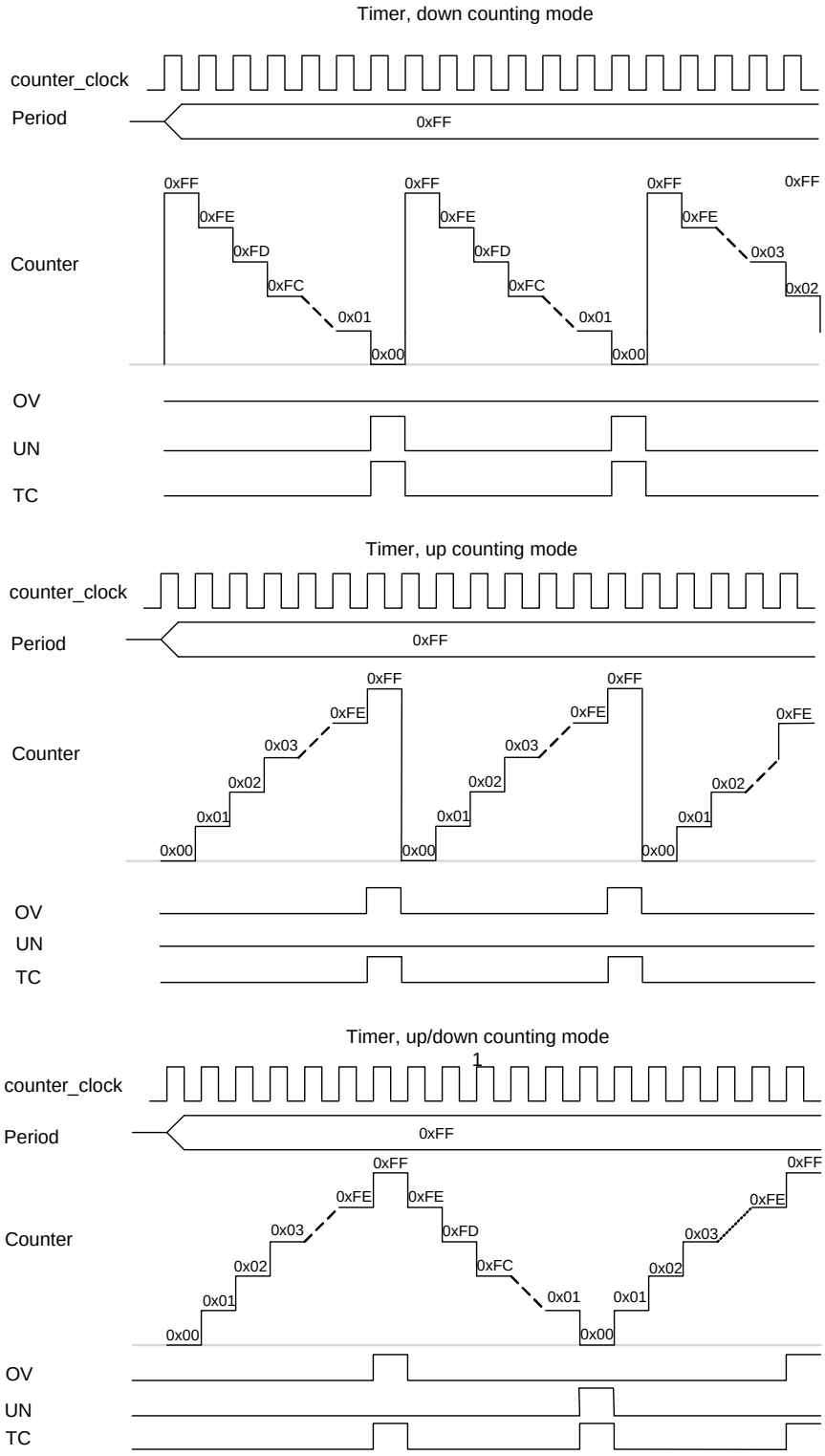
- The timer is an up, down, and up/down counter.
- The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
  Note It is not recommended to write values to this register while the counter is running.
- The period value for the timer is stored in the period register.

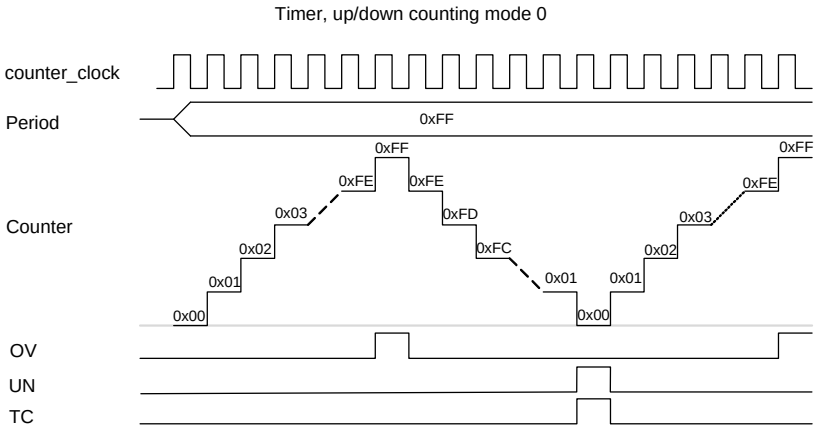
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with 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 (both modes 0 and 1 both), a period value of A results in 2*A counter cycles (1 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 one cycle of the HFCLK, as explained in Signals upon Trigger Conditions on page 150. 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.

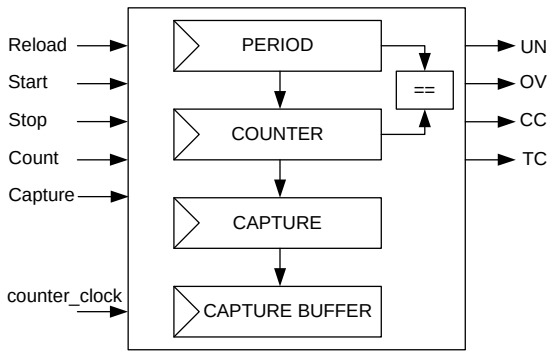
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit 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 pre-scaling 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 the TCPWM_CNT_CTRL register.
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 150.
12. Enable the counter by writing '1' to the corresponding bit in 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:

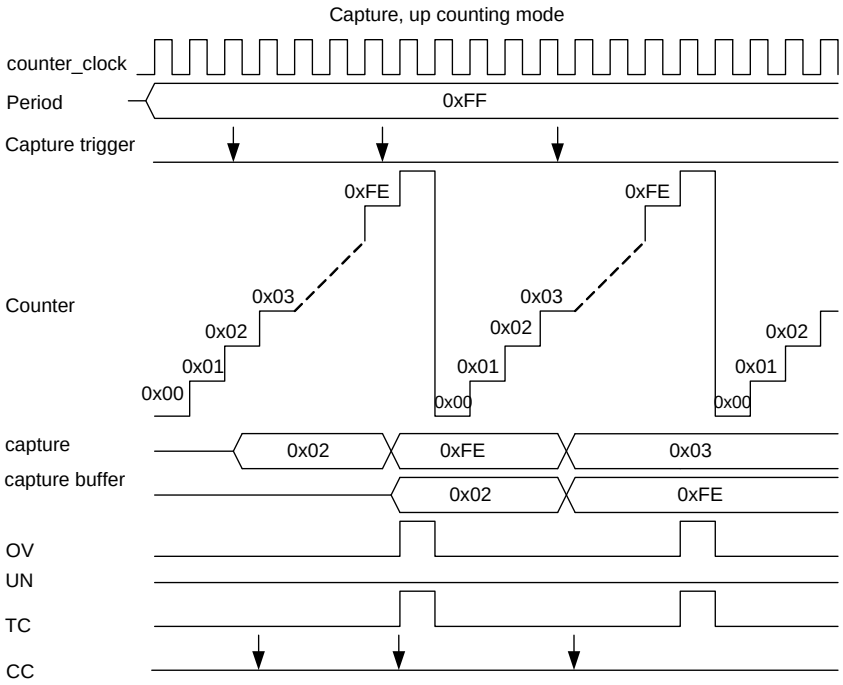
- During a capture event, generated either by hardware or software, the current count register value is copied to the capture register (TCPWM_CNT_CC) and the capture

register value is copied to the buffer capture register (TCPWM_CNT_CC_BUFF).

- A pulse on the CC output signal is generated when the counter value is copied to the capture register. This condition can also be used to generate an interrupt request.

Figure 16-7 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.

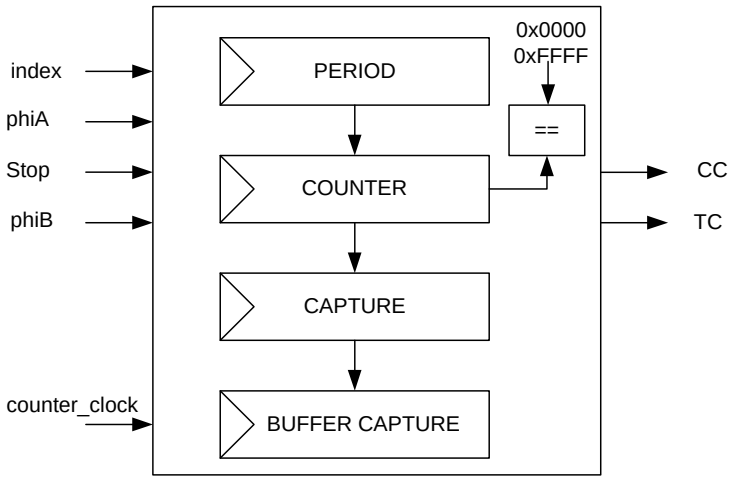
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock pre-scaling 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 150.
10. Enable the counter by writing '1' to the corresponding bit in 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.

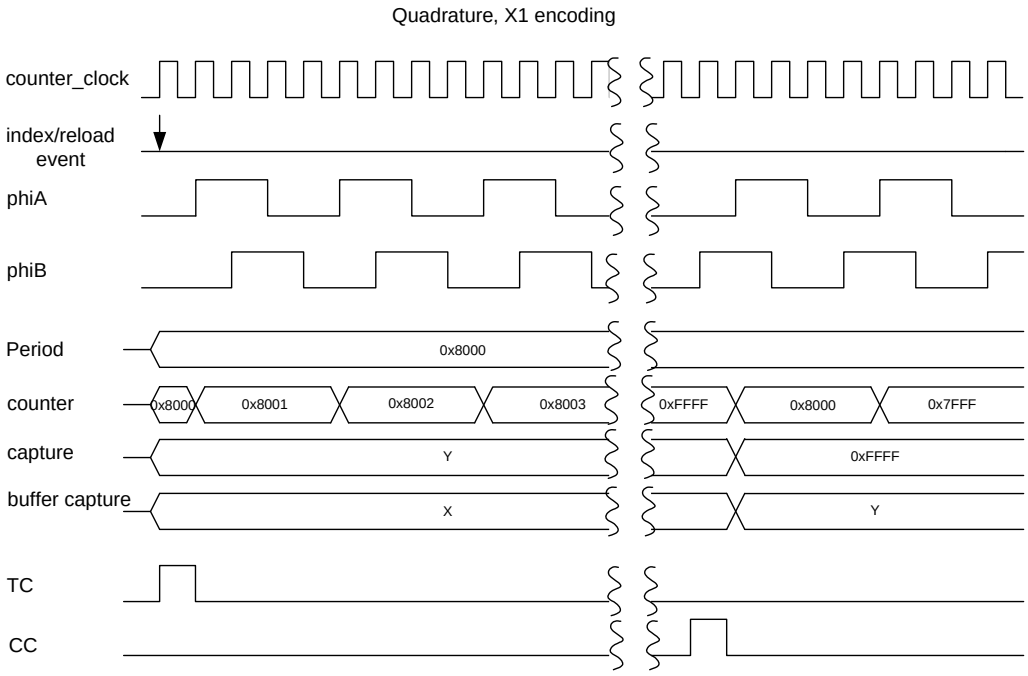
The Quadrature mode operation occurs as follows:

- 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 0x8000.
- 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 0x8000.

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

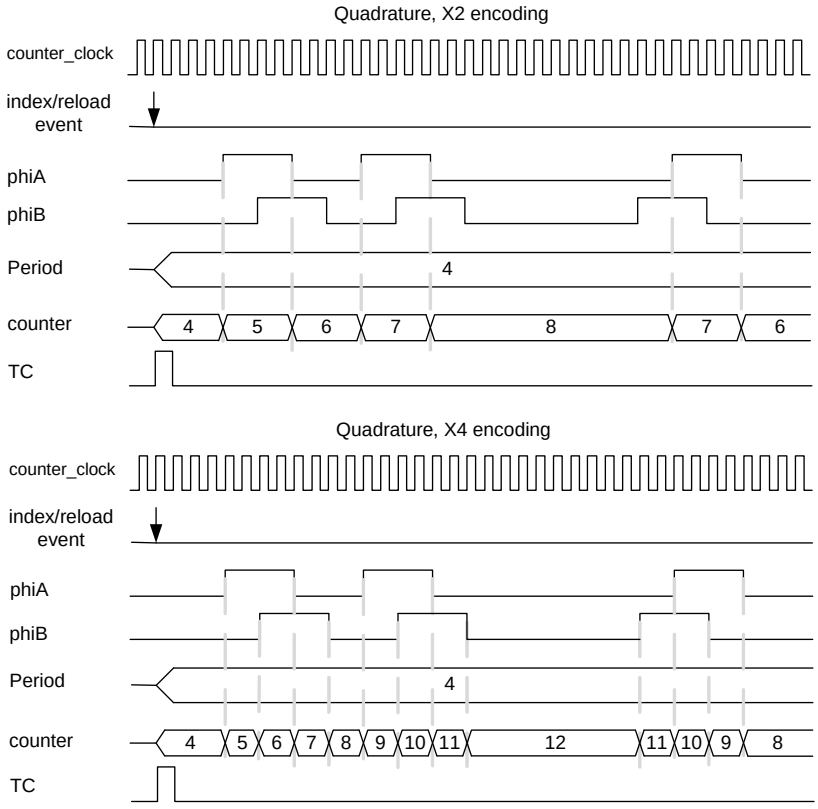


The quadrature phases are detected on the counter_clock. Within a single counter_clock period, the phases should not change value more than once.

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 corresponding bit in 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 150.
8. Enable the counter by writing '1' to the corresponding bit in 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

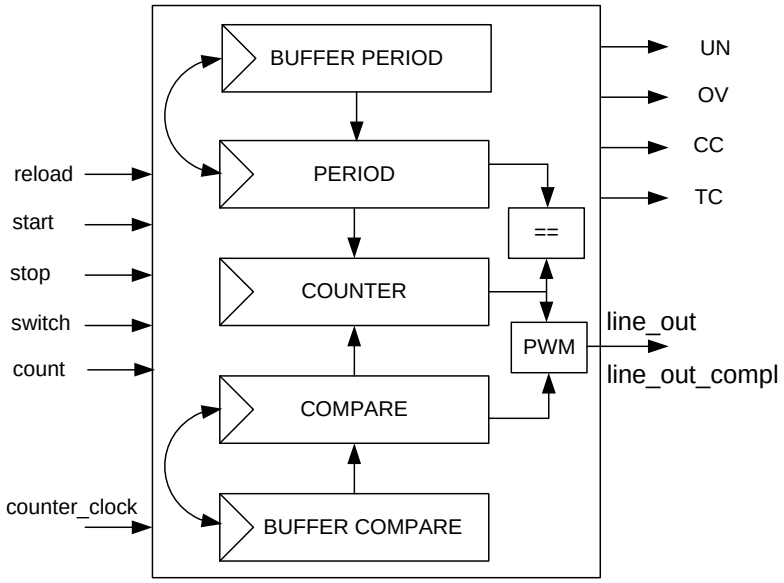
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

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 settings to be done are: underflow = clear, overflow = set, and CC = invert

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

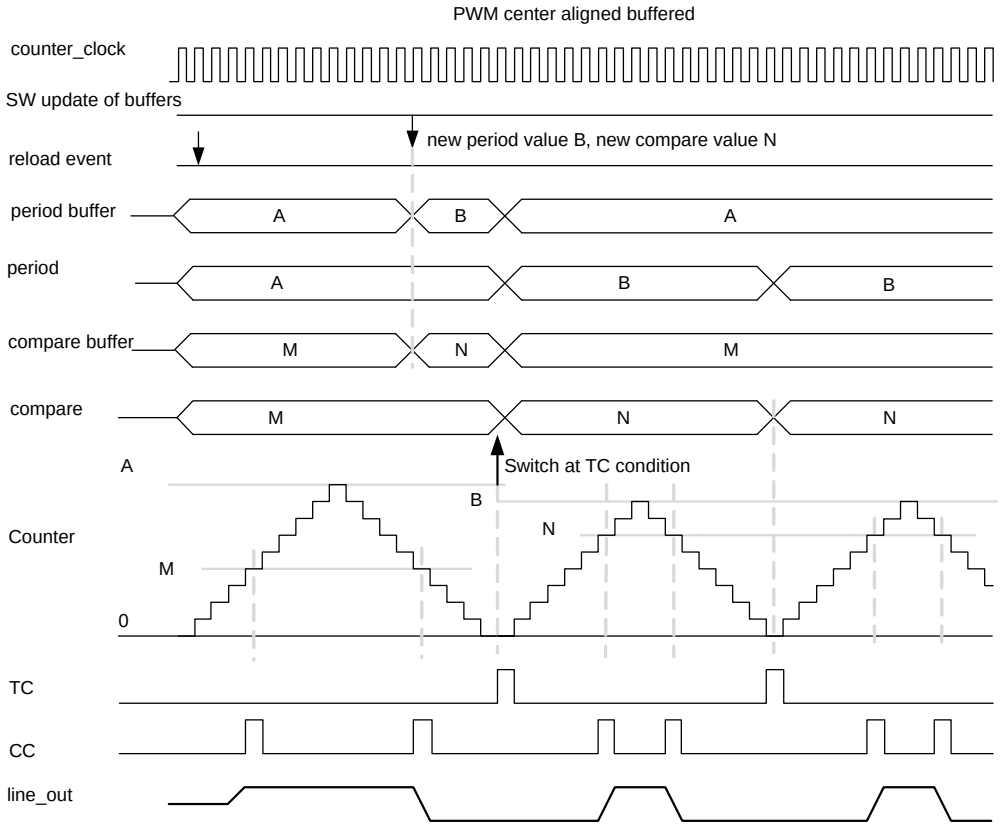
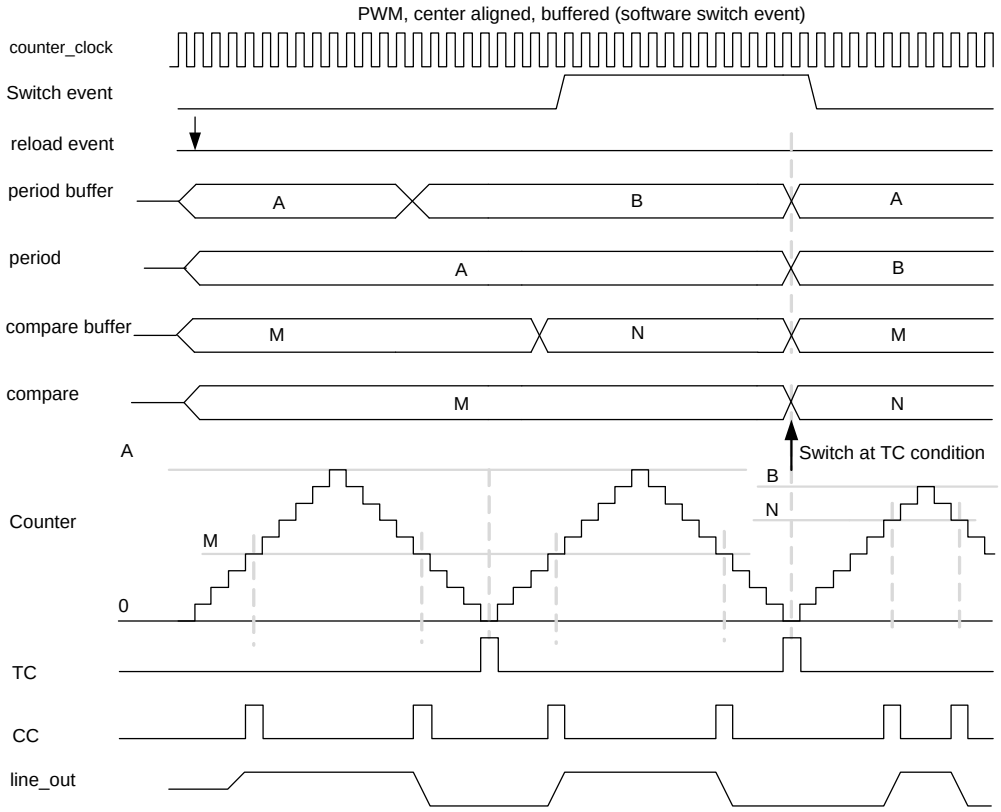


Figure 16-12 illustrates center-aligned PWM with software generated switch events:

- 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’). Ensure that within a PWM period, the period values remain the same.
- 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

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.

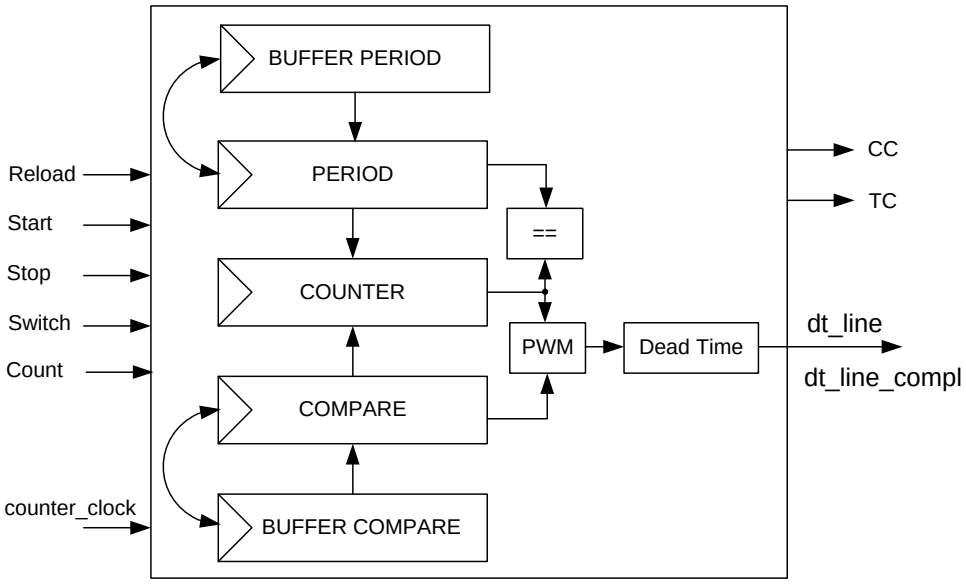
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock pre-scaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, 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.
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 150.
12. Enable the counter by writing '1' to the corresponding bit in 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.

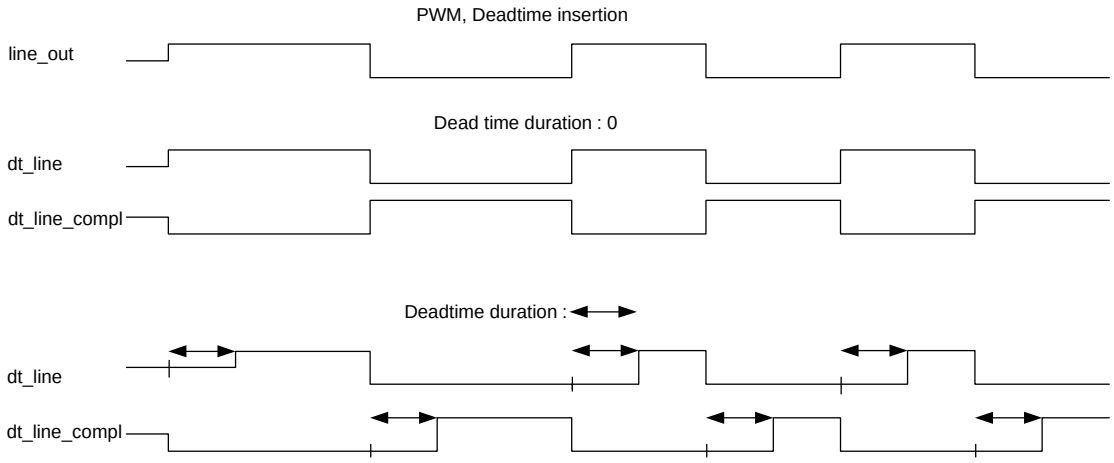
This mode follows PWM mode and supports the following features available with that mode:

- 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 pre-scaling.

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 corresponding bit in 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 160.

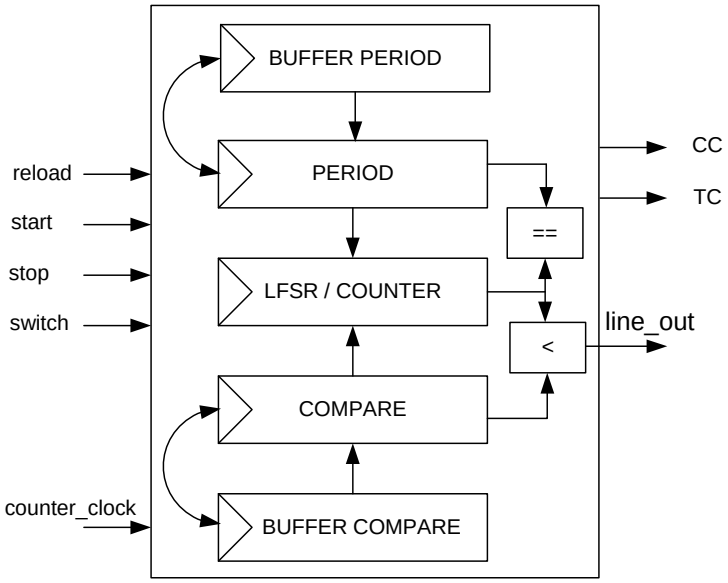
8. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by the 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 150.
12. Enable the counter by writing '1' to the corresponding bit in 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

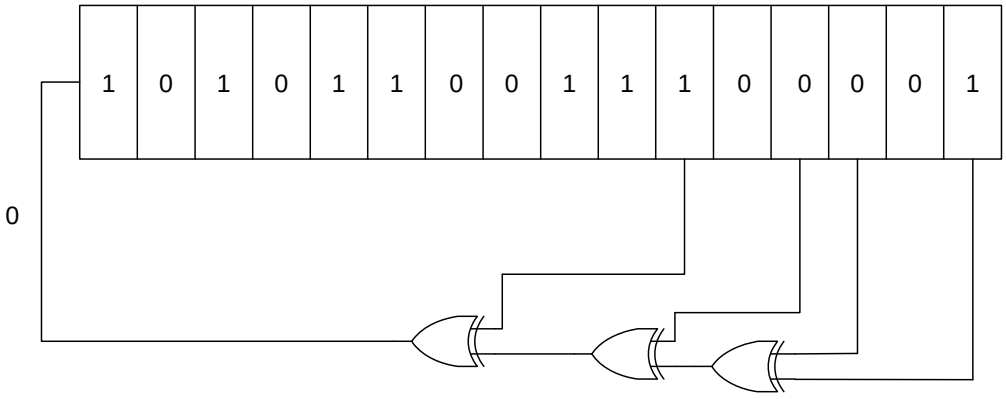
Figure 16-16. PWM-PR Mode Block Diagram



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.

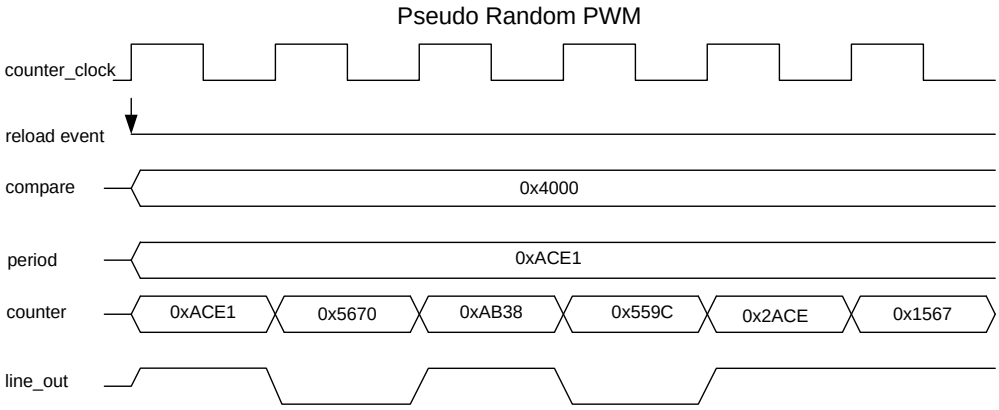
Figure 16-17. Pseudo-Random Sequence Generation using Counter Register



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 the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNT_PERIOD register and buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
6. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in Interrupts on page 150.
10. Enable the counter by writing '1' to the corresponding bit in 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_CNTx_CTRL	Counter control register	Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock pre-scaling, and counting direction
TCPWM_CNTx_STATUS	Counter status register	Reads the direction of counting, dead time duration, and clock pre-scaling; checks if counter is running
TCPWM_CNTx_COUNTER	Count register	Contains the 16-bit counter value
TCPWM_CNTx_CC	Counter compare/capture register	Captures the counter value or compares the value with the counter value
TCPWM_CNTx_CC_BUFF	Counter buffered compare/capture register	Buffer register for counter CC register; switches compare value
TCPWM_CNTx_PERIOD	Counter period register	Contains upper value of the counter
TCPWM_CNTx_PERIOD_BUFF	Counter buffered period register	Buffer register for counter period register; switches period value
TCPWM_CNTx_TR_CTRL0	Counter trigger control register 0	Selects trigger for specific counter events
TCPWM_CNTx_TR_CTRL1	Counter trigger control register 1	Determines edge detection for specific counter input signals
TCPWM_CNTx_TR_CTRL2	Counter trigger control register 2	Controls counter output lines upon CC, OV, and UN conditions
TCPWM_CNTx_INTR	Interrupt request register	Sets the register bit when TC or CC condition is detected
TCPWM_CNTx_INTR_SET	Interrupt set request register	Sets the corresponding bits in the interrupt request register
TCPWM_CNTx_INTR_MASK	Interrupt mask register	Mask for interrupt request register
TCPWM_CNTx_INTR_MASKED	Interrupt masked request register	Bitwise AND of interrupt request and mask registers

Note 'x' in the register name denotes the number of TCPWM. For example, the interrupt mask register for TCPWM0 is TCPWM_CNT0_INTR_MASK.

17. Integrated Inter-IC Sound Bus

PRoC™ BLE contains an Integrated Inter-IC Sound Bus (I^2S) block that operates in master mode. This section explains the I^2S master mode implementation in PRoC BLE. For more information on the I^2S protocol, refer to the I^2S bus specification.

17.1 Features

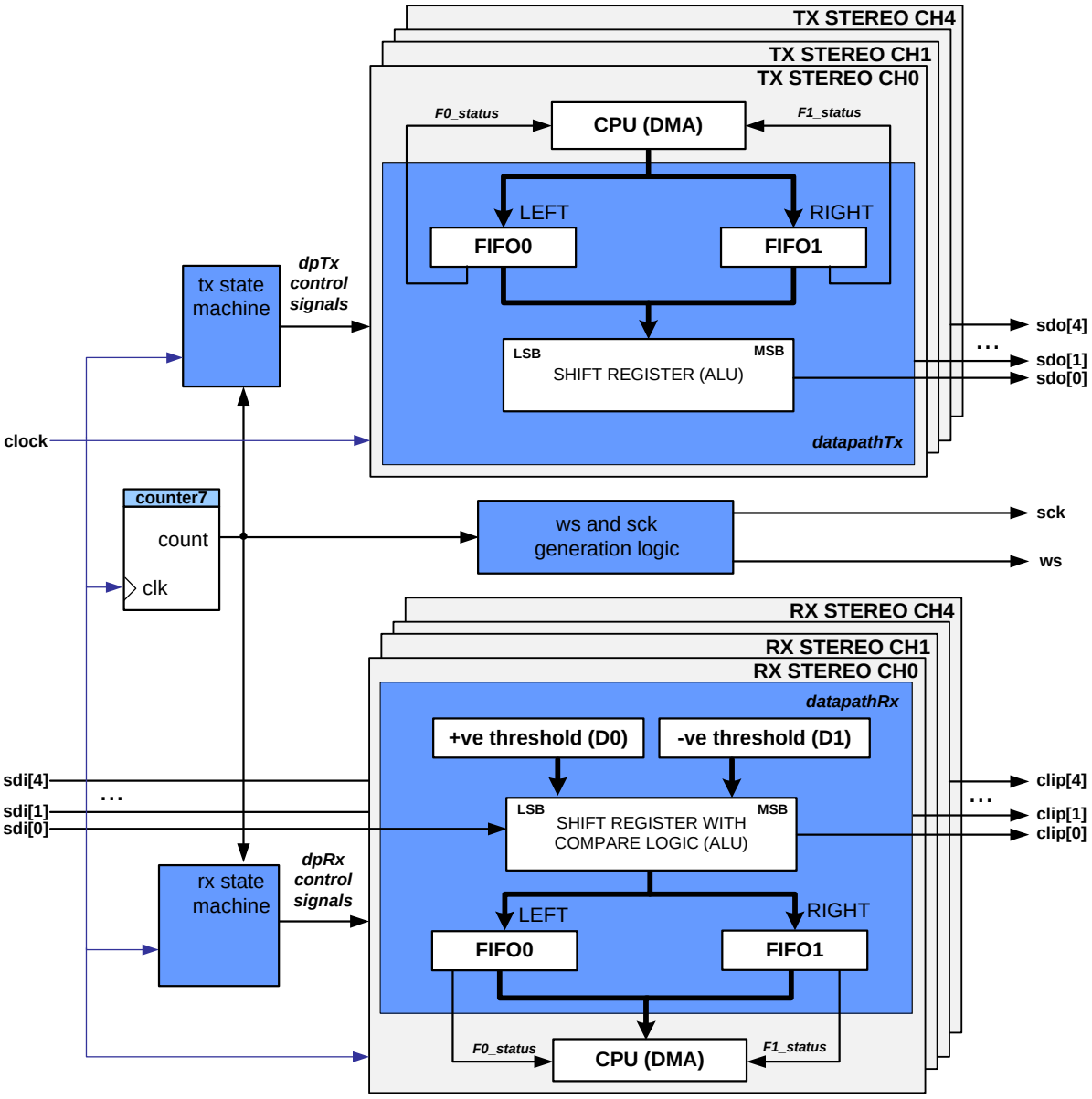
- Master only
- Single-channel and multichannel I^2S support
- 8 to 32 data bits per sample
- 16-, 32-, 48-, or 64-bit word select period
- Data rate up to 96 kHz with 64-bit word select period: 6.144 MHz
- Audio clip detection in I^2S RX mode
- Byte swap audio samples to match USB audio class endianness requirement

17.2 Overview

The I^2S is a serial bus interface standard used for connecting digital audio devices. The specification is from NXP® Semiconductor (I^2S bus specification, February 1986, revised June 5, 1996).

The I^2S block provides a serial bus interface for stereo audio data. This block operates only in master mode and supports both transmission (TX) and reception (RX). Figure 17-1 on page 170 shows a five-channel RX/TX stereo I^2S block. Each I^2S data line can support one stereo channel data (that is, two audio outputs, left and right). If configured for one mono channel data, only one audio output (left or right) of stereo channel 0 is used.

Figure 17-1. I²S Block Diagram



17.2.1 I²S Configurations

The PSoC Creator™ Component Catalog contains three schematic macro implementations of the I²S Component: RX only, TX only, and both RX and TX. For detailed information, refer to the component datasheet.

17.2.2 I²S Block Description

Figure 17-1 depicts a five-channel RX and TX I²S block. This diagram is the basis for further discussion on the I²S block. Table 17-1 on page 171 describes the interface signals shown in Figure 17-1.

Table 17-1. I²S Block Interface Signals

Pin	Direction	Description
sdi	Input	This is the serial data input. The width of this signal depends on the number of RX channels.
clock	Input	The output serial clock is generated from this input. This clock rate must be two times the desired clock rate for the output serial clock.
sdo	Output	This is the serial data output. The width of this signal depends on the number of TX channels.
sck	Output	This is the output serial clock.
ws	Output	This is the word select output, which indicates the channel being transmitted.
rx_interrupt	Output	This is the RX direction interrupt.
tx_interrupt	Output	This is the TX direction interrupt.
rx_dma0	Output	This is the RX direction DMA request for FIFO 0 (left or interleaved). The width of this signal depends on the number of RX channels.
rx_dma1	Output	This is the RX direction DMA request for FIFO 1 (right). The width of this signal depends on the number of RX channels.
tx_dma0	Output	This is the TX direction DMA request for FIFO 0 (left or interleaved). The width of this signal depends on the number of TX channels.
tx_dma1	Output	This is the TX direction DMA request for FIFO 1 (right). The width of this signal depends on the number of TX channels.
clip	Output	This output indicates that a clipping occurs on an input audio signal. It is high when the input signal is beyond the user-configured clipping threshold. The width of this signal depends on the number of RX channels.

The data for TX and RX are independent byte streams. The byte streams are packed with the most significant byte first and the most significant bit in bit 7 of the first word. The number of bytes used for each sample (for the left or right channel) is the minimum number of bytes to hold a sample. Any unused bits are ignored on RX and are sent by 0 on RX.

The data stream for one direction can be a single byte stream or two byte streams. In a single byte stream, the left and right channels are interleaved with a sample for the left channel first, followed by the right channel in a single FIFO (FIFO 0). In two byte streams, the left and right channel byte streams use separate FIFOs: FIFO 0 and FIFO 1. Each FIFO has 4 byte depths.

The RX and TX directions have separate enables. When not enabled, the TX direction transmits all 0 values, and the RX direction ignores all received data. The transition into and out of the enabled state occurs at a word select boundary so that a left/right sample pair is always transmitted or received.

The I²S interface is a continuous interface that requires an uninterrupted stream of data. Refer to the component datasheet for information on this configuration.

The number of bits is not fixed in the I²S standard, but 8 to 32 bits of data for each sample is the practical range. The I²S data format is big endian (MSB first). The I²S Component can be configured to swap the order of the data bytes before capturing the audio samples to RX FIFO(s) or outputting them onto the TX sdo line.

The Component can be configured to detect that the incoming digital audio data is clipping and to provide an indication by triggering a clip signal. In this mode, the MSB of the received audio sample is compared with predefined limits, and a clip output is asserted when the sample amplitude crosses these limits.

17.2.3 Error Handling

Two error conditions are possible.

- Transmit underflow, if the transmit FIFO is empty and a subsequent read occurs
- Receive overflow, if the receive FIFO is full and a subsequent write occurs

If transmit underflow occurs while transmission is enabled, the I²S block forces the constant transmission of zeros. Before transmission begins again, transmission must be disabled, the FIFOs should be cleared, and the data for transmit must be buffered. Then transmission is re-enabled.

If receive overflow occurs while reception is enabled, data capturing is stopped. Before reception begins again, reception must be disabled and the FIFOs should be cleared. Then reception is re-enabled.

18. Pulse-Width Modulator

PRoC™ BLE contains a Pulse-Width Modulator (PWM) block that provides an easy method of generating complex, real-time events accurately. This section explains the PWM block implementation in PRoC BLE.

18.1 Features

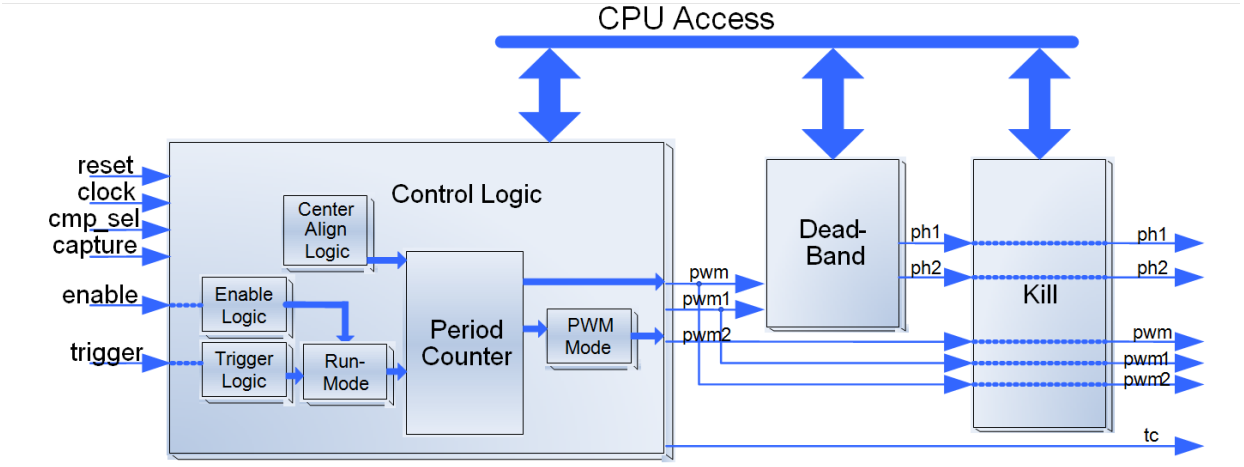
- 8- or 16-bit resolution
- Multiple pulse width output modes
- Configurable trigger
- Configurable capture
- Configurable hardware/software enable
- Configurable dead band
- Multiple configurable kill modes

18.2 Overview

The PWM block provides compare outputs to generate single or continuous timing and control signals in hardware. The most common use of the PWM is to generate periodic waveforms with adjustable duty cycles. The PWM also provides optimized features for power control, motor control, switching regulators, and lighting control. In addition PWM can also be used as a clock divider by driving a clock into the clock input and using the terminal count or a PWM output as the divided clock output.

Figure 18-1 shows the PWM block in PRoC BLE. Refer to the component datasheet for detailed information.

Figure 18-1. PWM Block in PRoC BLE



18.2.1 PWM Pin List

Table 18-1 describes the interface signals shown in Figure 18-1 on page 173.

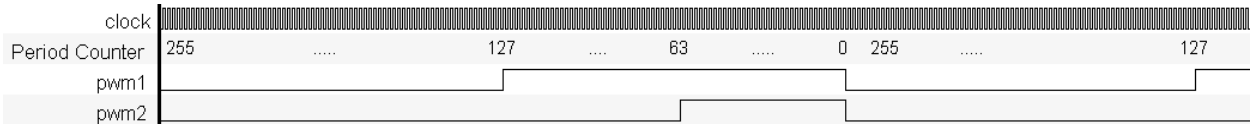
Table 18-1. PWM Block Interface Signals

Signal	Direction	Description
clock	Input	This input defines the signal to count. The counter is incremented or decremented on each rising edge of the clock.
reset	Input	This input resets the period counter and continues normal operation. In reset, the pwm, pwm1, and pwm2 outputs reflect the initial state. The outputs reflect the component reset state after two clock cycles.
enable	Input	This input works in conjunction with software enable and the trigger input (if the trigger input is enabled) to enable the period counter.
kill	Input	If dead band is enabled, then the pwm, pwm1, pwm2, ph1, and ph2 outputs are driven to '0'. If dead band is disabled, then pwm, pwm1, and pwm2 are driven to '0'. The kill signal has no impact on the period counter operation. The tc signal is triggered every period independently from the logic level on the kill input.
cmp_sel	Input	This input selects either the pwm1 or pwm2 output as the final output to the pwm terminal. When the input is '0' (low), the pwm output is pwm1, and when the input is '1' (high), the pwm output is pwm2.
capture	Input	This input forces the period counter value into the read FIFO.
trigger	Input	This input enables PWM operation. The input is registered with the input clock, so the counter starts one clock after the input is asserted.
tc	Output	The terminal count (tc) output is '1' when the period counter is equal to zero. In normal operation, this output is '1' for a single cycle where the counter is reloaded with period value.
interrupt	Output	The interrupt output is the logical OR of the group of possible interrupt sources. This signal goes high while any of the enabled interrupt sources are true. An interrupt can be programmed to be generated under any combination of the following conditions: when the PWM reaches the terminal count or when a compare output goes high.
pwm/pwm1	Output	The pwm or pwm1 output is the first or only pulse-width modulated output.
pwm2	Output	The pwm2 output is the second pulse-width modulated output.
ph1/ph2	Output	The ph1 and ph2 outputs are the dead band phase outputs of the PWM.

18.2.2 PWM Block Description

The default configuration for the PWM is a two-output, 8-bit PWM that creates one output with a compare of less than 127 (with a period of 255) and a second output of less than 63 using a 12-MHz clock. Figure 18-2 shows the inputs and outputs of the PWM in the default configuration. The two PWM outputs are defined independently of each other using two compare values and two compare modes. Each of these two outputs can be left aligned or right aligned.

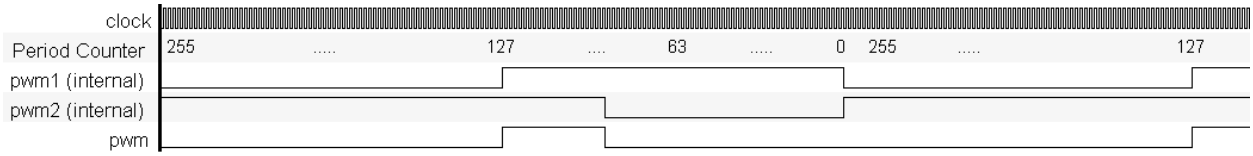
Figure 18-2. PWM Default Outputs



Different PWM configurations are available as follows.

- **Dual-edge PWM:** A dual-edge PWM uses the two compare outputs and two compare modes to generate a single PWM output. The final output is an ANDing of the two signals defined by the two compare values and compare modes, as shown in Figure 18-3.

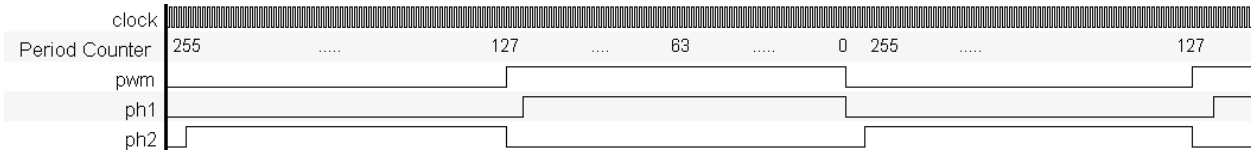
Figure 18-3. Dual-Edge PWM Output



- **Center-aligned PWM:** A center-aligned PWM implements the PWM differently from all other modes. In this mode period counter starts at zero and counts up to the period value, and when period value is reached the counter starts counting back down to zero. In this mode, the period value is half of the period of the final output. A single compare value and compare mode are available for this functionality.
- **Hardware-select PWM:** A hardware-select PWM is implemented as a two-output PWM, where the implementation has two independent compare values and compare modes. A hardware input, cmp_sel, selects which of the two inputs is the final PWM output.
- **Dither-mode PWM:** A dither-mode PWM is implemented as a hardware-select mode PWM with the caveat that the first and compare values have a difference of 1, and both compare modes are identical. Two hardware dither modes are provided to increase PWM flexibility. The first dither mode increases effective resolution by two bits when resources or clock frequency preclude a standard implementation in the PWM counter. The second dither mode uses a digital input to select one of the two PWM outputs on a cycle-by-cycle basis; this mode is typically used to provide a fast transient response in power converts.

Dead band is an add-on option to any of the PWM modes. The dead band outputs work on a single PWM output. In two-output mode, the dead band is implemented only on the pwm1 output. The optional dead band provides complementary outputs with adjustable dead time, where both outputs are low between each transition as shown in Figure 18-4. The complementary outputs and dead time are most often used to drive power devices in half-bridge configurations to avoid shoot-through currents and the resulting damage.

Figure 18-4. Dead Band Outputs



A kill input is also available that immediately disables the dead band outputs when they are enabled. Three kill modes are available to support multiple use scenarios.

- **Asynchronous:** The outputs are disabled while the kill input is active, and the outputs are re-enabled as soon as the kill input goes inactive.
- **Single cycle:** The outputs are disabled while the kill input is active (high), and the outputs are re-enabled at the beginning of the next period.
- **Latched:** The outputs are disabled when the kill input goes high, and the outputs are re-enabled only on the next reset if the kill input is inactive.

The trigger and reset inputs allow the PWM to be synchronized with other internal or external hardware. Three trigger modes are available to support multiple use scenarios.

- **Continuous:** This mode allows the PWM to run forever while enabled. This is the default option.
- **One-shot single:** This mode runs the PWM for a single period on a valid trigger event. A hardware reset pulse rearms the PWM and allows the next trigger event to cause the PWM to run another period.
- **One-shot multi:** This mode runs the PWM for a single period on a valid trigger event. After the period has elapsed, the PWM halts and rearms, waiting for the next trigger event.

19. Bluetooth Low Energy Subsystem (BLESS)

Bluetooth Low Energy (BLE) subsystem (BLESS) implements the Bluetooth LE Link Layer and Physical Layer as specified in Volume 6 of the Bluetooth v4.2 specifications. This subsystem is capable of multiple simultaneous procedures allowed by the Bluetooth v4.2 specification, and maintaining a single connection. This chapter explains the features, implementation, and operational modes of the BLESS.

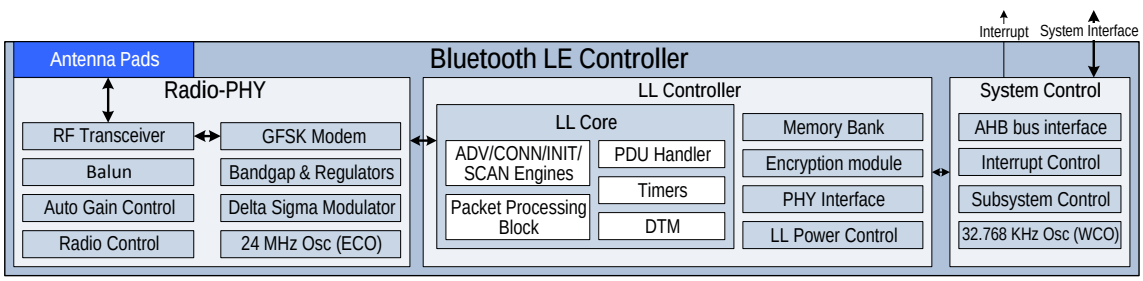
19.1 Features

This subsystem supports the following features:

- Link Layer
  - All mandatory and optional LE Link Layer features of Bluetooth v4.0 specifications and multiple optional LE Link Layer features of Bluetooth v4.2 specification.
  - Both master and slave roles.
  - Support for all packet types and control PDUs.
  - Duplicate and whitelist filtering.
  - Advanced Encryption Standard (AES) 128 encryption/decryption using CCM (Counter with Cipher Block Chaining (CBC) - Message Authentication Code (MAC)) mode.
  - Support for Bluetooth v4.1 features including low-duty cycle advertising, LE ping mechanism for secure pairing, and L2CAP connection oriented channels.
- Complete Bluetooth Low Energy Transceiver.
- Physical Layer
  - GFSK delta-sigma modulator setting range of 1 MHz ~ 5 GHz with symbol rate of 1 Mbps.
  - Auto-calibration for various analog components.
  - 12 MHz, 10-bit DAC and 12 MHz, 9-bit I/Q ADC with 1 MHz IF.
  - Automatic gain control.
  - Integrated balun with single-ended RF pin.
  - High-accuracy 24-MHz and 32.768-kHz oscillators.

19.2 Block Diagram

Figure 19-1. BLE Subsystem Block Diagram



19.3 How it Works

The BLESS includes the following major blocks:

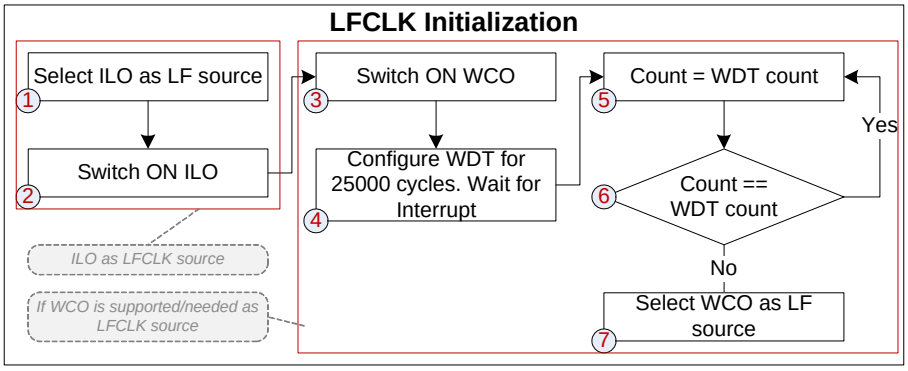
- Radio-PHY block (RF-PHY): This block implements the 2.4-GHz RF transceiver for Bluetooth communication, a 1 Mbps serial data GFSK modem, a delta sigma modulator for channel frequency setting, an auto gain control unit for receiver dynamic range and sensitivity, voltage regulators, calibration and filter control logic, and the 24-MHz external crystal oscillator (ECO).
- Link Layer (LL) Controller: The Link Layer Controller implements the state machines for various Bluetooth LE procedures (advertiser, scanner, initiator, and connection), roles (master and slave), timing critical functions (packet framing/de-framing, CRC generation/checking, encryption/decryption, packet transmission, and protocol time reference keeping), and direct test mode.
- System Control block: This block consists of the AHB bus interface logic, the interrupt control logic, the subsystem controller, which controls the subsystem clocks, reset, and power modes, and a 32.768-kHz watch crystal oscillator (WCO) to generate an accurate low-frequency clock for the system and used by the Link Layer low-power state machine and timekeeping logic.

19.3.1 LFCLK Initialization

The LFCLK has two sources: WCO and internal low-speed oscillator (ILO). This clock is used in the system as a low-frequency clock and in RF initialization. If the LFCLK source is WCO, it is additionally used for the BLESS low-power mode functionality. Any solution that does not support WCO must not initialize WCO, must not support BLESS low-power mode functionality, and must use the ILO as the LFCLK source for RF initialization.

To initialize LFCLK, the following sequence must be followed after power on reset (POR), external reset (XRES), brownout detect (BOD), or other system reset event, during which the ILO and the WCO are OFF. See LFCLK Input Selection on page 78 for more details.

Figure 19-2. LFCLK Initialization



1. Select ILO as the LF clock source by writing the LF_CLK_SEL field in the WDT_CONFIG register.
2. Switch on the ILO in the systems resource by setting the ENABLE field in the CLK_ILO_CONFIG register. The ILO takes about 2 ms to produce a clock. This clock has a high variance and can be used only if the BLESS low-power mode operation is not supported.

The following subsequence is required if the WCO is supported or is needed in the solution.

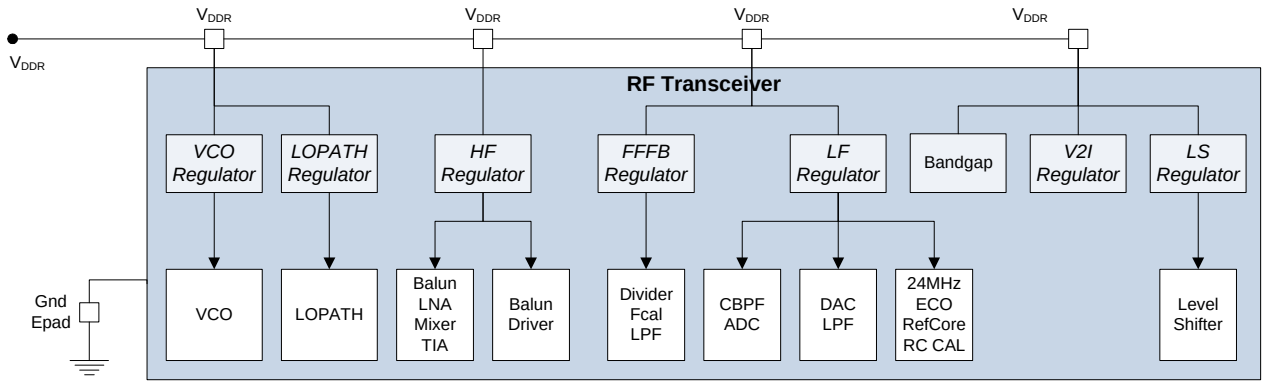
3. Switch on the WCO in the BLESS by setting the ENABLE field in the BLE_BLESS_WCO_CONFIG register. The WCO takes up to 0.5 second to produce a clock.
4. Configure the watchdog timer (WDT), running on the ILO clock, to count for the worst case of 1.7 seconds (due to ILO inaccuracy). See Watchdog Timer chapter on page 95 for the configuration.
5. When the WDT expires, the LFCLK must be switched from ILO to WCO on the next count increment in the WDT. This can be achieved by monitoring the corresponding WDT counter value continuously.
6. When the value changes, switch the clock source by writing the LF_CLK_SEL field in the WDT_CONFIG register.

19.3.2 Radio-PHY Block

19.3.2.1 Power Supply

The Radio-PHY block has regulators that provide power to different sub-blocks inside the block, as shown in Figure 19-3. This regulator network supplies only to the Radio-PHY block. For system-level power supply, refer to the Power Supply and Monitoring chapter on page 83.

Figure 19-3. BLE Subsystem Block Diagram



V_{DDR} provides the input to different regulators with a voltage ranging from 1.9 V to 5.5 V. RF transceiver regulators generate the regulated voltage of 1.8 V from the V_{DDR}.

See Table 19-1 for details about the power mode and regulator status. Details about the BLESS power modes are in Power Modes on page 182.

Table 19-1. RF Transceiver Regulator Status in Different BLE Modes

Mode	LDO_LS	LDO_HF	LDO_VCO	LDO_LOPATH	LDO_FFFB	LDO_V2I	LDO_LF
BLE_DeepSleep	Off	Off	Off	Off	Off	Off	Off
BLE_Sleep	On	Off	Off	Off	Off	On	On
BLE_IDLE	On	Off	Off	Off	Off	On	On
BLE_TX	On	On	On	On	On	On	On
BLE_RX	On	On	On	On	On	On	On

LevelShifter Regulator (LDO_LS)

This regulator supplies power to the level shifter and the bandgap inside the RF transceiver.

HighFrequency Regulator (LDO_HF)

This regulator supplies power to the high-frequency radio circuits such as low noise amplifier (LNA), mixer, and balun circuit.

LDO_FFFB

The Feed-Forward Feedback regulator supplies power to the circuits in the synthesizer block such as dividers and VCO-calibration block.

VCO Regulator (LDO_VCO)

The LDO_VCO regulator supplies power to the VCO section of the RF transceiver.

LO Path Regulator (LDO_LOPATH)

This regulator is used to supply power to the local oscillator path.

V-to-I Regulator (LDO_V2I)

LDO_V2I is used to power the voltage current converter circuit in the RF transceiver.

Low-Frequency Regulator (LDO_LF)

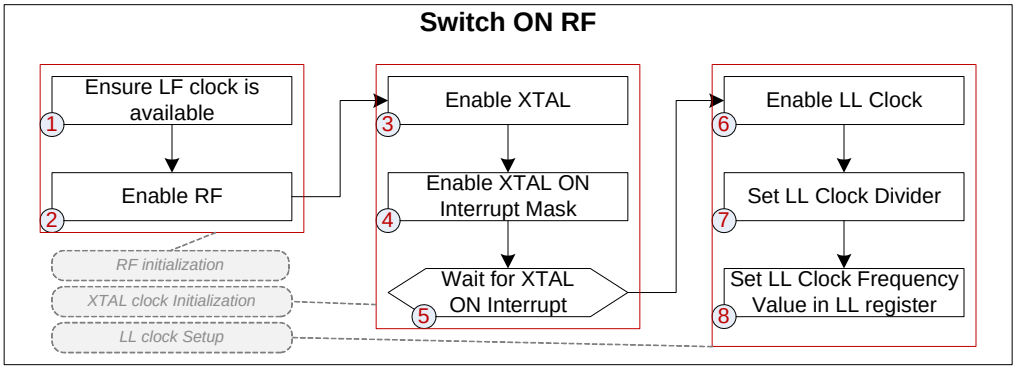
LDO_LF is used to power the low-frequency circuits in the RF transceiver. The low-frequency circuit consists of ADC, complementary band pass filter (CBPF), DAC, ECO, and so on.

19.3.2.2 RF Initialization

Bandgap (BG) and Level Shifter Regulator (LDO_LS)

The signals that control the RF regulators originate from registers located in system power domains. Therefore, until the system power is up, the RF regulators cannot be controlled. For details on the system power domains and modes, see the Power Supply and Monitoring chapter on page 83 and the Power Modes chapter on page 89.

Figure 19-4. RF and LL Clock Initialization



- For the RF to be switched on, an LF clock source must be available (either ILO or WCO). If WCO is not selected or is unavailable, the BLESS low-power modes are not supported.
- For the control signals to flow to the RF regulators, the RF bandgap must be switched on first by setting the RF_ENABLE field in the BLE_BLESS_RF_CONFIG register. This turns on the bandgap generator, followed by the LDO_LS.

ECO Initialization

The sub-sequence is shown in the center in Figure 19-4.

- When the BG and LDO_LS are switched on, the 24-MHz ECO can be enabled by setting the XTAL_ENABLE field in the BLE_BLERD_DBUS register.
- The clock availability is indicated by an interrupt, which can be enabled by setting the XTAL_ON_INTR_MASK field in the BLE_BLESS_LL_DSM_CTRL register.
- The interrupt status is captured in the XTAL_ON_INTR field in the BLE_BLESS_LL_DSM_INTR_STAT register.

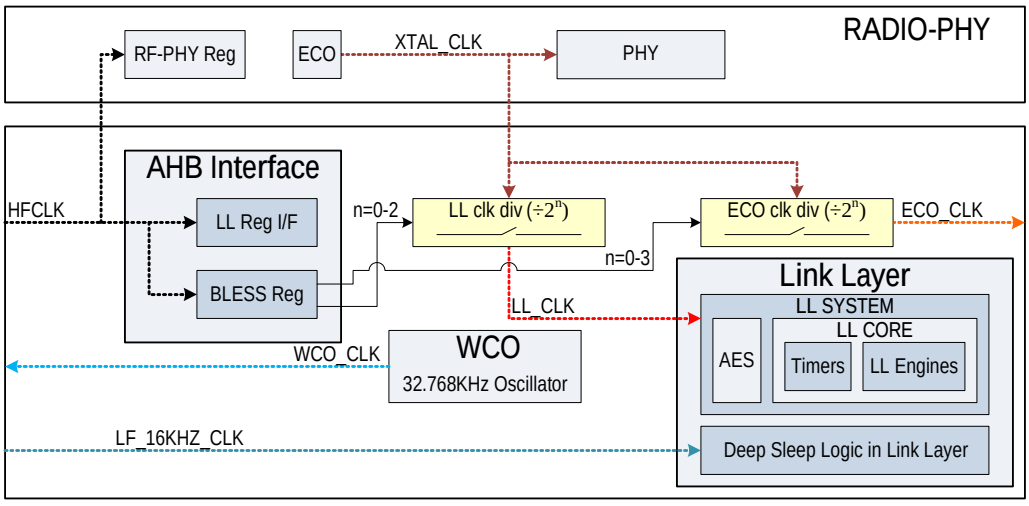
19.3.3 Link Layer Controller

This section provides details about the configuration and high-level functional modes of the Link Layer.

19.3.3.1 Clocking

Figure 19-5 shows the clock architecture of the BLESS.

Figure 19-5. BLESS Clock Network



The BLESS has three asynchronous clock domains, as shown in Figure 19-5.

XTAL_CLK: The XTAL_CLK is the 24-MHz clock from ECO. This clocks the RF-PHY controller, the GFSK modulator and the transmit logic. In the LL Controller, a divided clock (LL_CLK) clocks the Link Layer protocol logic and another divided clock (ECO_CLK) is used as the external clock source in the system.

- LL_CLK: The sub-sequence is shown on the right in Figure 19-4.
  - The primary control of the LL_CLK clock within the Link Layer is achieved by the CLK_EN bit in the BLE_BLESS_LL_CLK_EN register. When this is enabled, the clock can further be controlled by Link

Layer firmware commands and control bits, which can put the LL into Sleep mode or Deep Sleep mode and wake up as well.

- The LL_CLK is generated from the XTAL_CLK by a programmable divider, which divides by 1, 2, and 4. The LL_CLK can, hence, be 24 MHz, 12 MHz, or 6 MHz. The division value can be set in the LLCLK_DIV field of the BLE_BLESS_XTAL_CLK_DIV_CONFIG register.
- The corresponding frequency setting must be set into the BB_CLK_FREQ_MINUS_1 field in the BLE_BLELL_POC_REG_TIM_CTRL register. The mapping is shown in Table 19-2.

Table 19-2. LL_CLK Frequency and Register Setting

LL_CLK Frequency	LLCLK_DIV	BB_CLK_FREQ_MINUS_1
24 MHz	2'b00	4'd23
12 MHz	2'b01	4'd11
06 MHz	2'b10	4'd5

- ECO_CLK: The ECO_CLK is generated from the XTAL_CLK by a programmable divider, which divides by 1, 2, 4, and 8. The ECO_CLK can, hence, be 24 MHz, 12 MHz, 6 MHz, or 3 MHz. The division value can be set in the SYSCLK_DIV field of the BLE_BLESS_XTAL_CLK_DIV_CONFIG register. See External Crystal Oscillator (ECO) on page 76 for system level usage of this clock.

HFCLK: This clock is used to clock the RF-PHY and subsystem registers and the LL AHB interface. It can range from 3 to 48 MHz. See the Clocking System chapter on page 73 for system details on this clock.

LF_16KHZ_CLK: The LF_16KHZ_CLK is a either a 16.384-kHz accurate clock or a 16-kHz clock derived from the system low-frequency clock LFCLK. See Watch Crystal Oscillator (WCO) on page 77 and Clock Distribution on page 77 for multiplexing details.

- 16.384 kHz: The LFCLK source is WCO. The clock, in this mode, is used in the Link Layer to maintain a high accuracy BLE protocol timekeeping during Link Layer low-power modes. The clock can be gated by a firmware control bit, DISABLE_LF_CLK field, in the BLE_BLESS_LF_CLK_CTRL register during BLESS Deep Sleep mode to enter into extended deep sleep mode, where the BLE protocol timekeeping in low-power mode is not required.
- 16 kHz: The LFCLK source is ILO. This clock is not accurate and the BLE protocol timekeeping during BLESS low-power modes is not supported. This clock can be gated by a firmware control bit, DISABLE_LF_CLK field, in the BLE_BLESS_LF_CLK_CTRL register during BLESS Deep Sleep mode to enter into extended deep sleep mode.

19.3.3.2 Firmware Reset

The Link Layer can be reset using a Link Layer firmware soft reset request. This resets all the registers and the state machines to their default states. Note that the reset does not modify the FIFO content. The FIFO content becomes invalid and the FIFOs need to be reinitialized.

19.3.3.3 BLE Functional Modes and Configuration

This section details the various modes of BLE functions. The operation of the Link Layer can be defined by means of a state machine with the following main modes.

- Standby
- Advertising
- Scanning (Active or Passive)
- Initiating
- Connection (Master or Slave)
- Direct Test Mode (DTM)
- Encryption/Decryption

Standby State

This state is the default functional state. The Link Layer in this state does not transmit or receive any packets. It is possible to move from this state to advertising or scanning or initiating states based on the relevant functional requirement. This state can be entered from any other state.

Advertising State

The advertising state allows the Link Layer to transmit advertising channel packets; it also has the ability to respond to peer devices' responses triggered because of these advertising channel packets. A device in this state is termed as an advertiser and can be entered from the Standby state. A device moves to this state to either be discoverable or connectable. This is also required if the device wants to broadcast data to other devices in the area.

Scanning State

The Link Layer in this state facilitates the device to listen for advertising channel packets from the peer devices that are advertising. The intent of listening classifies this state into two:

- Active Scanning
- Passive Scanning

Passive scanning is used to simply listen for the presence of other devices. Active scanning permits the device to respond with scan requests to advertising devices to obtain further data about the peer device and potentially move further to engage in connection.

The device in this state is known as scanner and can be entered from the Standby state.

Initiating State

The Link Layer in the Initiating state will listen for advertising

channel packets from the specific peer device and respond to initiate a connection with the peer device. A device in this state is known as an initiator and this state can be entered from the Standby state.

Connection State

The final procedural state in the Link Layer protocol is the Connection state. This is entered from either the Advertising or the Initiating state; the device in this state is termed as being in connection.

This state has two roles:

- Master role
- Slave role

If this state is entered from Advertising state, then the device is qualified as a slave; if it is entered from Initiating state, it is known as a master.

Direct-Test Mode

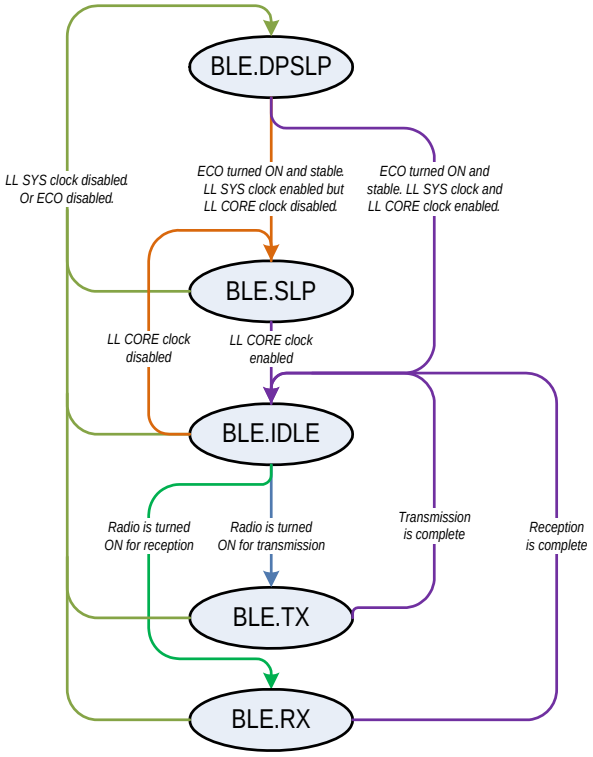
This mode provides ability to test the Physical Layer (radio digital/RF). It allows a tester to command a controller's Physical Layer to either transmit or receive a sequence of pre-defined test packets. The tester can then analyze the packets received, to determine if the Physical Layer is working according to the specification. The tester can also measure various RF parameters from the received packets to determine the compliance of the Physical Layer to RF Specifications.

Encryption/Decryption

This mode provides ability to encrypt a plain text data or decrypt an encrypted data. Bluetooth LE uses AES-128 (Advanced Encryption Standard) encryption/decryption using CCM (Counter with Cipher Block Chaining (CBC)-Message Authentication Code (MAC)).

19.3.4 Power Modes

Figure 19-6. BLE Subsystem Power Modes



BLESS supports five functional power modes - Deep Sleep mode (DSM), Sleep mode (SM), Idle, Transmit, and Receive. A series of clock gating logic allows achieving the power modes. A primary clock gate (LL DSM clock gate) logic provides the LL_SYS clock, which feeds all the submodules, including a secondary clock gate (LL SM clock gate). The secondary clock gate provides LL_CORE clock to the CORE module and all its submodules. If the LFCLK source is WCO, the lpo_16khz_clk feeding the Power Control module maintains the timing reference in Deep Sleep mode. Table 19-3 lists the BLE-System power mode coexistence.

Table 19-3. BLE-System Power Mode Coexistence Table

BLESS Power Mode	Description	System Power Mode			Application Scenario
		DPSLP	SLP	ACT	
BLE.DSM	Radio is off. Logic is powered. WCO may be enabled to keep LL timing. ECO may be enabled for fast wake up (but is gated).	✔	✔	✔	BLE enters this state between long event intervals for maximum power saving. Time keeping is done in a low-frequency clock. The system can be in Active, Sleep, or Deep Sleep mode based on other peripheral activity.
BLE.SM	Radio is off. Logic is powered. WCO may be enabled. ECO is enabled but the clock to LL Core and Digital modem logic is gated.		✔	✔	BLE enters this state between short event intervals for saving power. BLE also enters this state to perform AES-CCM Encryption/Decryption for non-BLE applications. The system can be in Active or Sleep mode based on other peripheral activity.
BLE.IDLE	Radio is off. Logic is active with ECO clocking LL and RD logic. No transmission/reception ongoing.		✔	✔	BLE enters this state before and after packet transceiving. All FIFO and register configuration is done in this state. The system can be in Active or Sleep mode based on BLE/peripheral activity.
BLE.TX	LL and RD logic are active. Radio is turned on and transmitting.		✔	✔	BLE enters this state when the radio is transmitting. The system can be in Active or Sleep mode based on peripheral activity.
BLE.RX	LL and RD logic are active. Radio is turned on and receiving.		✔	✔	BLE enters this state when the radio is receiving The system can be in Active or Sleep mode based on peripheral activity.

19.3.4.1 Deep Sleep Mode

The Deep Sleep mode is the lowest power functional mode supported by the BLESS. In this mode, the radio is off. This mode is entered for maximum power saving during advertising, scanning, or connection interval after packet transmission and reception is complete. The LL_SYS clock is gated or the ECO is turned off for power saving. For BLE applications that require Link Layer low-power mode timing reference/wake-up logic, the LFCLK must be set to WCO and must be on. For extended deep sleep mode, the LFCLK can be set to ILO with LF_16KHZ_CLK gated and the WCO switched off. The system can be in Active, Sleep, or Deep Sleep mode based on BLE or peripheral activity.

19.3.4.2 Sleep Mode

In the Sleep mode, the radio is off. The block maintains all the configurations. This mode is entered during short intervals between events. This mode can also be entered to perform non-BLE AES-CCM encryption/decryption activity. The CPU controls entry and exit to this state. The LL CORE clock is gated for power saving. The system can be in Active or Sleep mode based on BLE or peripheral activity.

19.3.4.3 Idle Mode

The Idle mode is the pre and post state for the Transmit and Receive states. In this state, the radio is turned off, but the Link Layer clock is enabled for Link Layer logic so that the CPU can start the protocol state machines, configure, and read FIFOs and registers. The system can be in Active or Sleep mode based on BLE or peripheral activity.

19.3.4.4 Transmit Mode

This mode is entered when the radio starts data transmission. The modulator and radio transmitter is on. The RF-PHY gets 1 Mbps serial data from the Link Layer and transmits 2.4 GFSK modulated data to the antenna port. BLE enters the Transmit mode from the Idle mode. The system can be in Active or Sleep mode based on BLE or peripheral activity.

19.3.4.5 Receive Mode

This mode is entered when the radio starts data reception. The demodulator and radio receiver is on. The RF-PHY translates the 1 Mbps data received from RF analog and forwards it after demodulation to the Link Layer controller. The system can be in Active or Sleep mode based on BLE or peripheral activity.

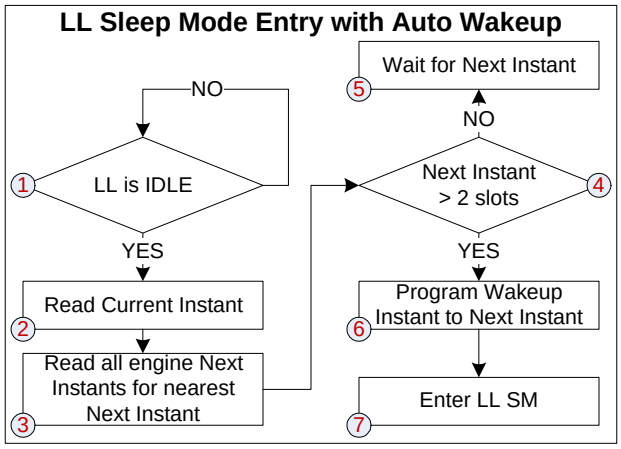
19.3.5 Mode Transitions

This section discusses various mode transition sequences involving BLESS.

19.3.5.1 LL Sleep Mode Entry with Auto Wakeup

This mode can be entered when the LL engines are idle and the next procedure event (Next Instant) is at least two slots away (1.25 ms). Figure 19-7 shows the flow.

Figure 19-7. LL Sleep Mode Entry with Auto Wakeup Flow Diagram

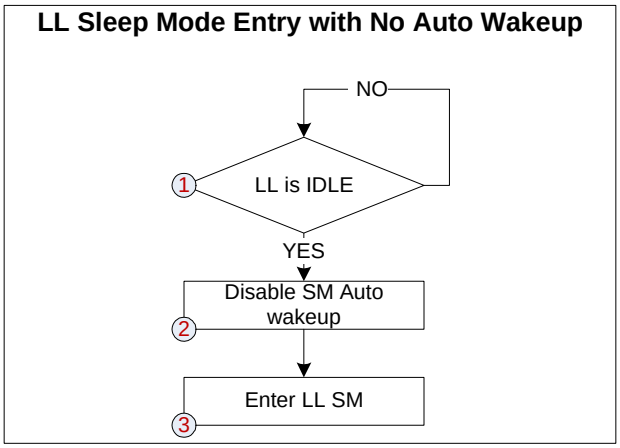


1. Check if LL is idle by reading the BLE_BLELL_CLOCK_CONFIG.LLH_IDLE register field.
2. If LL is idle, then read the current instant from the BLE_BLELL_TIM_CONTROL_L register.
3. Read the next instant for all the engines from the corresponding registers BLE_BLELL_ADV_NEXT_INSTANT, BLE_BLELL_INIT_NEXT_INSTANT, BLE_BLELL_SCAN_NEXT_INSTANT, and BLE_BLELL_NEXT_CE_INSTANT and choose the closest instant.
4. Check if the next instant is less than two slots away.
5. If the next instant is less than two slots away, then wait for the next instant.
6. If the next instant is more than two slots away, then program the value to wake up the instant in the BLE_BLELL_WAKEUP_CONTROL register.
7. Enter Sleep mode by gating the LL SM clock gater to gate CLK_CORE.

19.3.5.2 LL Sleep Mode Entry with No Auto Wakeup

This mode can be entered when no BLE activity is foreseen and LL is not ready to enter Deep Sleep mode. This mode can also be entered when the AES-CCM needs to be used for non-Bluetooth LE application. The flow is shown in Figure 19-8.

Figure 19-8. LL Sleep Mode Entry with No Auto Wakeup



1. Check if LL is idle by reading the BLE_BLELL_CLOCK_CONFIG.LLH_IDLE register field.
2. Disable SM auto wakeup by writing '0' to the BLE_BLELL_CLOCK_CONFIG.SM_AUTO_WKUP_EN field.
3. Enter Sleep mode by gating the LL SM clock gater to gate CLK_CORE.

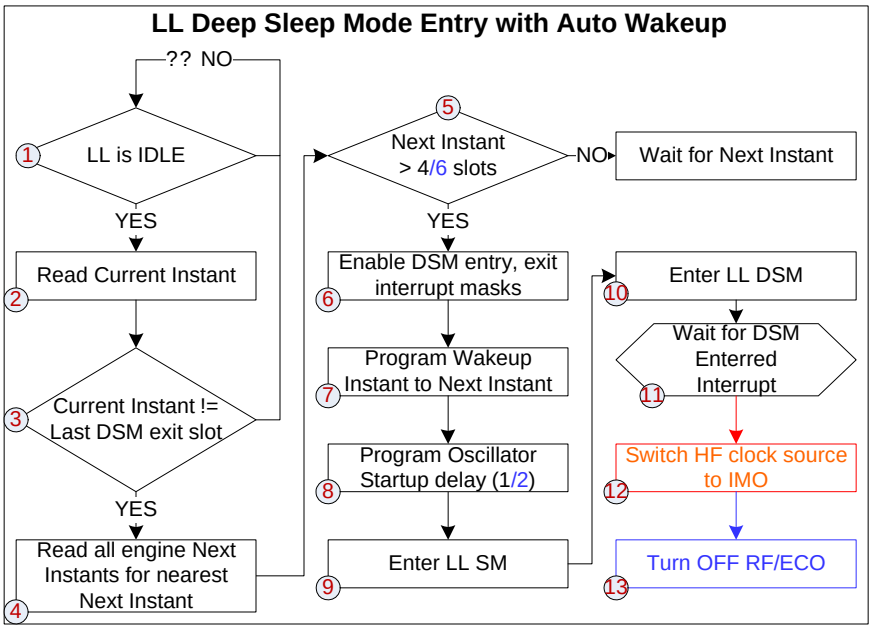
19.3.5.3 Manual Exit from Sleep Mode

To manually exit from the LL sleep mode with no auto wakeup, enable the LL SM clock gater to allow CLK_CORE.

19.3.5.4 LL Deep Sleep Mode Entry with Auto Wakeup

This state can be entered only when the LL engines are idle and the next instant is at least four slots (2.5 ms) away, if the ECO is scheduled to be kept on or at least six slots (3.75 ms) away, or if the ECO is scheduled to be switched off. The flow is shown in Figure 19-9; the step indicated in red is executed if the system is scheduled to enter Deep Sleep mode or if the ECO is scheduled to be switched off. The step in blue is executed if the ECO is scheduled to be switched off.

Figure 19-9. LL Deep Sleep Mode Entry with Auto Wakeup Flow Diagram



1. Check if LL is idle by reading the BLE_BLELL_CLOCK_CONFIG.LLH_IDLE register field.
2. If LL is idle, then read the current instant from the BLE_BLELL_TIM_CONTROL_L register.
3. Check if the current instant slot is the same as the last Deep Sleep Exit slot. If the slots are the same, then DSM entry is not allowed as this will affect the slot count value in DSM mode. Go to step 1.
4. If the slots are different, then read the next instants for all the engines from the corresponding registers BLE_BLELL_ADV_NEXT_INSTANT, BLE_BLELL_INIT_NEXT_INSTANT, BLE_BLELL_SCAN_NEXT_INSTANT, and BLE_BLELL_NEXT_CE_INSTANT and choose the closest instant.
5. Check if the next instant is less than four slots away if RF/ECO are intended to be kept on or six slots away if

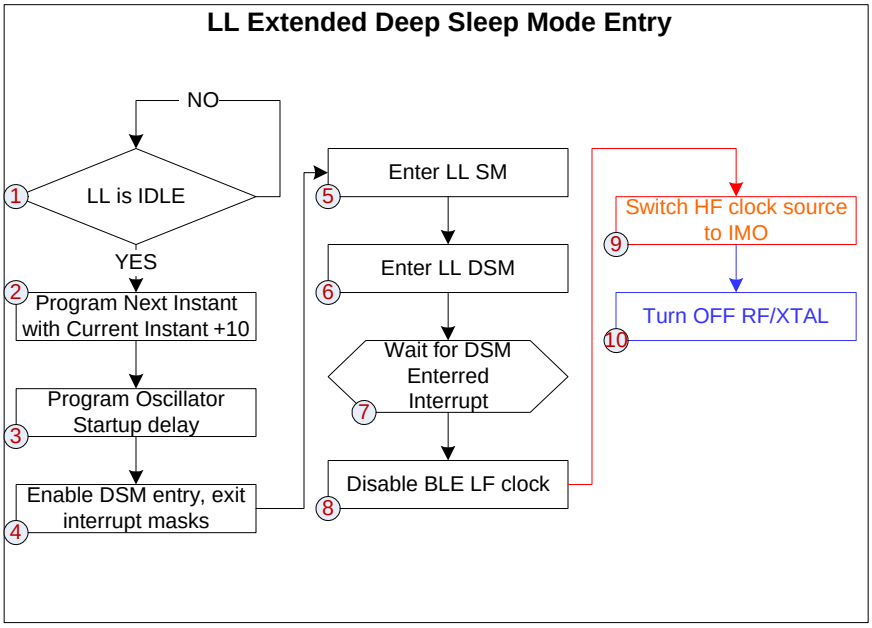
RF/ECO are intended to be switched off. If the next instant is less than four or six slots away, then wait for the next instant.

6. Enable DSM entry and exit interrupt masks by setting the DSM_ENTERED_INTR_MASK and DSM_EXITED_INTR_MASK fields in the BLE_BLESS_LL_DSM_CTRL register.
7. If the next instant is more than four or six slots away, then program the value to wake up the instant in the BLE_BLELL_WAKEUP_CONTROL register.
8. Program the oscillator startup delay in the BLE_BLELL_WAKEUP_CONFIG.OSC_STARTUP_DELAY register field with one slot if RF/ECO are intended to be kept on or two slots away if RF/ECO are intended to be switched off.
9. Enter Sleep mode by gating the LL SM clock gater to gate CLK_CORE.
10. Enter Deep Sleep mode by writing the command opcode ENTER_DSM to BLE_BLELL_COMMAND_REGISTER.
11. Wait for the DSM entered interrupt to be asserted to ensure successful DSM entry transition. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.DSM_ENTERED_INTR register field.
12. The step indicated in red is executed if the HF clock is set to ECO. The HF clock source is now switched to IMO.
13. The step indicated in blue is executed if the RF/ECO are intended to be switched off.

19.3.5.5 LL Extended Deep Sleep Mode Entry

This state can be entered when no BLE functionality is foreseen for long duration (minutes or more) and BLE timekeeping is not required. All Link Layer data and timing information become invalid. The flow is shown in Figure 19-10. The step indicated in red is executed if the system is scheduled to enter Deep Sleep mode or if the ECO is scheduled to be switched off. The step in blue is executed if the ECO is scheduled to be switched off.

Figure 19-10. LL Extended Deep Sleep Mode Entry Flow Diagram



1. Check if LL is idle by reading the BLE_BLELL_CLOCK_CONFIG.LLH_IDLE register field.
2. If LL is idle, then read the current instant from the BLE_BLELL_TIM_CONTROL_L register and program the value current instant + 10 to wake up the instant in the BLE_BLELL_WAKEUP_CONTROL register.
3. Program the oscillator startup delay in the BLE_BLELL_WAKEUP_CONFIG.OSC_STARTUP_DELAY register field with two slots.
4. Enable DSM entry and exit interrupt masks by setting the DSM_ENTERED_INTR_MASK and DSM_EXITED_INTR_MASK fields in the BLE_BLESS_LL_DSM_CTRL register.
5. Enter Sleep mode by gating the LL SM clock gater to gate CLK_CORE.
6. Enter Deep Sleep mode by writing the command opcode ENTER_DSM to the BLE_BLELL_COMMAND_REGISTER.
7. Wait for the DSM entered interrupt to be asserted to ensure successful DSM entry transition. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.DSM_ENTERED_INTR register field.
8. Disable BLE LF clock, LF_16KHZ_CLK, by setting the BLE_BLESS_LF_CLK_CTRL.DISABLE_LF_CLK regis-

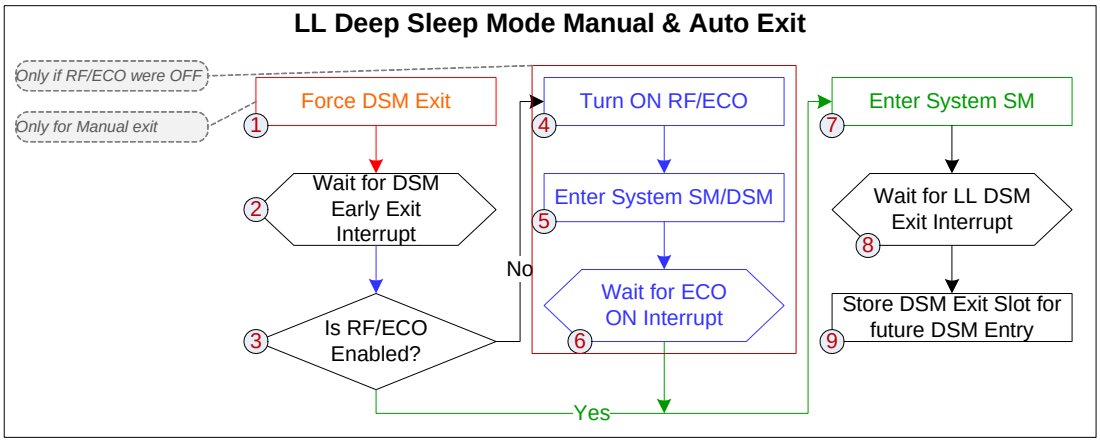
ter field. This will prevent the running of the low-power state machine.

9. The step indicated in red is executed if the HF clock is set to ECO. The HF clock source is now switched to IMO.
10. The step in blue is executed if the RF/ECO are intended to be switched off.

19.3.5.6 LL Deep Sleep Mode Manual and Auto Exit

To manually wake from LL deep sleep, the DSM exit can be asserted, which starts the wakeup sequence. In case of auto wakeup, the internal timeout will start the wakeup sequence. The flow is shown in Figure 19-11; the steps indicated in blue are executed if the ECO is off. The step in green is executed if the CPU has no other activity and the wakeup sequence is long.

Figure 19-11. LL Deep Sleep Mode Manual and Auto Exit Flow Diagram



1. If a manual exit is needed, then set the BLE_BLESS_LL_DSM_CTRL.DSM_EXIT register field to force an exit.
2. For manual and auto exit, wait for the DSM early exit interrupt. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.DSM_EXITED_INTR register field.
3. Check if RF/ECO is already enabled.

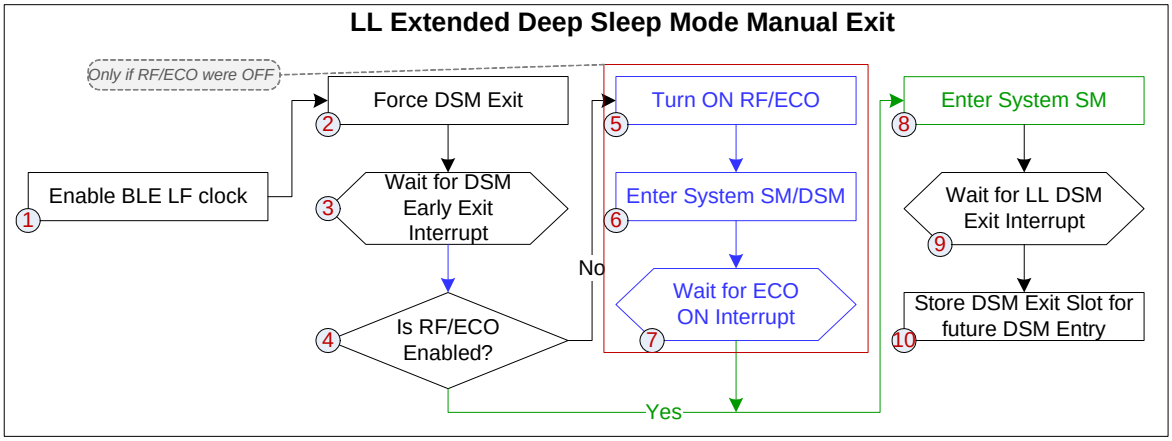
The following subsequence (steps 4 to 6) is required if RF/ECO was off.

4. Enable RF/ECO.
5. If the CPU has no other activity, then enter system Sleep mode or Deep Sleep mode after enabling the ECO on interrupt by setting the BLE_BLESS_LL_DSM_INTR.XTAL_ON_INTR_MASK register field (if not enabled previously).
6. Wait for the ECO ON interrupt to be asserted; if the system has entered the low-power mode in step 5, wake the system up. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.XTAL_ON_INTR register field.
7. After the ECO ON interrupt or if the ECO was on when the ‘n’ check was done in step 3, if the CPU has no other activity, then enter system Sleep mode.
8. Wait for the LL DSM exit interrupt to be asserted; if the system had entered sleep mode in step 7, then wake the system up. The status is captured in BLE_BLELL_EVENT_INTR.DSM_INTR.
9. On exit store the current slot value as the DSM exit slot, which will be used for future DSM entry calls.

19.3.5.7 LL Extended Deep Sleep Mode Manual Exit

To manually wake from LL extended deep sleep, the LF clock must be enabled and the DSM exit must be asserted, which starts the wakeup sequence. The flow is shown in Figure 19-12; the steps in blue are executed if the RF/ECO is off. The step in green is executed if the CPU has no other activity and the wakeup sequence is long.

Figure 19-12. LL Extended Deep Sleep Mode Manual Exit Flow Diagram



1. Enable BLE LF clock, LF_16KHZ_CLK, by clearing the BLE_BLESS_LF_CLK_CTRL.DISABLE_LF_CLK register field. This will start the low-power state machine.
2. Set the BLE_BLESS_LL_DSM_CTRL.DSM_EXIT register field to force an exit from DSM mode.
3. For manual and auto exit, wait for the DSM early exit interrupt. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.DSM_EXITED_INTR register field.
4. Check if RF/ECO is already enabled.

The following subsequence (steps 5 to 7) is required if RF/ECO was off.

5. Enable RF/ECO.
6. If the CPU has no other activity, then enter system Sleep mode or Deep Sleep mode after enabling the ECO on the interrupt by setting the BLE_BLESS_LL_DSM_INTR.XTAL_ON_INTR_MASK register field (if not enabled previously).
7. Wait for the ECO ON interrupt to be asserted; if the system has entered the low-power mode in step 5, wake the system up. The status is captured in the BLE_BLESS_LL_DSM_INTR_STAT.XTAL_ON_INTR register field.
8. After the ECO ON interrupt or if the ECO was on when the ‘n’ check was done in step 3, if the CPU has no other activity, then enter system Sleep mode.
9. Wait for the LL DSM exit interrupt to be asserted; if the system has entered the low-power mode in step 7, wake the system up. The status is captured in the BLE_BLELL_EVENT_INTR.DSM_INTR.
10. On exit store the current slot value as the DSM exit slot which will be used for future DSM entry calls.

19.3.6 Bluetooth LE 4.2 Feature – Data Length Extension

Data length extension is a BLE 4.2 feature that increases the maximum supported length of the LL data PDU from 27 to 251 octets. This feature is enabled in the sub-system by setting BLE_BLELL_LL_CONTROL_DLE. With this feature enabled, the transmit and receive packet lengths during connection are increased to 251 bytes. The encryption module also supports 251-byte packet encryption and decryption.

19.3.7 Bluetooth LE 4.2 Feature – Privacy 1.2

Privacy 1.2 is a BLE 4.2 feature in which private address generation and resolution is implemented in the BLE Link Layer. The functionality is partitioned between the BLESS hardware and Link Layer firmware.

This feature is enabled in the sub-system by setting BLE_BLELL_LL_CONTROL_PRIV_1_2.

19.3.7.1 Resolving List

The Link Layer maintains a set of records called resolving list for local and peer IRK value pairs. The resolving list contains the following fields.

- Peer Identity Address: A peer device's Public or Static address.
- Local Identity Resolution Key (IRK): A key, shared with the associated peer device and used to generate a Self Resolvable Private address (RPA).
- Peer IRK: A key, shared by the associated peer device and used to resolve a peer RPA.

The resolving list is implemented between firmware and hardware, with the hardware resolution list implemented as shown in Table 19-4. The firmware maintains the resolving list as exposed and used by the host layer but works with the hardware resolving list for list operations and management.

Table 19-4. Resolution List

Valid Entry	Peer Privacy level	Self Privacy Level	Peer Whitelist status	Peer ID addr type	Peer RPA valid	Rcvd self RPA valid	TX self RPA valid	INIT specific device	Self TX Addr type	Peer idnty addr 48 bits	Peer RPA 48 bits	Rcvd Self RPA 48 bits	TX self RPA 48 bits

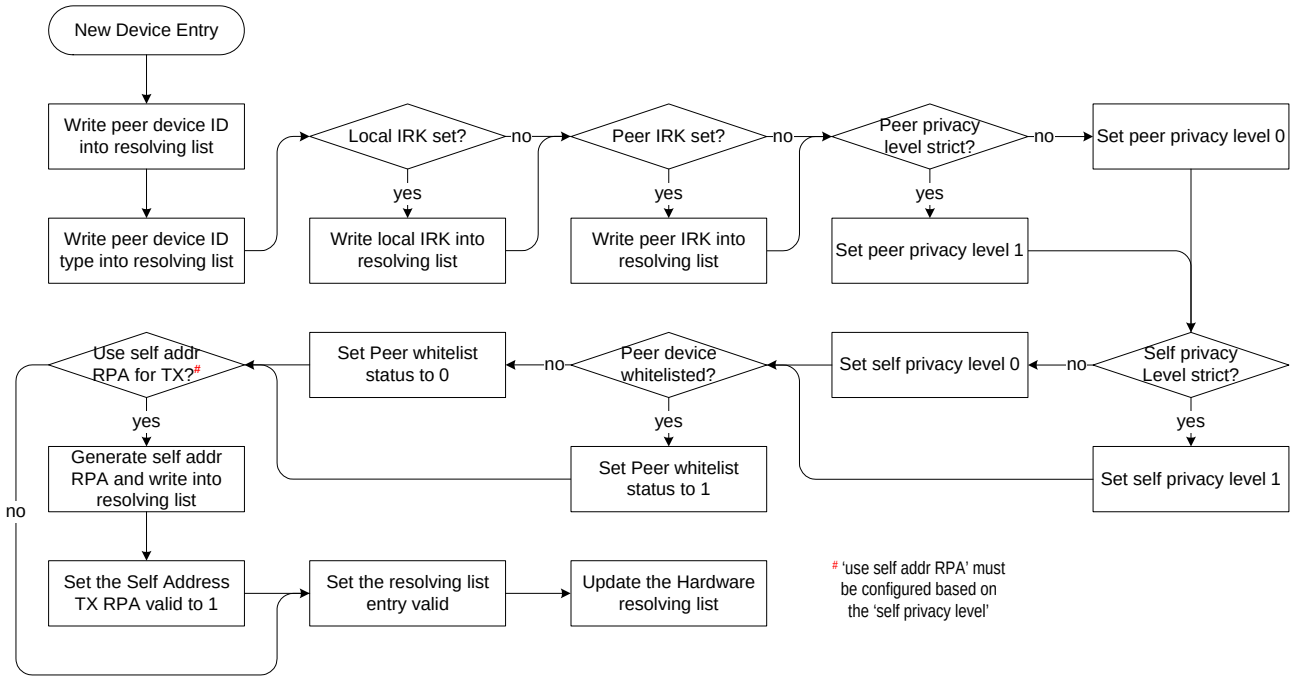
19.3.7.2 Resolving List Functions

Following are the generic procedures related to privacy that should be managed by the Link Layer firmware.

Add Device to Resolving List

To add a device to the resolving list, the firmware must add the DevA to the hardware resolving list with the appropriate flags set, and the device identity address and IRKs populated. The firmware procedure guideline is given in Figure 19-13.

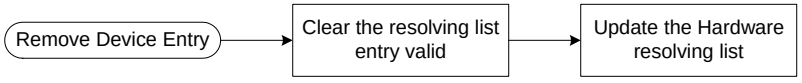
Figure 19-13. Add Device to Resolving List



Remove Device from Resolving List

To remove a device from the resolving list, the firmware must set the BLE_BLELL_RSLV_LIST_ENABLE_VALID_ENTRY bit to '0' for the device address in the hardware resolving list. The content of the entire row then becomes invalid and the row can be reused to store an entry for another device. The firmware procedure guideline is given in Figure 19-14.

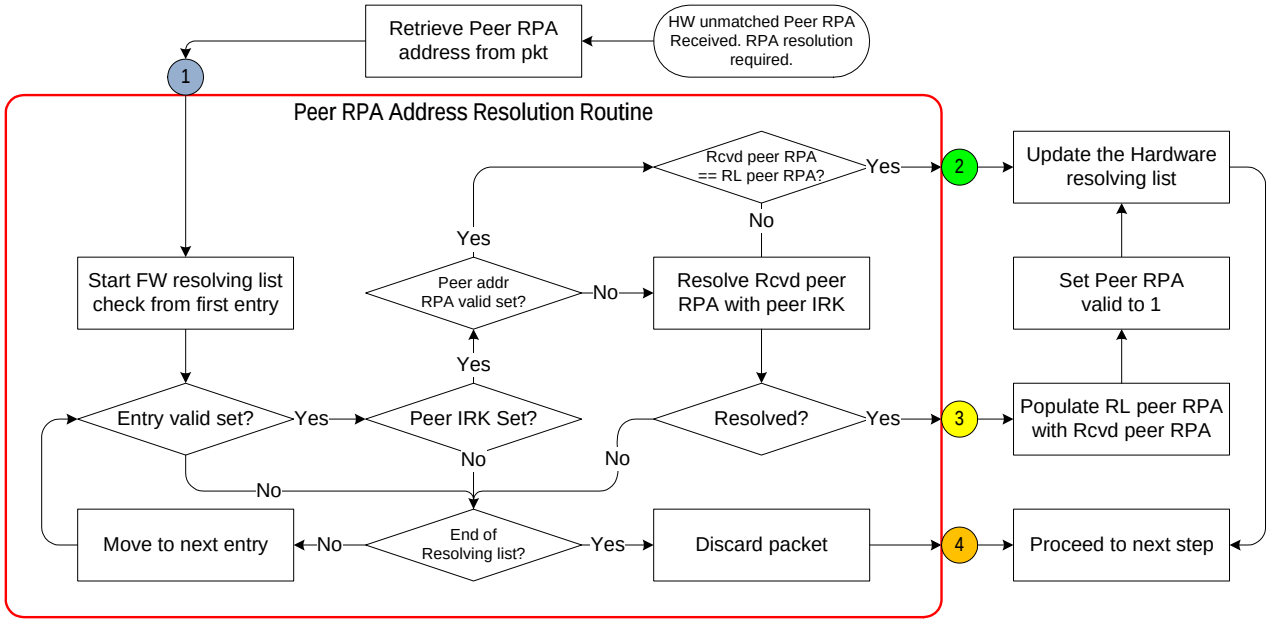
Figure 19-14. Remove Device from Resolving List



Handling Unresolved Peer RPA

When a packet is received with the peer address as RPA, the RPA is compared against the PEER_RPA field of all valid rows (with BLE_BLELL_RSLV_LIST_ENABLE_VALID_ENTRY and BLE_BLELL_RSLV_LIST_ENABLE_PEER_ADDR_RPA_VAL bits set). If no valid entry is present, it means the address is still unresolved. The hardware interrupts the firmware and provides the received peer RPA for resolution. The firmware procedure guideline for peer address resolution is given in Figure 19-15.

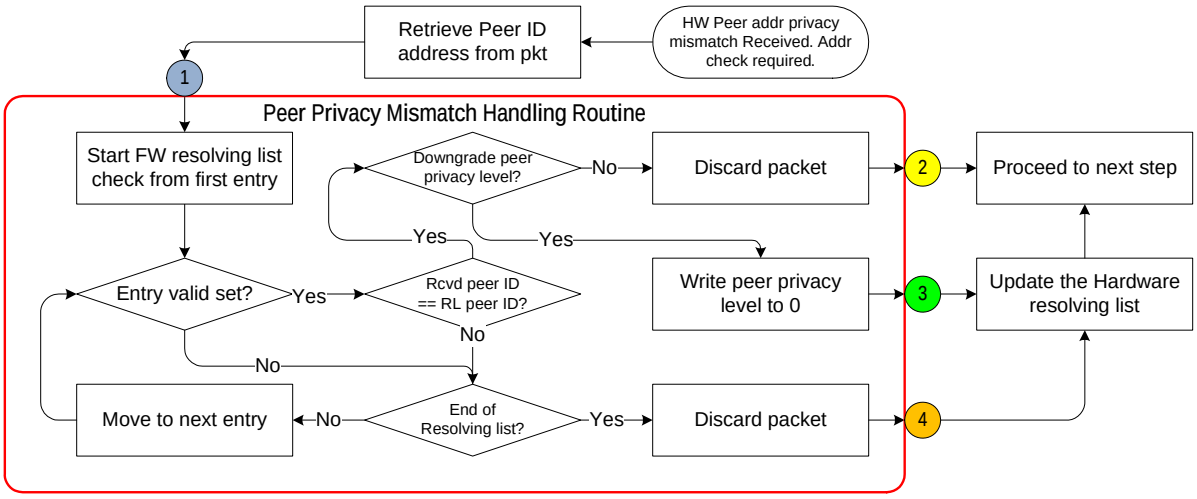
Figure 19-15. Peer RPA Address Resolution



19.3.7.3 Handling Peer Devices that Do Not Use RPA

When a packet is received with the peer address as Identity, but the expected type is an RPA (peer privacy level is set to 1), a peer address privacy mismatch is reported. The firmware must check the resolving list to match the received peer ID address and either downgrade the privacy level to allow accepting a packet with peer address as Identity, or reject the packet as a privacy violation. This option is retained to allow any future changes to the specifications in handling such devices.

Figure 19-16. Peer Privacy Mismatch Handler

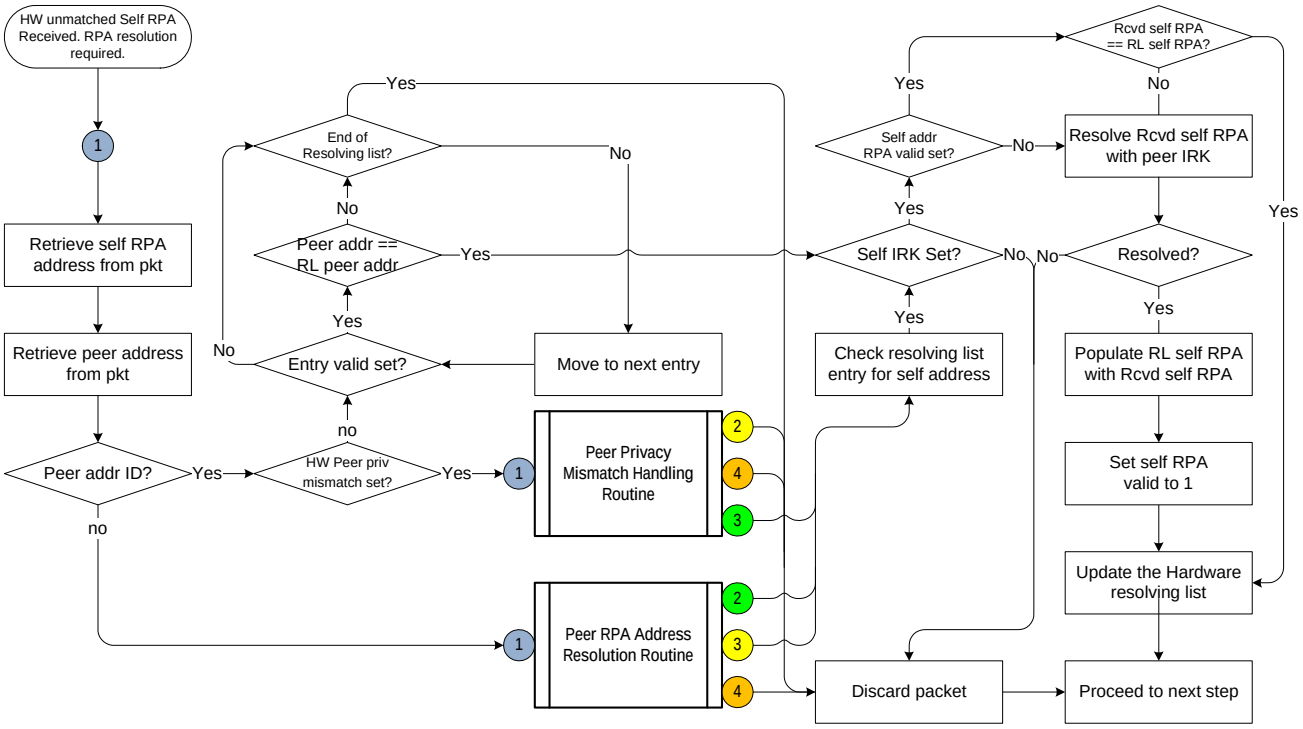


19.3.7.4 Handling Unresolved Self RPA

When a packet is received with the self address as an RPA, and if an entry is matched for the peer address (ID or RPA), the received self RPA is compared against the self RPA field of the matched row. If either the self RPA does not match or both peer address and self RPA do not match, then a self address unmatched interrupt is raised (along with the appropriate peer address interrupt).

The firmware has to first match the peer address, (based on the previous sections).

Figure 19-17. Self Unmatched RPA Address Resolution



19.4 Register Details

Table 19-5 gives the details of all the register fields discussed in this chapter. For more details, see PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

Table 19-5. Register List

Register	Bits	Field Name	Bit Description
BLE_BLELL_ADV_NEXT_INSTANT	15:0	ADV_NEXT_INSTANT	Shows the next start of an advertising event with reference to the internal reference clock.
BLE_BLELL_CLOCK_CONFIG	7	LLH_IDLE	Indicates if hardware is doing any transmit/receive operation. This information is used by firmware to decide to program the hardware into Deep Sleep mode. 1 – LL hardware is idle. 0 – LL hardware is busy. In this case, LL hardware will not enter Deep Sleep mode even if the firmware gives an enter DSM command. (Hardware generates a dsm exit interrupt to inform firmware that DSM entry was not successful).
	10	SM_AUTO_WKUP_EN	Enable Sleep mode auto wakeup enable. 1 – enable, 0 – disable. Enables hardware to automatically wake up from Sleep mode at the instant = *wakeup_instant* – *sm_offset_to_wakeup_instant*. The *wakeup_insant* is the field in the *wakeup control register* described earlier. The *sm_offset_to_wakeup_instant* value is the field described in the wakeup configuration register.
BLE_BLELL_COMMAND_REGISTER	7:0	COMMAND	8-bit command from firmware to the Link Layer controller. see PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM) for the list of instructions and their opcodes. The instruction results in the Link Layer hardware starting/stopping an operation.

Table 19-5. Register List

Register	Bits	Field Name	Bit Description
BLE_BLELL_NEXT_CE_INSTANT	15:0	NEXT_CE_INSTANT	16-bit internal reference clock value at which the next connection event will occur on a connection. The connection index register must be programmed with the index of the connection, before reading the register.
BLE_BLELL_EVENT_INTR	5	DSM_INTR	**Read:** Deep sleep mode exit interrupt. This bit is set when the link layer hardware exits Deep Sleep mode. **Write:** Clear deep sleep mode exit interrupt. Writing to the register with this bit set to 1 clears the interrupt source.
BLE_BLELL_INIT_NEXT_INSTANT	15:0	INIT_NEXT_INSTANT	Shows the instant with respect to internal reference clock of resolution 625 µs at which next initiator scanning event begins.
BLE_BLELL_POC_REG_TIM_CTRL	7:3	BB_CLK_FREQ_MINUS_1	LLH clock configuration. The clock frequency of the clock input to this design is configured in this register. This is used to derive a 1-MHz clock.
BLE_BLELL_SCAN_NEXT_INSTANT	15:0	NEXT_SCAN_INSTANT	Shows the instant with respect to internal reference clock of resolution 625 µs at which next scanning event begins.
BLE_BLELL_TIM_CONTROL_L	15:0	TIM_REF_CLOCK	16-bit internal reference clock. The clock is a free running clock, incremented by a 0.625 ms periodic pulse. It is used as a reference clock to derive all the timing required according to protocol.
BLE_BLELL_WAKEUP_CONFIG	7:0	OSC_STARTUP_DELAY	Oscillator stabilization/startup delay. This is in X.Y format where X is in terms of number of BT slots (625 us) and Y is in terms of number of clock periods of the 16.384-kHz clock input. The 16.384-kHz clock is required for the RF oscillator to stabilize after it is turned ON. In this period, the clock is assumed to be unstable, and so the controller does not turn on the clock to internal logic till this period is over. This means, the wake up from deep sleep mode must account for this delay before the wakeup instant. Osc_startup_delay[7:5] is number of slots(625us) Osc_startup_delay[4:0 is number of clock periods of 16KHz clock (Warning: Min. value of Osc_startup_delay [4:0] sup-ported is 1 and Max. value is 9. Therefore programma-ble range is 1 to 9)
BLE_BLELL_WAKEUP_CONTROL	15:0	WAKEUP_INSTANT	Instant, with reference to the internal 16-bit clock reference, at which the hardware must wakeup from deep sleep mode. This is calculated by firmware based on the next closest instant where a controller operation is required (like advertiser/scanner). Firmware reads the next instant of the procedures in the corresponding *_NEXT_INSTANT registers. This value is used only when hardware auto wakeup from deep sleep mode is enabled in the clock control register. Note: It is recommended to program wakeup_instant such that the actual instant to wakeup is at least two counts (two slots of 625 us) ahead of the reference clock when entering DSM. The actual instant to wakeup is “wakeup_instant – dsm_offset_to_wakeup_instant – osc_startup_delay, and it should be greater than “reference clock + 2”
BLE_BLERD_DBUS	15	XTAL_ENABLE	Crystal enable. High active.
BLE_BLESS_LF_CLK_CTRL	0	DISABLE_LF_CLK	When set to 1, gates the LF clock input to the Link Layer. This is done for extended DSM mode where the DSM state machine needs to be frozen to prevent a default auto exit.
BLE_BLESS_LL_CLK_EN	0	CLK_EN	Set this bit ‘1’ to enable the clock to Link Layer.
BLE_BLESS_LL_DSM_CTRL	0	DSM_EXIT	When the Link Layer is in Deep Sleep mode, firmware can set this bit to wake the Link Layer.
	1	DSM_ENTERED_INTR_MASK	Masks the DSM Entered Interrupt, when disabled.
	2	DSM_EXITED_INTR_MASK	Masks the DSM Exited Interrupt, when disabled.
	3	XTAL_ON_INTR_MASK	Masks the Crystal Stable Interrupt, when disabled.
BLE_BLESS_LL_DSM_INTR_STAT	0	DSM_ENTERED_INTR	On a firmware request to LL to enter into state machine, working on LF clock, LL transitions into Deep Sleep mode and asserts this interrupt. The interrupt can be cleared by writing one into this location.
	1	DSM_EXITED_INTR	On a firmware request to LL to exit from Deep Sleep mode, working on LF clock, LL transitions from Deep Sleep mode and asserts this interrupt when the Deep Sleep clock gater is turned ON. The interrupt can be cleared by writing one into this location.
	8	XTAL_ON_INTR	Enables crystal stable signal rising edge interrupt. The interrupt can be cleared by writing one into this location.
BLE_BLESS_RF_CONFIG	0	RF_ENABLE	Enables the RF oscillator band gap. 1: band gap is enabled 0: band gap is disabled
BLE_BLESS_WCO_CONFIG	31	ENABLE	Master enable for WCO oscillator.

Table 19-5. Register List

Register	Bits	Field Name	Bit Description
BLE_BLESS_XTAL_CLK_DIV_CONFIG	1:0	SYSCLK_DIV	System clock pre-divider value. The 24-MHz crystal clock is divided to generate the system clock. 0: NO_DIV: SYSCLK= XTALCLK/1 1: DIV_BY_2: SYSCLK= XTALCLK/2 2: DIV_BY_4: SYSCLK= XTALCLK/4 3: DIV_BY_8: SYSCLK= XTALCLK/8
	3:2	LLCLK_DIV	Link Layer clock pre-divider value. The 24-MHz crystal clock is divided to generate the Link Layer clock. 0: NO_DIV: LLCLK= XTALCLK/1 1: DIV_BY_2: LLCLK= XTALCLK/2 2: DIV_BY_4: LLCLK= XTALCLK/4 3: DIV_BY_8: LLCLK= XTALCLK/8
WDT_CONFIG	1:0	WDT_MODE0	Watchdog Counter Action on Match (WDT_CTR0=WDT_MATCH0).
		NOTHING	Do nothing
		INT	Assert WDT_INTx
		RESET	Assert WDT Reset
		INT_THEN_RESET	Assert WDT_INTx, assert WDT Reset after third unhandled interrupt
	2	WDT_CLEAR0	Clear Watchdog Counter when WDT_CTR0=WDT_MATCH0. In other words, WDT_CTR0 divides LFCLK by (WDT_MATCH0+1). 0: Free running counter 1: Clear on match
	3	WDT_CASCADE0_1	Cascade Watchdog Counters 0,1. Counter 1 increments the cycle after WDT_CTR0=WDT_MATCH0. 0: Independent counters 1: Cascaded counters
	9:8	WDT_MODE1	Watchdog Counter Action on Match (WDT_CTR1=WDT_MATCH1).
		NOTHING	Do nothing
		INT	Assert WDT_INTx
		RESET	Assert WDT Reset
		INT_THEN_RESET	Assert WDT_INTx, assert WDT Reset after third unhandled interrupt
	10	WDT_CLEAR1	Clear Watchdog Counter when WDT_CTR1=WDT_MATCH1. In other words, WDT_CTR1 divides LFCLK by (WDT_MATCH1+1). 0: Free running counter 1: Clear on match
	11	WDT_CASCADE1_2	Cascade Watchdog Counters 1,2. Counter 2 increments the cycle after WDT_CTR1=WDT_MATCH1. It is allowed to cascade all three WDT counters. 0: Independent counters 1: Cascaded counters
	16	WDT_MODE2	Watchdog Counter 2 Mode.
		NOTHING	Free running counter with no interrupt requests.
		INT	Free running counter with interrupt request when a specified bit in CTR2 toggles (see WDT_BITS2).
	28:24	WDT_BITS2	Bit to observe for WDT_INT2: 0: Assert when bit0 of WDT_CTR2 toggles (one int every tick) .. 31: Assert when bit31 of WDT_CTR2 toggles (one int every 2^31 ticks)
	31:30	LFCLK_SEL	Select source for LFCLK: 0: ILO - Internal R/C oscillator 1: WCO - Internal crystal oscillator 2-3: Reserved - do not use Not all products support all clock sources. Selecting a clock source that is not supported will result in undefined behavior. To safely change LFCLK_SEL, wait for WDT_CTRLOW/WDT_CTRHIGH to change; then, change the setting immediately.

Table 19-5. Register List

Register	Bits	Field Name	Bit Description
RSLV_LIST_ENABLE (4.2)[a]	0	VALID_ENTRY	Indicates if the index is valid.
	1	PEER_ADDR_IRK_SET	Indicates if the listed peer device has shared its IRK. 0 - Identity address in a received packet is accepted. If a valid peer device RPA is available in the list, then the RPA in a received packet is accepted. 1 - Only the peer device RPA, if available in the list, in a received packet is accepted. An Identity address in the received packet is reported as a privacy mismatch.
	2	SELF_ADDR_IRK_SET_RX	Indicates if the local IRK has been shared with the listed peer device. 0 - Self Identity address in a received packet is accepted. If a valid self RPA is available in the list, then the RPA in a received packet is accepted. 1 - Only the self device RPA, if available in the list, in a received packet is accepted. A Self Identity address in the received packet is reported as a privacy mismatch.
	3	WHITELISTED_PEER	Indicates if the listed peer device is in the whitelist.
	4	PEER_ADDR_TYPE	Indicates the address type of the listed peer device.
	5	PEER_ADDR_RPA_VAL	Indicates that the peer device RPA in the list is valid.
	6	SELF_ADDR_RXD_RPA_VAL	Indicates that the received self RPA in the list is valid.
	7	SELF_ADDR_TX_RPA_VAL	Indicates that the self RPA in the list to be transmitted is valid.
	8	SELF_ADDR_INIT_RPA_SEL	When Initiator whitelist is disabled, this bit indicates the specific device from which ADV packets will be accepted.
	9	SELF_ADDR_IRK_SET_RX	Indicates the TX address type to be used for SCANA and INITA. 0 - Self Identity address is used in SCANA/INITA in SCAN_REQ/CONN_REQ packets 1 - Self RPA address provided in RSLV_LIST_TX_INIT_RPA field in the resolving list with the associated valid bit in SELF_ADDR_TX_RPA_VAL above is used in SCANA/INITA in SCAN_REQ/CONN_REQ packets
LL_CONTROL (4.2)[a]	0	PRIV_1_2	Enables Privacy 1.2 feature.
	1	DLE	Enables Data Length Extension feature in DTM, connection, and encryption modules.

a. These registers are only available in Bluetooth 4.2 devices.

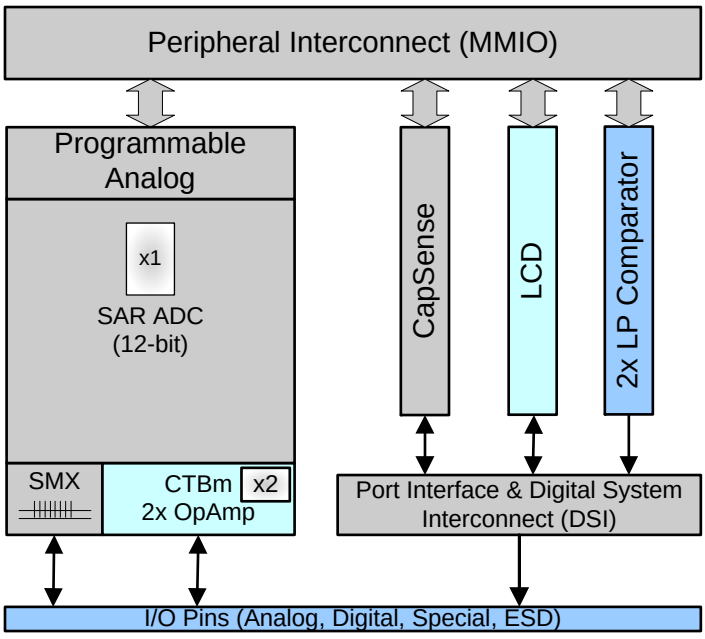
Section E: Analog System

This section encompasses the following chapter:

- Precision Reference chapter on page 197
- SAR ADC chapter on page 201
- Low-Power Comparator chapter on page 245
- Continuous Time Block mini (CTBm) chapter on page 251
- LCD Direct Drive chapter on page 233
- CapSense chapter on page 245
- Temperature Sensor chapter on page 255

Top Level Architecture

Analog System Block Diagram



20. Precision Reference

PRoC has a precision reference block, which creates multiple reference bias voltages and currents for the whole chip. This block is also responsible to provide temperature dependent references to the internal main oscillator (IMO) circuit and the flash memory block for accurate IMO output frequency and error free flash read/write operations respectively, across temperature range of the device.

20.1 Features

The precision reference block has following features:

- Bandgap circuit to generate 1.024 V and 2.4 µA references
- Trim buffer to generate different output voltage levels - 1.2 V, 1.024 V, and 0.8 V with input from the bandgap circuit
- Multiple fast and slow low-power buffers, which not only enhance the drive capability of various reference outputs, but also isolate noise from one another
- Multiple fast and slow current mirror circuits
- Temperature-dependent voltage reference for flash memory
- Temperature-dependent current reference for the IMO

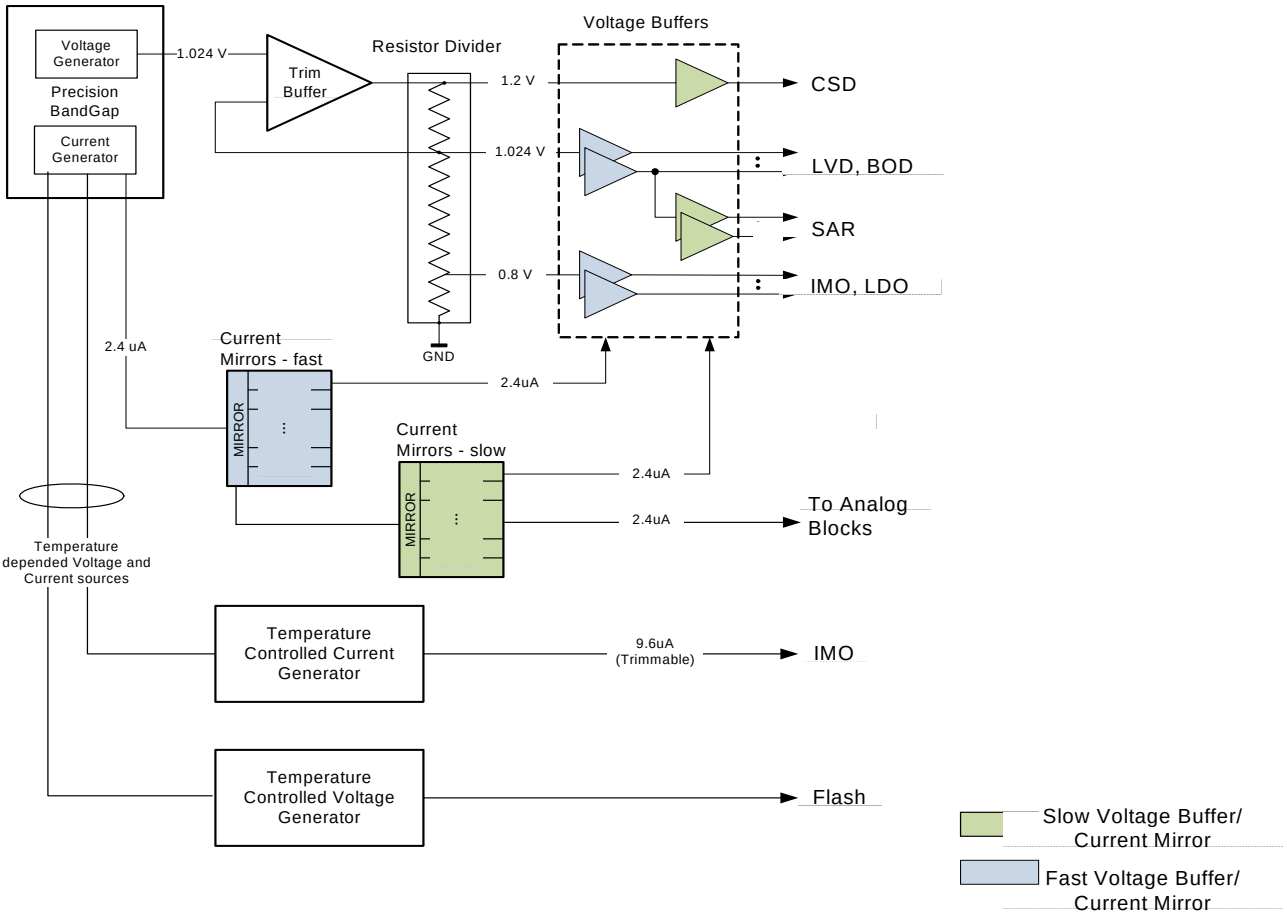
20.2 Block Diagram

Figure 20-1 illustrates the block diagram.

The precision reference is mainly composed of these blocks:

- A precision bandgap block, which generates the precision voltage and current references
- A trim buffer, which generates different output voltage references for various applications and trims the voltage magnitude of 1.024-V output
- A group of fast low-power buffers and slow low-power buffers, which not only enhance the drive capability of various reference outputs, but also isolate the noise from one another
- A group of fast leaf cells and slow leaf cells, which create multiple copies of current references in fast and slow domains, respectively
- A temperature-controlled voltage generator block for the flash system
- A temperature-controlled current source for the IMO

Figure 20-1. Voltage Reference Block Diagram



20.3 How it Works

The work principles of the main components are detailed in this section.

20.3.1 Precision Bandgap

This circuit is the source of all the references generated by the precision reference block. It provides the second order curvature corrected 1.024-V voltage reference and 2.4-uA current reference. The voltage reference is routed to the trim buffer and the current reference is routed to the current mirror circuits.

20.3.2 Trim Buffer

The trim buffer is an opamp network, which takes input from the bandgap circuit and generates three different references (1.2 V, 1.024 V, and 0.8 V). The references are tapped at the resistor array in the feedback network, resulting in high output impedance. This necessitates use of buffers to drive the references.

20.3.3 Low-Power Buffers

PRoC has multiple low-power buffers divided into two groups - fast and slow. These buffers take input from the trim buffer circuit and drive the destination blocks. The fast buffer has the capability to reach within 1 percent of the final value in 9 us. The slow buffer can reach within 40 us. Multiple buffers ensure low reference-line capacitances, which in turn reduces the settling time. Fast voltage buffers are used for the references driven to the blocks that are crucial for system startup. These include the IMO, flash, low dropout (LDO) regulator, low voltage detect (LVD), and brownout detect (BOD) circuit.

The output of the fast buffer is driven to the slow buffer. This ensures that the extra loading due to the non-startup related blocks are isolated from those driven by fast buffers. Slow buffers drive function blocks, such as SAR ADC and CapSense CSD.

Fast buffers are always enabled along with the bandgap block; slow buffers can be individually enabled or disabled by the user using the VREF_EN bits of the PWR_BG_CONFIG register.

20.3.4 Current Mirrors

Current mirror circuits are used to generate multiple copies of 2.4 uA reference from the bandgap circuit. Similar to voltage buffers, there are two types of current mirrors - fast and slow current mirrors. The fast current mirror circuit has a settling time of 9 us to reach within 1 percent of the final voltage and slow current mirror can settle in 40 us. The fast current mirrors are used to provide bias to the fast voltage buffers. The slow current mirror outputs are used to drive the analog blocks such as SAR, CTBm, CSD, and LPCOMP. A 2.4-µA source current from the fast current mirror is driven to the flash block.

20.3.5 Temperature-Controlled Voltage Generator

The bias signal generated by this block controls the reference for flash memory, depending on the temperature. It receives input from the precision bandgap block. The temperature-dependent voltage reference compensates the pump voltage generated in the flash memory block required for proper read and write operations across the temperature range of the device.

20.3.6 Temperature-Controlled Current Generator

This block generates the temperature dependent current reference for the IMO to maintain its clock frequency within ±2% across the device operating temperature.

Table 20-1. Voltage and Current References

Voltage or Current References	Buffer	Accuracy Targets	Block Usage
1.024 V	Fast	±2%	LVD – Low voltage detect on external supply
1.2 V	Slow	±2%	Capsense reference
1.024 V	Slow	±1%	SAR ADC
1.2 V	Fast	±2%	Flash
1.024 V	Fast	±2%	BOD – To detect brownouts on internal voltages
0.8 V	Fast	±2%	IMO – Comparator threshold in relaxation oscillator
0.8 V	Fast	±2%	LDO – V_{CCD} and V_{CCA} regulator reference
2.4 µA	Slow	±2.5%	Bias current for analog circuits (CSD - IDAC, SAR ADC, LPCOMP, CTBm)
2.4 µA	Fast	±2.5%	Flash
9.6 µA	–	±5%	IREF for IMO, with programmable temperature compensation

20.4 Configuration

During power-up, the precision reference block is initialized with default trim settings saved in the nonvolatile latch (NVL) and SFLASH. These settings are programmed during manufacturing and no field adjustment is needed.

21. SAR ADC

The PRoC has one successive approximation register analog-to-digital converter (SAR ADC). The SAR ADC is designed for applications that require moderate resolution and high data rate. It consists of the following blocks (see Figure 21-1):

- SARMUX
- SAR ADC core
- SARREF
- SARSEQ

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

The ninth channel is an injection channel that is used for infrequent and incidental sampling of pins and signals, for example, the internal temperature sensor.

The result from each channel is double-buffered and a complete scan may be configured to generate an interrupt at the end of the scan. Alternatively, the data can be routed to programmable digital blocks (UDBs) for further processing without CPU intervention. The sequencer may also be configured to flag overflow, collision, and saturation errors that can be configured to assert an interrupt.

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

21.1 Features

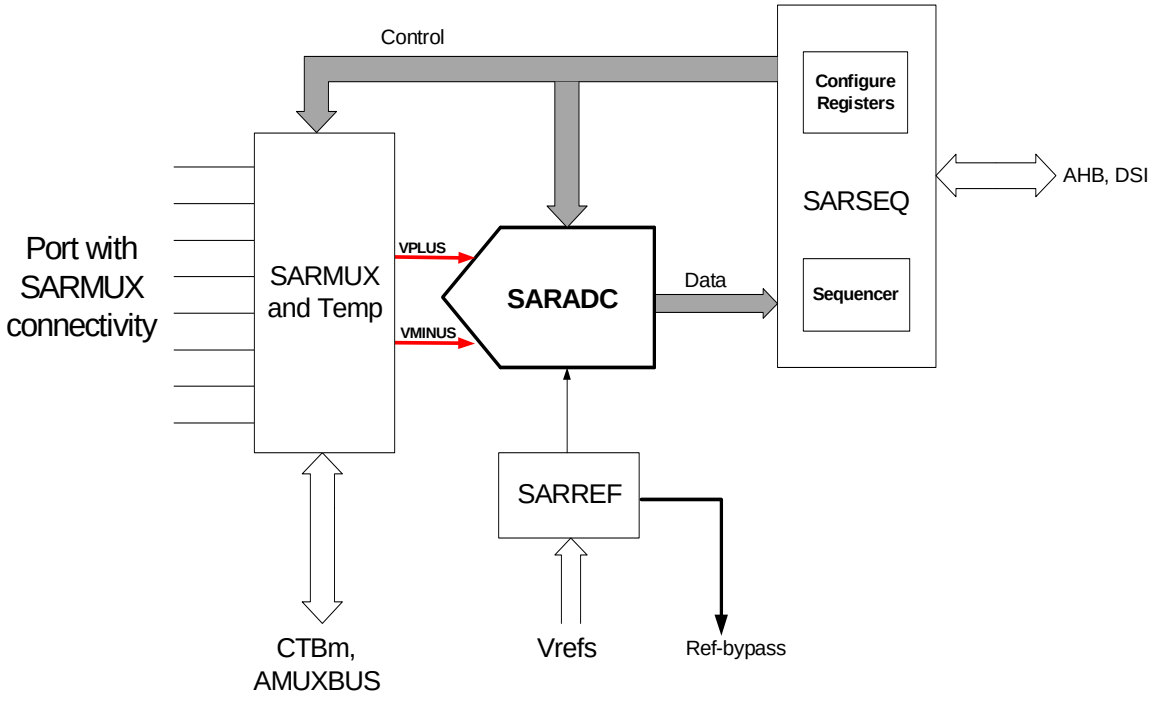
- Operates across the entire device power supply range
- Maximum 1 Msps sample rate
- Eight individually configurable channels and one injection channel
- Each channel has the following features:
  - Input from external pin or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Programmable acquisition times
  - Selectable 8-, 10-, and 12-bit resolution
  - Single-ended or differential measurement
  - Averaging
  - Results are double-buffered
  - Result may be left or right aligned
- Scan triggered by timer, pin, or UDB
  - One shot–periodic or continuous mode
- Hardware averaging support
  - First order accumulate
  - Samples averaging from 2 to 256 (powers of 2)
- Results represented in 16-bit sign extended values
- Selectable voltage references

- Internal V_{DDA} and $V_{DDA}/2$ references
- Internal 1.024-V reference with buffer
- External reference

- Interrupt generation
  - Finished scan conversion
  - Saturation detect and over-range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect
- Configurable injection channel
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single-ended or differential, averaging
- Option to process data in programmable digital blocks to off-load CPU
- Option to control switches from programmable digital blocks
- Option to control SAR ADC and switches from programmable digital blocks
  - Implement an alternative SAR sequencer
  - Able to achieve 1 Msps
- Low-power modes
  - ADC core and reference voltage has low-power mode separately

21.2 Block Diagram

Figure 21-1. Block Diagram



21.3 How it Works

This section includes the following contents:

- Introduction of each block: SAR ADC core, SARMUX, SARREF, and SARSEQ
- SAR ADC system resource: Interrupt, low-power mode, and SAR ADC status
- System operation modes
  - Register mode
  - DSI mode
- Configuration examples

21.3.1 SAR ADC Core

PRoC SAR ADC core is a 12-bit SAR ADC. The maximum sample rate for this ADC is 1 Msps. The SAR ADC core has the following features:

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

21.3.1.1 Single-ended and Differential Mode

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

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

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

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

21.3.1.2 Input Range

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

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

21.3.1.3 Result Data Format

Result data format is configurable from two aspects:

- Singed/unsigned
- Left/right alignment

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

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

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

Table 21-1. Result Data Format

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

Table 21-1. Result Data Format

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

21.3.1.4 Negative Input Selection

The negative input connection choice affects the voltage range, SNR, and effective resolution (see Table 21-2). In single-ended mode, negative input of the SAR ADC can be connected to V_{SSA}, V_{REF}, or any of the eight pins with SARMUX connectivity.

Table 21-2. Negative Input Selection Comparison

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

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

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

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

21.3.1.5 Resolution

PRoC supports 12-bit resolution (default) and a selectable alternate resolution: either 8-bit or 10-bit for each channel.

Resolution affects conversion time:

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

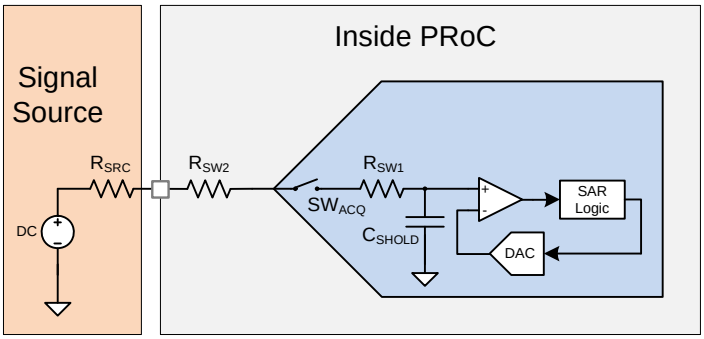
Total acquisition and conversion time (sar_clk) = acquisition time + resolution (bit) + 2

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

21.3.1.6 Acquisition Time

Acquisition time is the time taken by sample and hold (S/H) circuit inside SAR ADC to settle. After acquisition time, the input signal source is disconnected from the SARADC core, and the output of the S/H circuit will be used for conversion. Each channel can select one from four acquisition time options, from 4 to 1023 SAR clock cycles defined in global configuration registers SAR_SAMPLE_TIME01 and SAR_SAMPLE_TIME23.

Figure 21-2. Acquisition Time



The acquisition time should be sufficient to charge the internal hold capacitor of the ADC through the resistance of the routing path, as shown in Figure 21-2. The recommended value of acquisition time is:

$t_{ACQ} >= 9 \times (R_{SCR} + R_{SW2} + R_{SW1}) \times C_{SH}$

Where:

C_{SH} ~= 10 pF

$R_{SW2} + R_{SW1}$ = ~ 500 to 1000 ohms, depending on the routing path (See Analog Routing on page 206 for details).

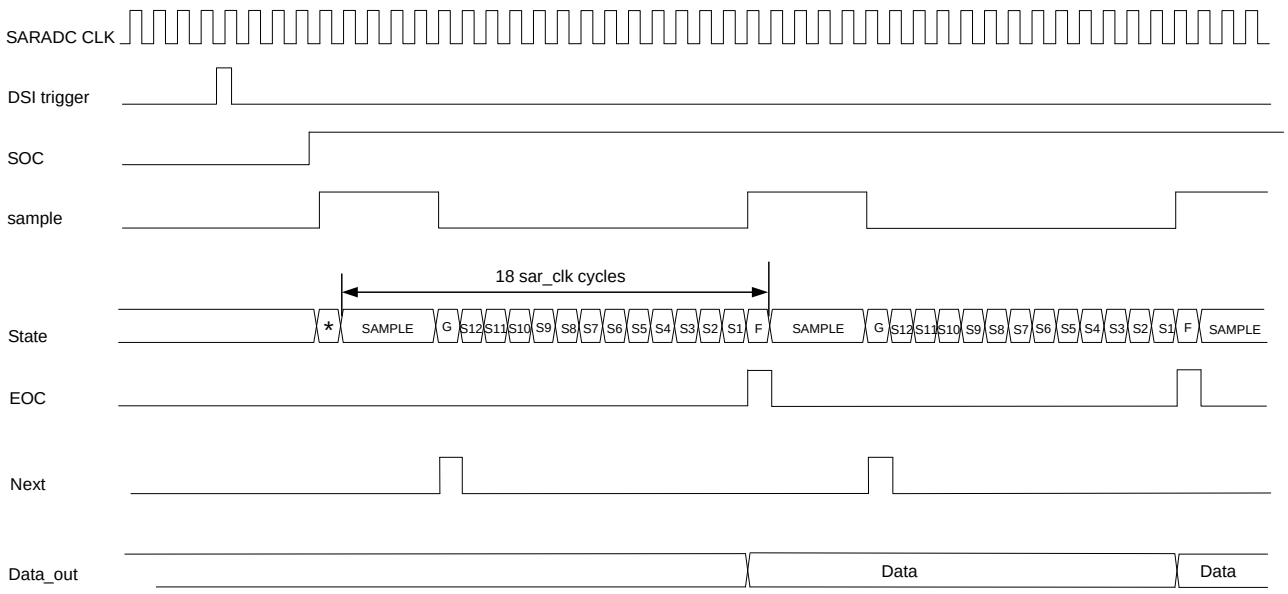
R_{SRC} = series resistance of the signal source

21.3.1.7 SAR ADC Clock

SAR ADC clock frequency must be between 1 MHz and 18 MHz for PRoC BLE200 and 1 MHz to 14.5 MHz for PRoC BLE100 which comes from the IMO via a clock divider. Note that a fractional divider is not supported for SAR ADC. To get a 1-Msps sample rate in PRoC an 18-MHz SAR ADC clock is required. To achieve this, the system clock (IMO) must be set to 36 MHz rather than 48 MHz. To get a 806-ksps sample rate for the PRoC BLE100 device, IMO must be set to 29 MHz. A 12-bit ADC conversion with the default acquisition time of four clocks requires 18 clocks in which to complete. A 10-bit and 8-bit conversion requires 16 and 14 clocks respectively.

21.3.1.8 SAR ADC Timing

Figure 21-3. SAR ADC Timing



As the timing graph shows, there is a sar_clk delay before raising start-of-conversion (SOC). A 12-bit resolution conversion needs 14 clocks (one bit needs one sar_clk, plus two excess sar_clk for G and F state). With acquisition time equal to four sar_clk cycles by default, 18 clock sar_clk cycles are required for total ADC acquisition and conversion. After sample (acquisition), it will output the next pulse (or dsi_sample_done), the SARMUX can route to other pin and signal, it will be done automatically with sequencer control (see SARSEQ on page 214 for details).

21.3.2 SARMUX

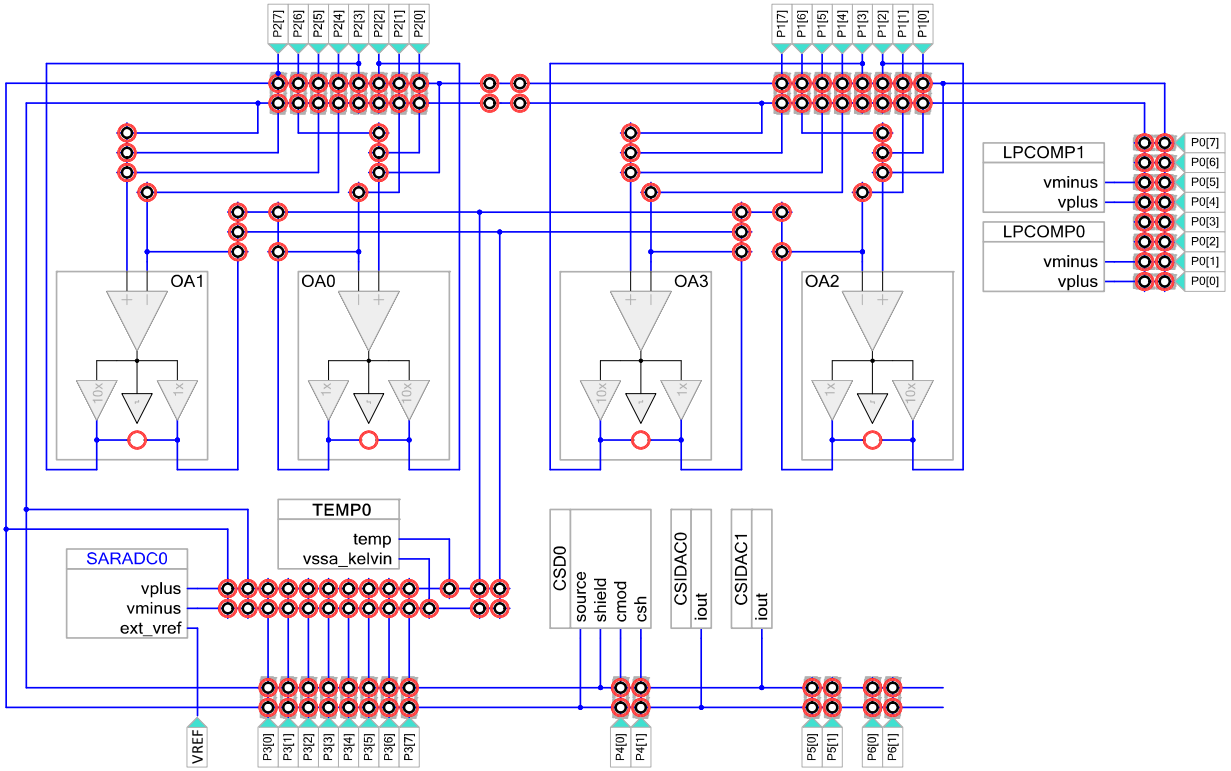
SARMUX is an analog dedicated programmable multiplexer. The main features of SARMUX are:

- Switch on resistance: 600 Ω (maximum)
- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ) or UDBs.
- Charge pump inside:
  - If V_{DDA} < 4.0 V, charge pump should be turned on to reduce switch resistance
  - If V_{DDA} >= 4.0 V, charge pump is turned off and delivers V_{DDA} as its output
- Multiple inputs:
  - Analog signals from pins (port 2)
  - Temperature sensor output
  - CTBm output via sarbus0/1 (not fast enough to sample at 1 Msps)
  - AMUXBUS_A/B (not fast enough to sample at 1 Msps)

21.3.2.1 Analog Routing

SARMUX has many switches that may be controlled by SARSEQ block (sequencer controller), or the DSI. Sequencer and DSI are the hardware control method, which can be masked by the hardware control bit in the register, SAR_MUX_SWITCH_HW_CTRL. Different control methods have different control capability on the switches. See Figure 21-4.

Figure 21-4. SARMUX Switches and Control Capability



Sequencer control: The switches are controlled by the sequencer in SARSEQ block. After configuring each channel's analog routing, it enables multi-channel automatic scan in a round-robin fashion, without CPU intervention. Not every switch can be controlled by the sequencer; see Figure 21-4. The corresponding registers are: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_CTRL, and SAR_MUX_SWITCH_HW_CTRL. The detailed configuration is available in register mode; see Firmware Analog Routing on page 225.

Firmware control: Programmable registers directly define the VPLUS/VMINUS connection. It can control every switch in SARMUX; see Figure 21-4. For example, in firmware control, it is possible to do a differential measurement between any two pins or signals, not just two adjacent pins (as in sequencer control). However, it needs CPU intervention for multi-channel acquisition. The corresponding registers are: SAR_MUX_SWITCH0, SAR_MUX_SWITCH_HW_CTRL. and SAR_CTRL. The detailed configuration is available in register mode; see Firmware Analog Routing on page 225.

DSI control: Switches are controlled by DSI signals from the UDB, which can act as a secondary sequencer with a customized logic design. DSI can control most switches. Thus, it can do a differential measurement between any two pins and signals and firmware control. The detailed configuration is available in DSI mode; see SARMUX Analog Routing on page 221.

21.3.2.2 Analog Interconnection

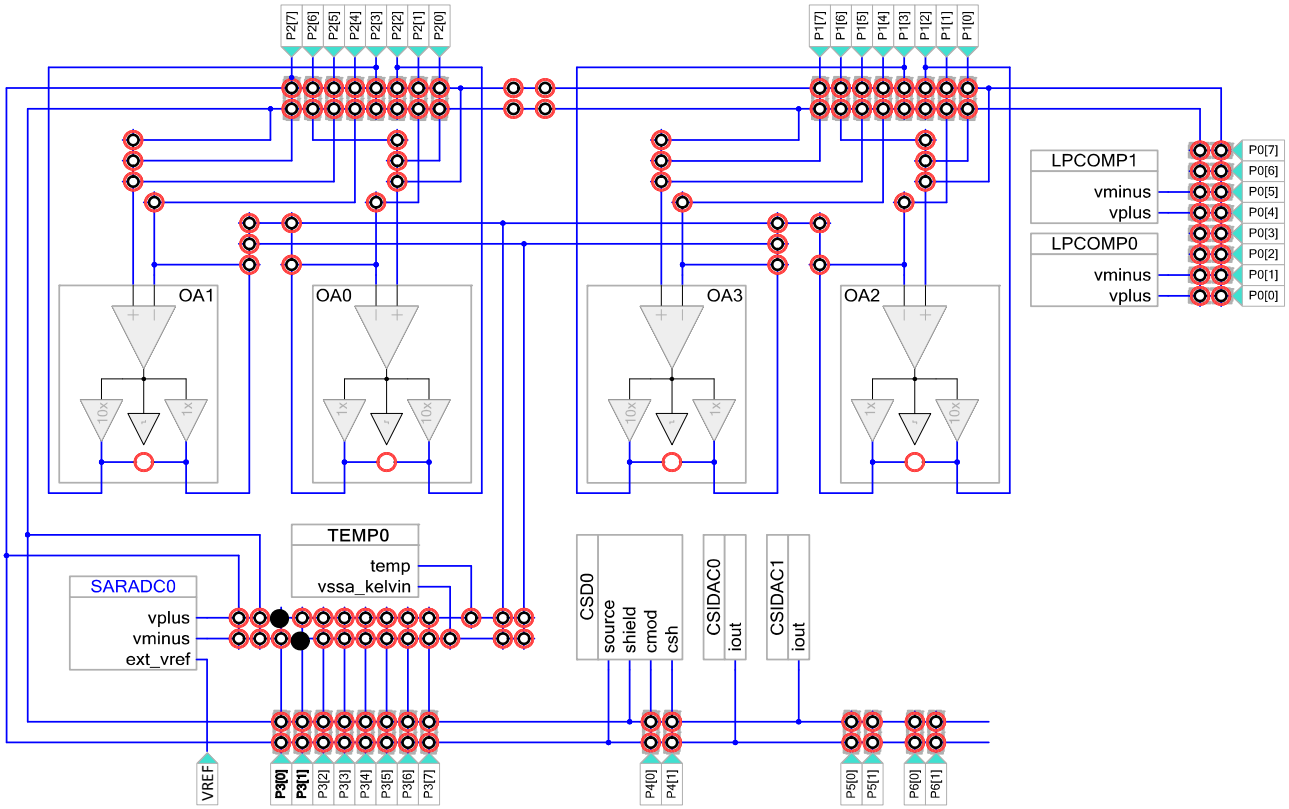
PRoC analog interconnection is very flexible. SAR ADC can be connected to multiple inputs via SARMUX, including both external pins and internal signals. For example, it can connect to a neighboring block such as CTBm. It can also connect to other pins except port 2 through AMUXBUS_A/B, at the expense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

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

Input from External Pins

Figure 21-5 shows how two GPIOs that support SARMUX are connected to SAR ADC as a differential pair (Vpuls/Vminus) via switches. These two switches can be controlled by sequencer, firmware, or DSI. The pins are arranged in adjacent pairs; for example, in SARMUX port P3[0] and P3[1], P3[2] and P3[3], and so on. If you need to use pins that are not paired as a differential pair, such as P3[1] and P3[2], the sequencer does not work; use firmware or DSI.

Figure 21-5. Input from External Pins

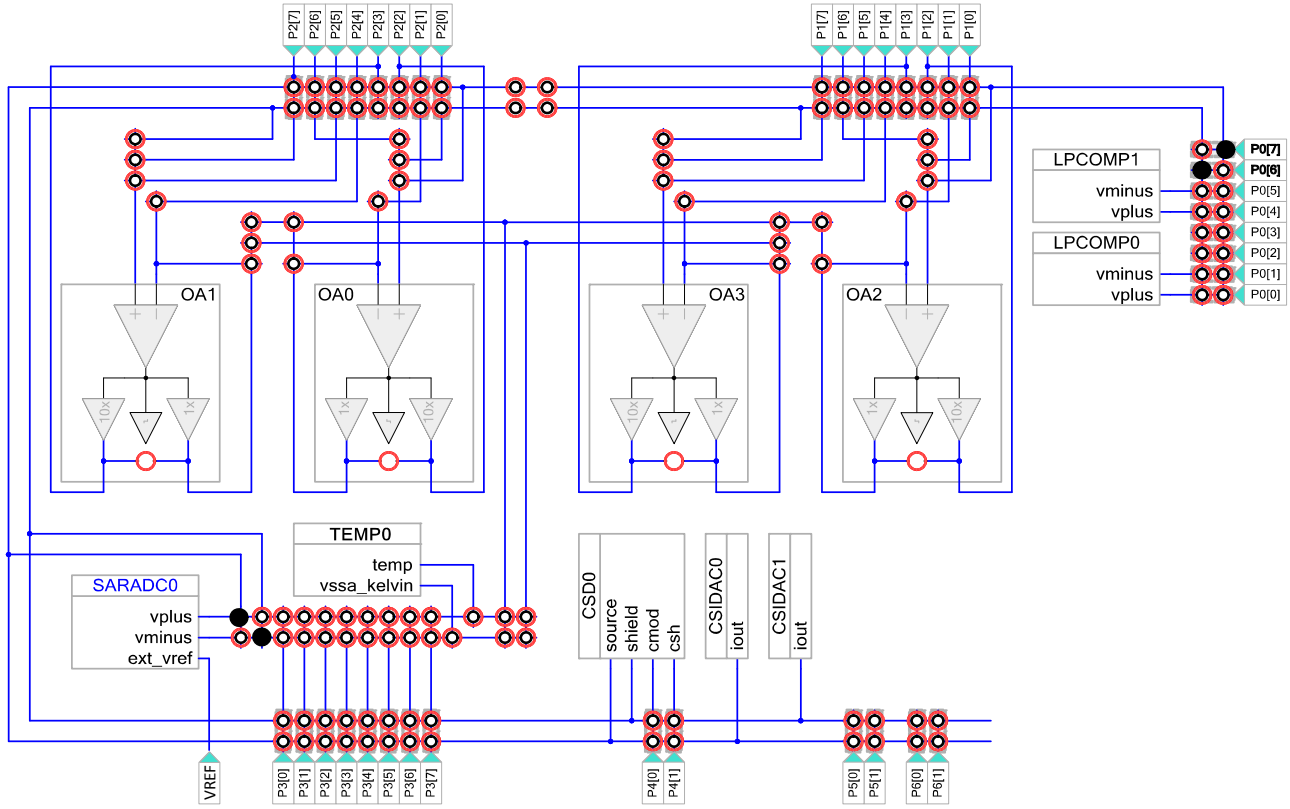


Input from Analog Bus (AMUXBUS_A/B)

Figure 21-6 shows how two pins that do not support SARMUX connectivity are connected to ADC as a differential pair. Additional switches must connect these to two pins: AMUXBUS_A and AMUX-BUS_B, and then connect AMUXBUS_A and AMUXBUS_B to ADC.

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

Figure 21-6. Input from Analog Bus



Input from CTBm Output via sarbus

SAR ADC can be connected to CTBm output via sarbus 0/1. Figure 21-7 shows how to connect an opamp (configured as a follower) output to a single-ended SAR ADC. Negative terminal is connected to V_{REF}. Figure 21-8 shows how to connect two opamp outputs to SAR ADC as a differential pair. It must connect opamp output to sarbus 0/1, then connect SAR ADC input to sarbus 0/1. Because there are also additional switches, it is not fast enough to sample at 1 Msps. However, the on-chip opamps add value for many applications.

Figure 21-7. Input from CTBm Output via sarbus

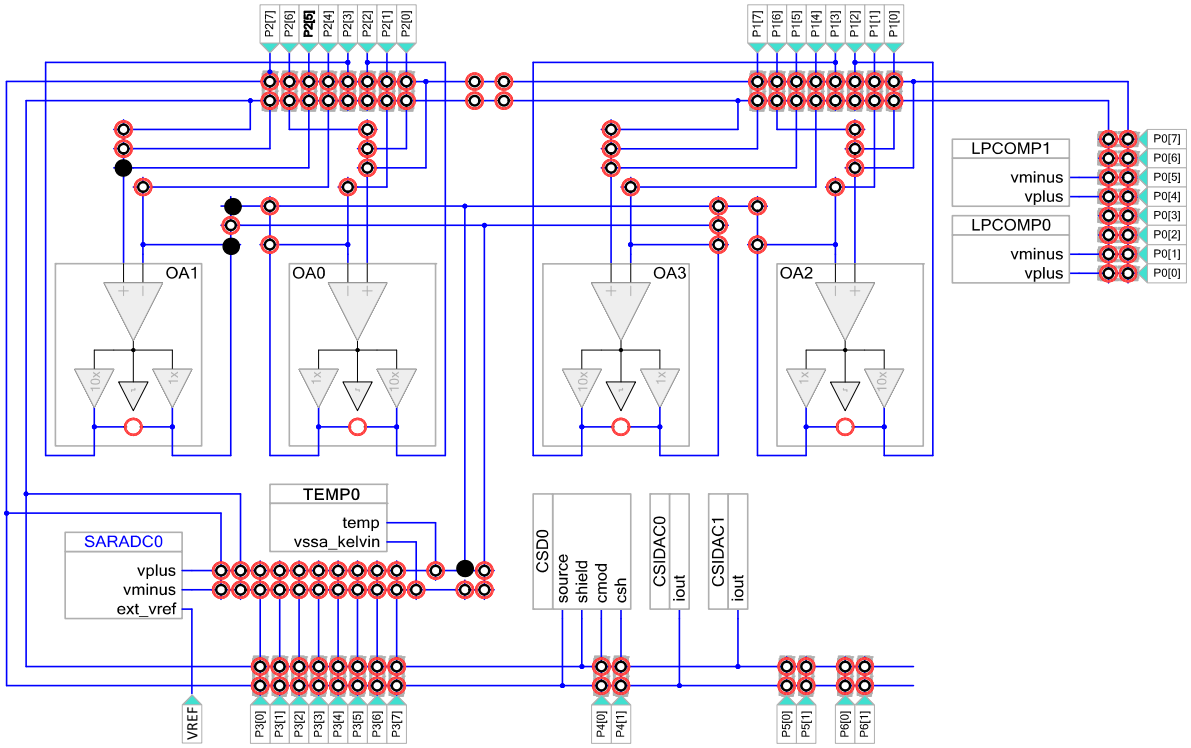
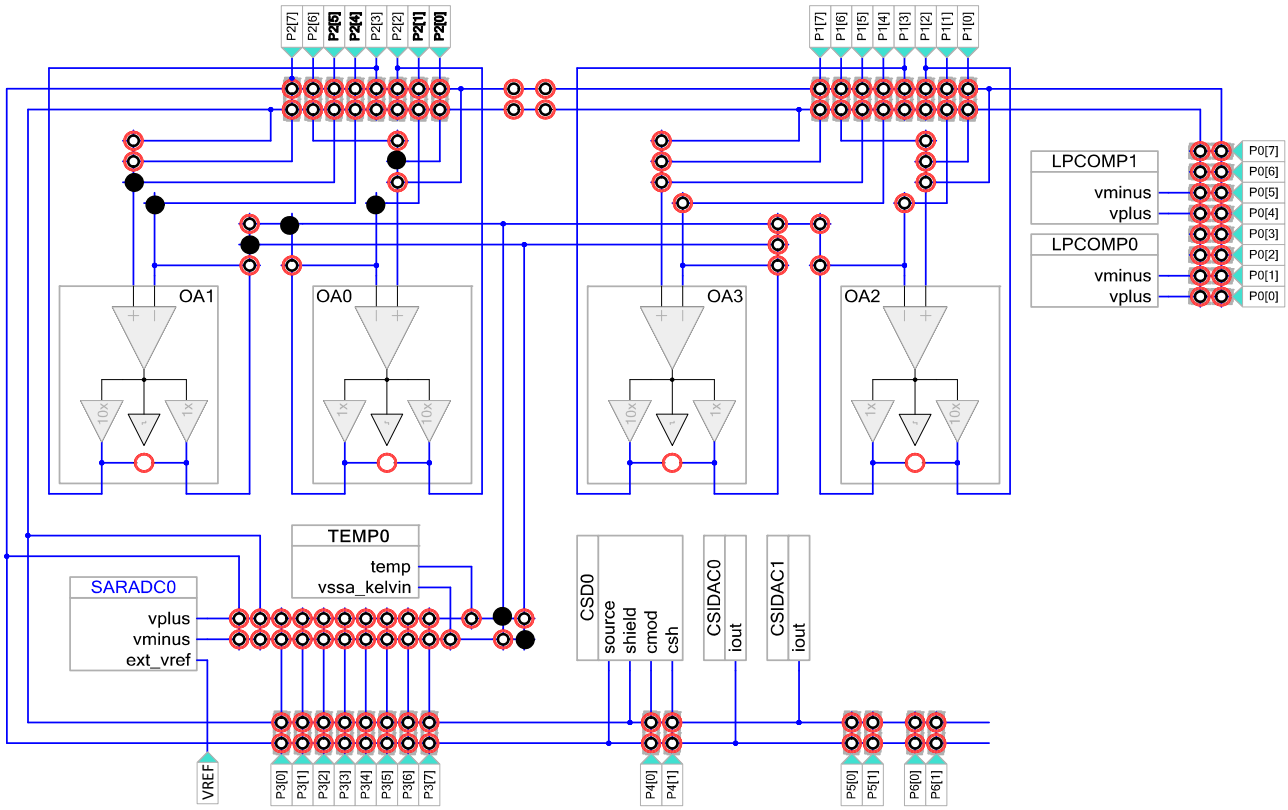


Figure 21-8. Inputs from CTBm Output via sarbus0 and sarbus1

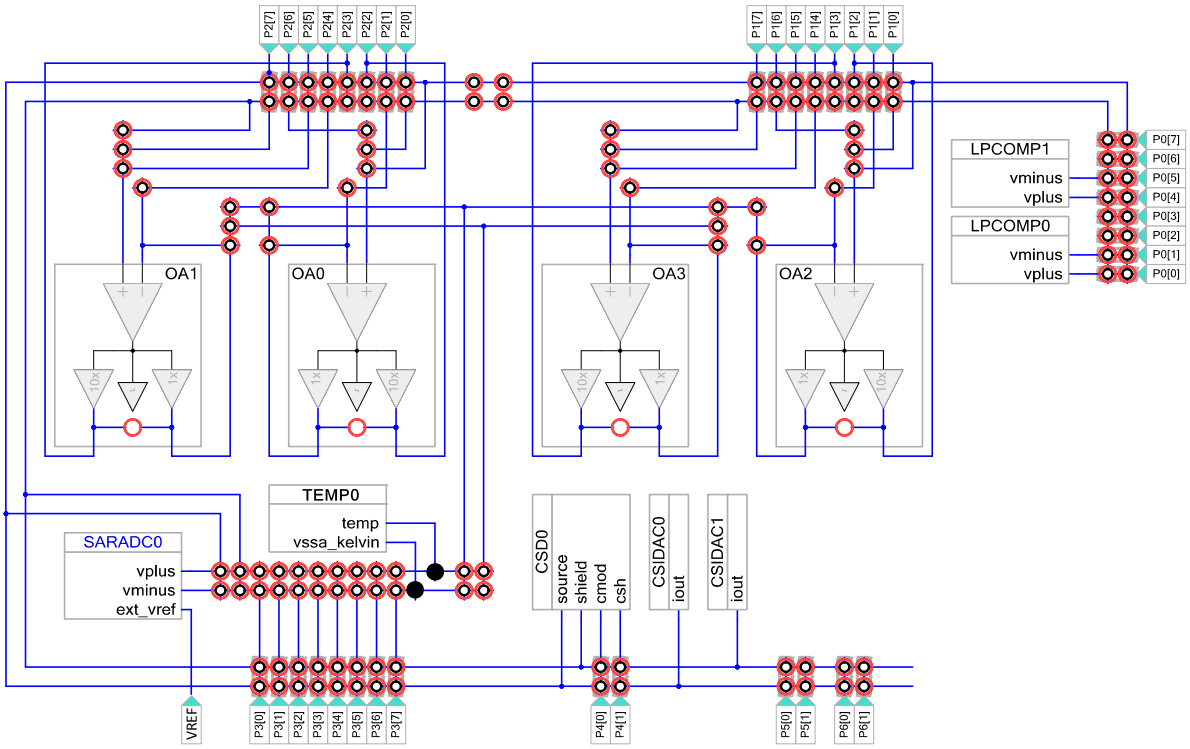


Input from Temperature Sensor

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Note for temperature sensor, differential conversions are not available (conversion result is undefined), thus always use it in singled-ended mode. Reference is from internal 1.024 V.

As Figure 21-9 shows, temperature sensor can be routed to positive input of SAR ADC via switch, which can be controlled by sequencer, firmware, or DSI. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit will disable temperature sensor by cutting its bias current.

Figure 21-9. Inputs from Temperature Sensor

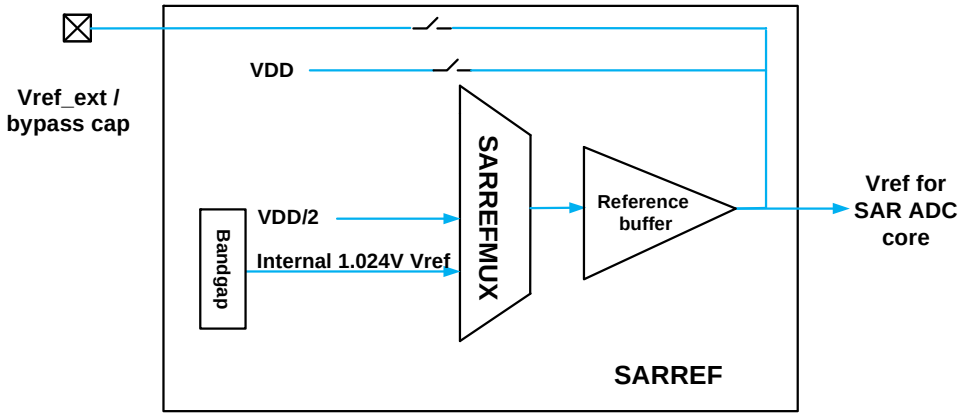


21.3.3 SARREF

The main features of SARREF are:

- Reference options: V_{DDA}, $V_{DDA}/2$, 1.024-V bandgap (±1 percent), external reference
- Reference buffer + bypass cap to enhance internal reference drive capability

Figure 21-10. SARREF Block Diagram



21.3.3.1 Reference Options

The reference voltage selection for the SAR ADC consists of a reference mux and switches inside the SARREF. The selection allows connecting V_{DDA}, $V_{DDA}/2$, and 1.024-V internal reference from a bandgap or an external V_{REF} connected to an Ext Vref/SAR bypass pin (see the device datasheet for details). The control for the reference mux in SARREF is in the global configuration register SAR_CTRL [6:4].

21.3.3.2 Bypass Capacitors

The internal references, 1.024 V from bandgap or $V_{DDA}/2$ are buffered with the reference buffer. This reference may be routed to the Ext Vref/SAR bypass pin where an external capacitor can be used to filter internal noise that may exist on the reference signal.

The SAR ADC sample rate cannot exceed 100 ksps without an external reference bypass capacitor. For example, without a bypass capacitor and with 1.024-V internal V_{REF}, the maximum SAR ADC clock frequency is 1.6 MHz. When using an external reference, it is recommended that an external capacitor is used. Bypass capacitors can be enabled by setting SAR_CTRL [7].

Table 21-3 lists different reference modes and its maximum frequency/sample rate for 12-bit continuous mode operation.

Table 21-3. Reference Modes

Reference Mode	Reference SAR_CTRL [6:4]	Bypass Cap SAR_CTRL[7]	Buffer	Max Frequency	Max Sample Rate
1.024 V internal V_{REF} without bypass cap	4	0	Yes	1.6 MHz	100 ksps
1.024 V internal V_{REF} with bypass cap	4	1	Yes	18 MHz	1 Msps
External V_{REF}	5	X	No	18 MHz	1 Msps
$V_{DDA}/2$ without bypass cap	6	0	Yes	1.6 MHz	100 ksps
$V_{DDA}/2$ with bypass cap	6	1	Yes	18 MHz	1 Msps
V_{DDA}	7	X	No	9 MHz	500 ksps

1.024-V internal V_{REF} startup time varies with the different bypass capacitor size, Table 21-4 lists two common values for the bypass capacitor and its startup time specification. If reference selection is changed between scans, make sure the 1.024-V

internal V_{REF} is settled when SAR ADC starts sampling.

Table 21-4. Bypass Capacitor Values

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

21.3.3.3 Input Range versus Reference

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

21.3.4 SARSEQ

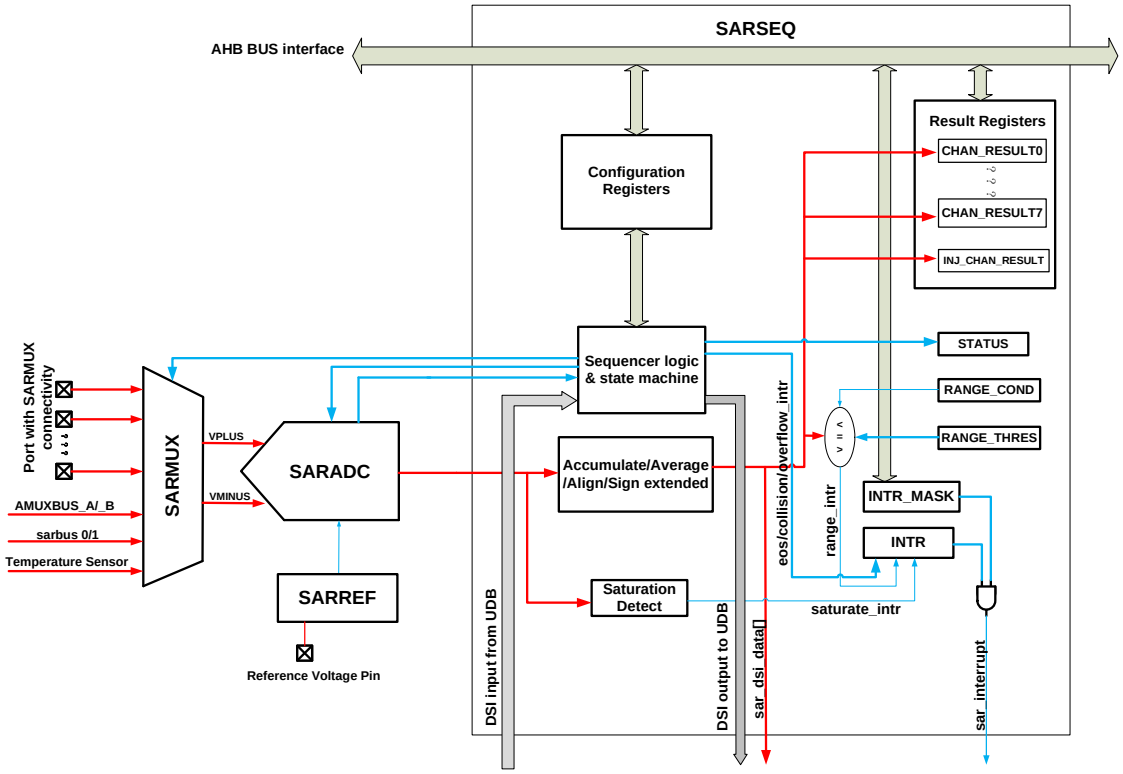
SARSEQ is a dedicated sequencer controller that automatically sequences the input mux from one channel to the next while placing the result in an array of registers, one per channel.

- Control SARMUX analog routing automatically without CPU intervention
- Control SAR ADC core (such as resolution, acquisition time, and reference)
- Receive data from SAR ADC and pre-process (average, range detect)
- Results are double-buffered so the CPU can safely read the results of the last scan while the next scan is in progress.

The features of SARSEQ are:

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

Figure 21-11. SARSEQ Block Diagram



21.3.4.1 Averaging

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

In register control mode, channel configuration SAR_CHAN_CONFIG register has an enable bit (AVG_EN) to enable averaging. In DSI control mode, average is enabled by dsi_cfg_average signal.

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

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

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

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

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

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

21.3.4.2 Range Detection

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

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

0: result < RANGE_LOW (below range)

1: RANGE_LOW ≤ result < RANGE_HIGH (inside range)

2: RANGE_HIGH ≤ result (above range)

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

See Range Detection Interrupts on page 217 for details.

21.3.4.3 Double Buffer

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

21.3.4.4 Injection Channel

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

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

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

- Maskable end-of-conversion interrupt INJ_EOC_INTR
- Maskable range detect interrupt INJ_RANGE_INTR
- Maskable saturation detect interrupt INJ_SATURATE_INTR
- Maskable collision interrupt INJ_COLLISION_INTR

SAR_INTR, SAR_INTR_MASK, SAR_INTR_MASKED, and SAR_INTR_SET are the corresponding registers.

These features are described in detail in Global SARSEQ Configuration on page 221, Channel Configurations on page 222, and Interrupt on page 216.

Tailgating

The injection channel conversion can be triggered by setting the start or enable bit INJ_START_EN (SAR_INJ_CHAN_CONFIG [31]). You should select tailgating by setting INJ_TAILGATING=1 (SAR_INJ_CHAN_CONFIG [30]). The injection channel will be scanned at the end of the ongoing scan of regular channels without any collision. However, if there is no ongoing scan or the SAR ADC is idle, INJ_START_EN will enable the injection channel to be scanned at the end of the next scan of regular channels. After completing the conversion for the injection channel, the end-of conversion interrupt (INJ_EOC_INTR) is set and the INJ_START_EN bit is cleared. The conversion data of the injection is put in the SAR_INJ_RESULT register. Similar to the SAR_CHAN_RESULT, the registers contain mirror bits for "valid" (=INJ_EOC_INTR), range detect, saturation detect interrupt, and a mirror bit of the collision interrupt (INJ_COLLISSION_INTR).

21.3.5 Interrupt

Each of the interrupts described in this section has an interrupt mask in the SAR_INTR_MASK register. By making the interrupt mask low, the corresponding interrupt source is ignored. The SAR interrupt is generated if the interrupt mask bit is high and the corresponding interrupt source is pending.

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

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

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

21.3.5.1 End-of-Scan Interrupt (EOS_INTR)

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

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

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

21.3.5.2 Overflow Interrupt

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

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

21.3.5.3 Collision Interrupt

It is possible that a new trigger is generated while the SARSEQ is still busy with the scan started by the previous trigger. Therefore, the scan for the new trigger is delayed until after the ongoing scan is completed. It is important to

notify the firmware that the new sample is invalid. This is done through the collision interrupt, which is raised any time a new trigger, other than the continuous trigger, is received.

There are two collision interrupts: for the DSI trigger (DSI_COLLISION_INTR), and for the injection channel (INJ_COLLISION_INTR). This allows the firmware to identify which trigger collided with an ongoing scan.

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

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

21.3.5.4 Injection End-of-Conversion Interrupt (INJ_EOC_INTR)

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

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

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

21.3.5.5 Range Detection Interrupts

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

Range detection interrupt for a specified channel can be masked by setting the SAR_RANGE_INTR_MASK register specified bit to ‘0’. Register SAR_RANGE_INTR_MASKED reflects a bitwise AND between the interrupt request and mask registers. If the value is not zero, then the SAR interrupt signal to the NVIC is high.

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

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

21.3.5.6 Saturate Detection Interrupts

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

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

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

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

21.3.5.7 Interrupt Cause Overview

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

21.3.6 Trigger

The two possible ways to trigger a scan are:

- A periodic trigger comes in over the DSI connections (dsi_trigger). This trigger is connected to the output of a TCPWM; however, it can also be connected to any GPIO pin or a UDB. The UDB can implement a state machine looking for a certain sequence of events.
- A continuous trigger is activated by setting the CONTINUOUS bit in SAR_SAMPLE_CTRL register. In this mode, after completing a scan the SARSEQ starts the next scan immediately; therefore, the SARSEQ is always BUSY. As a result, all other triggers are essentially ignored.

The two triggers are mutually exclusive, although there is no hardware requirement. When a DSI trigger coincides with a continuous trigger, both triggers are effectively handled at the same time (a collision interrupt may be set for the DSI trigger).

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

21.3.6.1 DSI Trigger Configuration

- DSI Synchronization

The DSI interface of SARSEQ runs at the system clock frequency (clk_sys); see Clocking System chapter on page 73 for details. If the incoming DSI trigger signal is not synchronous to the AHB clock, the signal needs to be synchronized by double flopping it (default). However, if the DSI trigger signal is already synchronized with the AHB clock, then these two flops can be bypassed. The configuration bit, DSI_SYNC_TRIGGER in the SAR_SAMPLE_CTRL register, controls the double flop bypass. DSI_SYNC_TRIGGER affects the trigger width (TW) and trigger interval (TI) requirement of the DSI pulse trigger signal.

- DSI Trigger Level

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

- Transmission Time

After the 'dsi_trigger' is raised, it takes some transmission time before the SAR ADC is told to start sampling. With different DSI_SYNC_TRIGGER and DSI_TRIGGER_LEVEL configuration, the transmission time is different; Table 21-5 shows the maximum time. Two trigger pulse intervals should be longer than the transmission time, otherwise, the second trigger is ignored.

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

Table 21-5. DSI Trigger Maximum Time

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

Table 21-6. Trigger Signal Requirement

Trigger Spec	Requirement
Trigger Width (TW)	TW should be greater enough so that a trigger can be locked. If DSI_SYNC_TRIGGER=1, TW >= 2 clk_sys cycle. If DSI_SYNC_TRIGGER=0, TW >= 1 SAR clock cycle.
Trigger interval (TI)	Trigger interval of the DSI pulse trigger signal should be longer than the transmission time (as specified in Table 21-5); otherwise, the second trigger pulse will be ignored.

21.3.7 SAR ADC Status

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

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

SAR_MUX_SWITCH_STATUS register gives the current switch status of MUX_SWITCH0 register.

These status registers help to debug SAR behavior.

21.3.8 Low-Power Mode

The current consumption of the SAR ADC can be divided into two parts: SAR ADC core and SARREF. There are several methods to reduce the power consumption of the SAR operation. The easiest way is to reduce the trigger frequency; that is, reduce the number of conversions per second.

The SAR ADC offers the ICONT_LV[1:0] configuration bits, which control overall power of the SAR ADC. Maximum

clock rates for each power setting should be observed.

Table 21-7. ICONT_LV for Low Power Consumption

ICONT_LV[1:0]	Relative Power of SAR ADC Core (%)	Maximum Frequency [MHz]	Minimum Sample Time [cycles]	Maximum Sample Speed (at 12-bit) [ksps]
0	100	18	4	1000
1	50	4.5	3	250
2	133	18	4	1000
3	25	4.5	2.25	125

Finally, to reduce power, use a lower resolution on channels that do not need high accuracy. This shortens the conversion by up to four out of 18 cycles (for 8-bit resolution and minimum sample time).

21.3.9 System Operation

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

1. Set SAR ADC control mode: 21.3.10 Register Mode or 21.3.11 DSI Mode
2. Set SARMUX analog routing (pin/signal selection) via sequencer/firmware/DSI
3. Set the global SARSEQ conversion configurations
4. Configure each channel source (such as pin address)
5. Enable the channels
6. Set the trigger type
7. Set interrupt masks
8. Start the trigger source
9. Retrieve data after each end of conversion interrupt
10. Do injection conversions if needed

Register mode means using registers to control the SARMUX and SAR ADC conversion; DSI mode means using DSI from UDB to control. The major difference between these two control modes is shown in Table 21-8. DSI mode can be enabled by setting DSI_MODE bit (SAR_CTRL [29]).

Table 21-8. Difference between Control Modes

Control Mode	Register	DSI
DSI_MODE	0	1
SARMUX control	Sequencer control registers: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL SAR_CTRL Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL	DSI signal control signals: dsi_out, dsi_oe,dsi_swctrl, dsi_sw_negvref Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL
Global configuration	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND
Channel configuration	Channel configure registers: CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG	By DSI signal: dsi_cfg_st_sel, dsi_cfg_average, dsi_cfg_resolution, dsi_cfg_differential (CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG are ignored)
Trigger	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])
Interrupt	All Apply	All Apply (only EOS_INTR, RANGE_INTR, SATURATE_INTR output on DSI signal)

Table 21-8. Difference between Control Modes

Control Mode	Register	DSI
DSI output	Support	Support
Result data	8 channel result registers 1 injection channel result register	Only channel0 result register is available
Injection	Support	Not supported
Average	Support average on one PIN/signal	Support average on different PIN/signal

21.3.10 Register Mode

Use registers to configure the SAR ADC; this is the most common usage. Detailed register bit definition is available in PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

21.3.10.1 SARMUX Analog Routing

In register mode, there are two ways to control the SARMUX analog routing: sequencer and firmware.

Sequencer Control

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

With sequencer control, the pin or internal signal a channel converts is specified by the combination of port and pin address. The PORT_ADDR bits are SAR_CHANx_CONFIG [6:4] and PIN_ADDR bits are SAR_CHANx_CONFIG [2:0]. Table 21-9 shows the PORT_ADDR and PIN_ADDR setup with corresponding SARMUX selection. The unused port/pins are reserved for other products in the PRoC series.

Table 21-9. PORT_ADDR and PIN_ADDR

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

a. sarbus0 and sarbus1 connect to the output of the CTBm block, which contains opamp0/1. When PORT_ADDR=1, sarbus0 connects to positive terminal of SAR ADC regardless of the value of PIN_ADDR; sarbus1 can only connect to the negative terminal of SAR ADC when differential mode is enabled and PORT_ADDR=1.

For differential conversion, the negative terminal connection is dependent on the positive terminal connection, which is defined by PORT_ADDR and PIN_ADDR. By setting DIFFERENTIAL_EN, the channel will do a differential conversion on the even/odd pin pair specified by the pin address with PIN_ADDR [0] ignored. P3.0/P3.1, P3.2/P3.3, P3.4/P3.5, P3.6/P3.7 are valid differential pairs for sequencer control. More flexible analog can be implemented by firmware or DSI.

For single-ended conversions, NEG_SEL (SAR_CTRL [11:9]) is intended to decide which signal is connected to negative input. In differential mode, these bits are ignored. Negative input choice affects the input voltage range and effective resolution. See Negative Input Selection on page 204 for details. The options include: V_{SSA}, V_{REF}, or an external input from any of the eight pins with SARMUX connectivity. To connect negative input to V_{REF}, an additional bit, SAR_HW_CTRL_NEGVREF (SAR_CTRL[13]) must be set, because the MUX_SWITCH_HW_CTRL register does not have that hardware control bit.

Firmware Control

By default, the SARMUX operates in firmware control. VPLUS (positive) and VMINUS (negative) inputs of SAR ADC can be controlled separately by setting the appropriate bits in SAR_MUX_SWITCH0 [29:0]. Clear appropriate bits in the hardware switch control register (SAR_MUX_SWITCH_HW_CTR[n]=0). Otherwise, hardware control method (sequencer/DSI) will control the SARMUX analog routing.

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

NEG_SEL (SAR_CTRL [11:9]) decides which signal is connected to the negative terminal (vminus) of SAR ADC in single-ended mode. In differential mode, these bits are ignored. In single-ended mode, when using sequencer control, you must set these bits. When using firmware control, NEG_SEL is ignored and SAR_MUX_SWITCH0 should be set to control the negative input. A special case is when SAR_MUX_SWITCH0 does not connect internal V_{REF} to vminus; then, set NEG_SEL to '7'. Negative input choice affects the input voltage range, SNR, and effective resolution. See Negative Input Selection on page 204 for details.

21.3.10.2 Global SARSEQ Configuration

A number of conversion options that apply to all channels are configured globally. In several cases, the channel configuration has bits to choose what parts of the global configuration to use. Global configuration is applied to both register control and DSI control mode.

SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, and SAR_RANGE_COND are all global configuration registers.

Typically, these configurations should not be modified while a scan is in progress. If configuration settings that are in use are changed, the results are undefined. Configuration settings that are not currently in use can be changed without affecting the ongoing scan.

Table 21-10. Global Configuration Registers

Configurations	Control Registers	Detailed Reference
Reference selection	SAR_CTRL[6:4]	21.3.3.1 Reference Options
Signed/unsigned selection	SAR_SAMPLE_CTRL [3:2]	21.3.1.3 Result Data Format
Data left/right alignment	SAR_SAMPLE_CTRL [1]	21.3.1.3 Result Data Format
Negative input selection in single-ended mode	SAR_CTRL[11:9]	21.3.1.4 Negative Input Selection
Resolution	SAR_SAMPLE_CTRL[0][a]	21.3.1.5 Resolution
Acquisition time	SAR_SAMPLE_TIME01 [25:0] SAR_SAMPLE_TIME32 [25:0]	21.3.1.6 Acquisition Time
Averaging count	SAR_SAMPLE_CTRL[7:4]	21.3.4.1 Averaging
Range detection	SAR_RANGE_THRES [31:0] SAR_RANGE_COND [31:30]	21.3.4.2 Range Detection

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

21.3.10.3 Channel Configurations

Channel configuration includes:

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

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

Table 21-11. Channel Configuration Registers

Configurations	Registers	Detailed Reference
Single-ended/differential	SAR_CHANx_CONFIG [8]	21.3.1.1 Single-ended and Differential Mode
Acquisition time selection	SAR_CHANx_CONFIG [13:12]	21.3.1.6 Acquisition Time
Resolution selection	SAR_CHANx_CONFIG [9]	21.3.1.5 Resolution
Average enable	SAR_CHANx_CONFIG [10]	21.3.4.1 Averaging
DSI output enable	SAR_CHANx_CONFIG [30]	21.3.11.8 DSI Output Enable

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

Table 21-12. Resolution

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

21.3.10.4 Channel Enables

A CHAN_EN register is available to individually enable each channel. All enabled channels are scanned when the next trigger happens. After a trigger, the channel enables can

immediately be updated to prepare for the next scan. This does not affect the ongoing scan. Note that this is an exception to the rule; all other configurations (global or channel) should not be changed while a scan is in progress.

21.3.10.5 Interrupt Masks

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

- End-of-scan interrupt
- Overflow interrupt
- Collision interrupt
- Injection end-of-conversion interrupt
- Range detection interrupt
- Saturate detection interrupt

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

See 21.3.5 Interrupt for details.

21.3.10.6 Trigger

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

- Firmware trigger: SAR_START_CTRL [0]
- DSI trigger: dsi_trigger
- Continuous trigger: SAR_SAMPLE_CTRL [16]

See 21.3.6 Trigger for details.

21.3.10.7 Retrieve Data after Each Interrupt

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

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

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

For firmware convenience, bit [31] in SAR_CHAN_WORK register is the mirror bit of the corresponding bit in SAR_CHAN_WORK_VALID register. Bit[29], bit [30],and bit[31] in SAR_CHAN_RESULT are the mirror bits of the corresponding bit in SAR_SATURATE_INTR, SAR_RANGE_INTR, and SAR_CHAN_RESULT_VALID registers. Note that the interrupt bits mirrored here are the raw (unmasked) interrupt bits. It helps firmware to check if the data is valid by just reading the data register.

If DSI output is enabled, it allows the SARSEQ result data to be processed by the UDBs and the channel number allows the possibility of applying different processing to data of different channels. See 21.3.11.8 DSI Output Enable for detailed description.

21.3.10.8 Injection Conversions

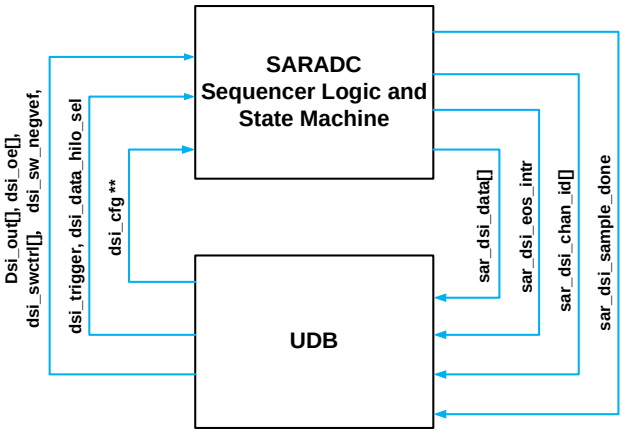
Injection channel can be triggered by setting the start bit INJ_START_EN (INJ_CHAN_CONFIG [31]). To prevent the collision of regular automatic scan, it is recommended to enable tailgating by setting INJ_CHAN_CONFIG [30]. When it is enabled, INJ_START_EN will enable the injection channel to be scanned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

See 21.3.4.4 Injection Channel for details.

21.3.11 DSI Mode

In DSI control mode, all of SAR ADC configuration can be done by DSI signals from UDB except the global configuration, such as interrupt masks, range detect settings, and triggers. The major difference between DSI mode and register mode is that the DSI mode allows hardware to dynamically control the ADC configuration. Figure 21-12 is a subset of the SAR ADC block diagram (Figure 21-1), which specifies the DSI input and output signals.

Figure 21-12. DSI Control Mode Block Diagram



The DSI control mode is selected by setting the DSI_MODE bit in the SAR_CTRL register. In this mode, the SARSEQ ignores all channel configurations in CHAN_EN, CHAN_CONFIG, and INJ_CHAN_CONFIG. Instead, it uses the configuration coming in via the DSI signal.

The following DSI signals are used.

Table 21-13. DSI Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging, if available) for one channel; the internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0]. If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>.
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_out	8	dsi_out[0]=1, P3.0 connected to ADC dsi_out[1]=1, P3.1 connected to ADC … dsi_out[7]=1, P3.7 connected to ADC **Note** MUX_SWITCH0 configuration determines whether the pin is connected to vplus or vminus.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, opamp0 output connected to ADC dsi_oe[3]=1, opamp1 output connected to ADC **Note** MUX_SWITCH0 configuration determines whether the signal is connected to vplus or vminus.
dsi_swctrl[0]	1	SARMUX analog switch control, connect vssa_kelvin to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temp_sens to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input
dsi_cfg_st_sel	2	Configuration control for DSI control mode: select 1 of 4 global sample times
dsi_cfg_average	1	Configuration control for DSI control mode: enable averaging

Table 21-13. DSI Signals

Signal	Width	Description
dsi_cfg_resolution	1	Configuration control for DSI control mode: 0=12-bit resolution 1=use globally configured alternate resolution (8 or 10 bit)
dsi_cfg_differential	1	Configuration control for DSI control mode: 0= single-ended, 1=differential
dsi_trigger	1	Trigger to start SARSEQ scanning all enabled channels
dsi_data_hilo_sel	1	Selects between high and low byte output for sar_dsi_data[7:0]. This signal is fully asynchronous (affects sar_dsi_data without any clock involved).

21.3.11.1 Firmware Analog Routing

In DSI mode, analog routing can be implemented by DSI signals and firmware. Firmware control is always available regardless of the register configuration and it is the same as in register mode. See 21.3.2.1 Analog Routing for firmware control details.

21.3.11.2 DSI Analog Routing

DSI signals from UDB block are used to control SARMUX switches. In DSI control mode, the SARSEQ does not output any switch enables from the sequencer. Figure 21-4 shows that DSI can control every switch, except the DFT (design for test) switch. Thus, negative and positive input of SAR ADC can be connected to any switches in DSI mode.

Besides the DSI signals, appropriate hardware and firmware control bits in registers should be set. These registers and signals include SAR_MUX_SWITCH0 [n] = 1 and SAR_MUX_SWITCH_HW_CTRL[n] = 1. When V_{REF} is connected to the negative input, set SAR_CTRL [11:9] = 7 (firmware control field) and SAR_CTRL [13] = 1 (hardware control bit) except DSI signals.

DSI signals have control over the negative terminal of SAR ADC through dsi_swctrl[0] and dsi_sw_neg v_{REF} for single-ended mode. If NEG_SEL (SAR_CTRL[11:9]) is set, only NEG_SEL=7 is useful; the other value is ignored.

Table 21-14 shows the DSI signals.

Table 21-14. DSI Analog Routing

Signal	Width	Description
dsi_out	8	dsi_out[0]=1, P3.0 connected to ADC dsi_out[1]=1, P3.1 connected to ADC … dsi_out[7]=1, P3.7 connected to ADC **Note** Whether the pin is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, sarbus0 output connected to ADC dsi_oe[3]=1, sarbus1 output connected to ADC **Note** Whether the signal is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_swctrl[0]	1	SARMUX analog switch control, connect V_{SSA} to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temperature sensor to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input

21.3.11.3 Global SARSEQ Configuration

Global configuration applies to both register mode and DSI control mode. See 21.3.10.2 Global SARSEQ Configuration for details.

21.3.11.4 DSI Channel Configuration

For DSI control mode, only channel 0 is available. The channel 0 configuration can be done with DSI signals, as shown in Table 21-15. CHAN_EN and channel configurations in CHAN_CONFIG and INJ_CHAN_CONFIG are ignored.

The dsi_cfg_* signals can optionally be synchronized to the SAR clock domain (actually clk_hf) by setting

DSI_SYNC_CONFIG. Bypassing synchronization may be required when running the SAR at a low frequency.

Table 21-15. DSI Channel Configuration

Signal	Width	Configuration	Description
dsi_cfg_st_sel	2	Acquisition time	Configuration control for DSI control mode: select 1 of 4 global sample times
dsi_cfg_average	1	Average enable	Configuration control for DSI control mode: enable averaging
dsi_cfg_resolution	1	Resolution	Configuration control for DSI control mode: 0: 12-bit resolution 1: use globally configure resolution bit SUB_RESOLUTION (8 or 10 bit)
dsi_cfg_differential	1	Differential/single-ended	Configuration control for DSI control mode: 0: single-ended 1: differential

21.3.11.5 Interrupt

For an introduction to the SAR ADC interrupt, see Interrupt Masks on page 223. All interrupt masks work normally in register control mode. Not all interrupts are sent on DSI; SATURATE_INTR, RANGE_INTR, and EOS_INTR are sent via the DSI signal.

- Along with the data, SATURATE_INTR is output on dsi_chan_id[0]; SATURATE_INTR[0] is set in DSI control mode because only channel 0 is valid in DSI mode.
- Along with the data, RANGE_INTR is output on dsi_chan_id[1]; RANGE _INTR[0] is set in DSI control mode because only channel 0 is valid in DSI mode.
- Channel enables are ignored; this means only one conversion is done per trigger. An EOS_INTR is generated for each conversion.
- EOS_INTR is always sent via the DSI signal sar_dsi_eos_intr (a copy of dsi_data_valid).

Table 21-16 lists the interrupts that are sent via DSI signals.

Table 21-16. DSI Signal Interrupts

Signal	Width	Description
sar_dsi_chan_id	4	Register mode: Channel ID (ID of the channel that is currently being converted) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_eos_intr	1	End-of-scan interrupt to indicate that the SARSEQ has finished a scan of all enabled channels

21.3.11.6 Trigger

Typically, DSI control mode is used along with the DSI trigger. However, other trigger sources, such as firmware trigger and continuous trigger are also supported. The trigger configuration is the same as in the register control mode. See Trigger on page 217 for details.

For DSI trigger, the configuration settings (dsi_cfg_*) and switch settings should be stable no later than the cycle in which the dsi_trigger is sent. They should remain stable until the positive edge of the sar_dsi_sample_done.

21.3.11.7 Retrieve Data

The result data and channel number are sent out on sar_dsi_data. It is equivalent to dsi_out_en high in register control mode. See 21.3.11.8 DSI Output Enable for details. After each conversion, the data is also written to both CHAN_WORK0 and CHAN_RESULT0 registers.

21.3.11.8 DSI Output Enable

If the DSI_OUT_EN bit (SAR_CHANx_CONFIG[31]) is set, the result data and channel number are also sent out on the DSI bus (sar_dsi_data, sar_dsi_chan_id), next to being stored in the regular result register. This allows for the SARSEQ result data to be processed by the UDBs and the channel number allows for the possibility to apply different processing to data of different channels.

The data sent out on the DSI bus is formatted in the same way it is stored in the result register. However, by default only the 12 LSBs are sent out; it is not recommended to use left alignment unless more than 12 bits are required. To get the upper eight LSBs, the dsi_data_hilo_sel input needs to be set to ‘1’. To get the full 16-bit data from result register, first set dsi_data_hilo_sel = 0 to get the lower 12-bit data and then set dsi_data_hilo_sel = 1 to get the upper 8-bit data. Additional data process is needed to deal with the data overlap.

The channel number (sar_dsi_chan_id) will be sent out earlier, after the SAR ADC has completed sampling that channel. The channel number by itself can trigger the UDBs to drive some GPIO pins, which in turn can power up (or down) some off-chip device. This drives an analog input pin that will be scanned by one of the subsequent channels in the same scan (a long sample time is useful here).

Note that the data is sent out one cycle after the conversion is completed. Channel numbers, data, and their respective valid signals are maintained for two system clock cycles on the DSI bus.

Table 21-17. DSI Output Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging if there is) for one channel. The internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0] If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_data_hilo_sel	1	Selects between high and low byte output for sar_dsi_data[7:0]. This signal is fully asynchronous (affects sar_dsi_data without any clock involved)

21.3.12 Analog Routing Configuration Example

Table 21-18 shows some examples of pin and signal selection for sequencer control, firmware control, and DSI control.

Table 21-18. Analog Routing Configuration Example

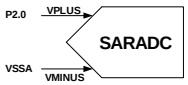
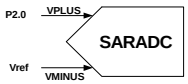
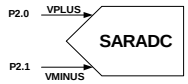
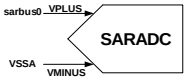
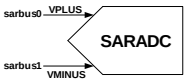
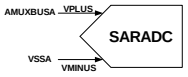
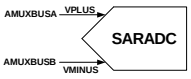
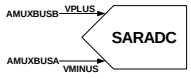
	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_out [0] =1 dsi_swctrl[0]=1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 7 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0]=1 HW_CTRL_NEGVREF =1 (CTRL[13])	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] =0 NEG_SEL = 7 (CTRL [11:9]) HW_CTRL_NEGVREF =0 (CTRL[13])	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 dsi_out [0] =1 dsi_sw_negvref =1 HW_CTRL_NEGVREF =1 (CTRL[13])
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 or PIN_ADDR = 1 (CHANx_CONFIG[2:0]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[1] = 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[1] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_out [0] =1 dsi_out [1] =1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH0 [9] = 1 MUX_SWITCH_HW_CTRL[1]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 1 (CHANx_CONFIG[6:4]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] =1 MUX_SWITCH_HW_CTRL[16] =1 **Note** Connecting sarbus1 to VPLUS is not supported for Port/Pin control	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [2] =1 dsi_swctrl[0]=1 MUX_SWITCH0 [16] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH_HW_CTRL[22] =1

Table 21-18. Analog Routing Configuration Example<Italic> (continued)

	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 1 **(**CHANx_CONFIG[6:4]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[23] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [2] = 1 dsi_oe [3] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** **NEG_SEL = 0** (CTRL [11:9]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[16]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [0] = 1 dsi_swctrl[0]=1 MUX_SWITCH0[18] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[19]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1
	Not supported. The differential pair is fixed for Port/Pin control	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =0 MUX_SWITCH_HW_CTRL[19] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =1 MUX_SWITCH_HW_CTRL[19] = 1

21.3.13 Temperature Sensor Configuration

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Differential conversions are not available for temperature sensors (conversion result is undefined). Therefore, always use it in single-ended mode. The reference is from internal 1.024 V.

A pin or signal can be routed to the SAR ADC in three ways. Table 21-19 lists the methods to route temperature sensors to SAR ADC. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit disables temperature sensor by cutting its bias current.

Table 21-19. Route Temperature to SAR ADC

Control Methods	Setup
Sequencer	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4]) PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0]) SWITCH_DISABLE = 0 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
Firmware	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) SWITCH_DISABLE = 1 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 0 SAR_MUX_SWITCH_HW_CTRL[17]= 0 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
DSI	SWITCH_DISABLE = 1 (SAR_CTRL[30]) VREF_SEL = 0 (SAR_CTRL[6:4]) Set DSI Signals: dsi_cfg_differential=1 dsi_swctrl[1]=1 dsi_swctrl[0]=1 SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]

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

21.4 Registers

Name	Offset	Qty.	Width	Description
SAR_CTRL	0x0000	1	32	Global configuration register Analog control register
SAR_SAMPLE_CTRL	0x0004	1	32	Global configuration register Sample control register
SAR_SAMPLE_TIME01	0x0010	1	32	Global configuration register Sample time specification ST0 and ST1
SAR_SAMPLE_TIME23	0x0014	1	32	Global configuration register Sample time specification ST2 and ST3
SAR_RANGE_THRES	0x0018	1	32	Global range detect threshold register
SAR_RANGE_COND	0x001C	1	32	Global range detect mode register
SAR_CHAN_EN	0x0020	1	32	Enable bits for the channels
SAR_START_CTRL	0x0024	1	32	Start control register (firmware trigger)
SAR_CHAN_CONFIG	0x0080	8	32	Channel configuration register
SAR_CHAN_WORK	0x0100	8	32	Channel working data register
SAR_CHAN_RESULT	0x0180	8	32	Channel result data register
SAR_CHAN_WORK_VALID	0x0200	1	32	Channel working data register valid bits
SAR_CHAN_RESULT_VALID	0x0204	1	32	Channel result data register valid bits
SAR_STATUS	0x0208	1	32	Current status of internal SAR registers (for debug)
SAR_AVG_STAT	0x020C	1	32	Current averaging status (for debug)
SAR_INTR	0x0210	1	32	Interrupt request register
SAR_INTR_SET	0x0214	1	32	Interrupt set request register
SAR_INTR_MASK	0x0218	1	32	Interrupt mask register
SAR_INTR_MASKED	0x021C	1	32	Interrupt masked request register: If the value is not zero, then the SAR interrupt signal to the NVIC is high. When read, this register reflects a bit-wise AND between the interrupt request and mask registers
SAR_SATURATE_INTR	0x0220	1	32	Saturate interrupt request register
SAR_SATURATE_INTR_SET	0x0224	1	32	Saturate interrupt set request register
SAR_SATURATE_INTR_MASK	0x0228	1	32	Saturate interrupt mask register
SAR_SATURATE_INTR_MASKED	0x022C	1	32	Saturate interrupt masked request register
SAR_RANGE_INTR	0x0230	1	32	Range detect interrupt request register
SAR_RANGE_INTR_SET	0x0234	1	32	Range detect interrupt set request register
SAR_RANGE_INTR_MASK	0x0238	1	32	Range detect interrupt mask register
SAR_RANGE_INTR_MASKED	0x023C	1	32	Range interrupt masked request register
SASR_INTR_CAUSE	0x0240	1	32	Interrupt cause register
SAR_INJ_CHAN_CONFIG	0x0280	1	32	Injection channel configuration register
SAR_INJ_RESULT	0x0290	1	32	Injection channel result register
SAR_MUX_SWITCH0	0x0300	1	32	SARMUX firmware switch controls
SAR_MUX_SWITCH_CLEAR0	0x0304	1	32	SARMUX firmware switch control clear
SAR_MUX_SWITCH_HW_CTRL	0x0340	1	32	SARMUX switch hardware control
SAR_MUX_SWITCH_STATUS	0x0348	1	32	SARMUX switch status
SAR_PUMP_CTRL	0x0380	1	32	Switch pump control

22. LCD Direct Drive

The PRoC Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PRoC device to directly drive STN and TN segment LCDs.

22.1 Features

The PRoC LCD segment drive block has the following features:

- Supports up to 32 segments and four commons (mux ratio 1:4)
- Supports Type A (standard) and Type B (low-power) drive waveforms
- Any GPIO can be configured as a common or segment
- Supports three drive methods:
  - Digital correlation
  - PWM at 1/2 bias
  - PWM at 1/3 bias
- Ability to drive 3-V displays from 1.8 V V_{DD} in Digital Correlation mode
- Operates in active, sleep, and deep-sleep modes
- Digital contrast control

22.2 LCD Segment Drive Overview

A segmented LCD panel has the liquid crystal material between two sets of electrodes and various polarization and reflector layers. The two electrodes of an individual segment are called commons (COM) or backplanes and segment electrodes (SEG). From an electrical perspective, an LCD segment can be considered as a capacitive load; the COM/SEG electrodes can be considered as the rows and columns in a matrix of segments. The opacity of an LCD segment is controlled by varying the root-mean-square (RMS) voltage across the corresponding COM/SEG pair.

The following terms/voltages are used in this chapter to describe LCD drive:

- **V_{RMSOFF}**: The voltage that the LCD driver can realize on segments that are intended to be off.
- **V_{RMSON}**: The voltage that the LCD driver can realize on segments that are intended to be on.
- **Discrimination Ratio (D)**: The ratio of V_{RMSON} and V_{RMSOFF} that the LCD driver can realize. This depends on the type of waveforms applied to the LCD panel. Higher discrimination ratio results in higher contrast.

Liquid crystal material does not tolerate long term exposure to DC voltage. Therefore, any waveforms applied to the panel must produce a 0-V DC component on every segment (on or off). Typically, LCD drivers apply waveforms to the COM and SEG electrodes that are generated by switching between multiple voltages. The following terms are used to define these waveforms:

- **Duty**: A driver is said to operate in 1/M duty when it drives 'M' number of COM electrodes. Each COM electrode is effectively driven 1/M of the time. PRoC supports 1/2, 1/3, and 1/4 duties.
- **Bias**: A driver is said to use 1/B bias when its waveforms use voltage steps of (1/B) × VDRV. VDRV is the highest drive voltage in the system (equals to V_{DD} in PRoC). PRoC supports 1/2 and 1/3 biases in PWM drive modes.
- **Frame**: A frame is the length of time required to drive all the segments. During a frame, the driver cycles through the commons in sequence. All segments receive 0-V DC (but non-zero RMS voltage) when measured over the entire frame.

PRoC supports two different types of drive waveforms in all drive modes. These are:

- **Type-A Waveform**: In this type of waveform, the driver structures a frame into M sub-frames. 'M' is the number of COM electrodes. Each COM is addressed only once during a frame. For example, COM[i] is addressed in sub-frame i.
- **Type-B Waveform**: The driver structures a frame into 2M sub-frames. The two sub-frames are inverses of each other. Each COM is addressed twice during a frame. For example, COM[i] is addressed in sub-frames i and M+i. Type-B waveforms are slightly more power efficient because it contains fewer transitions per frame.

22.2.1 Drive Modes

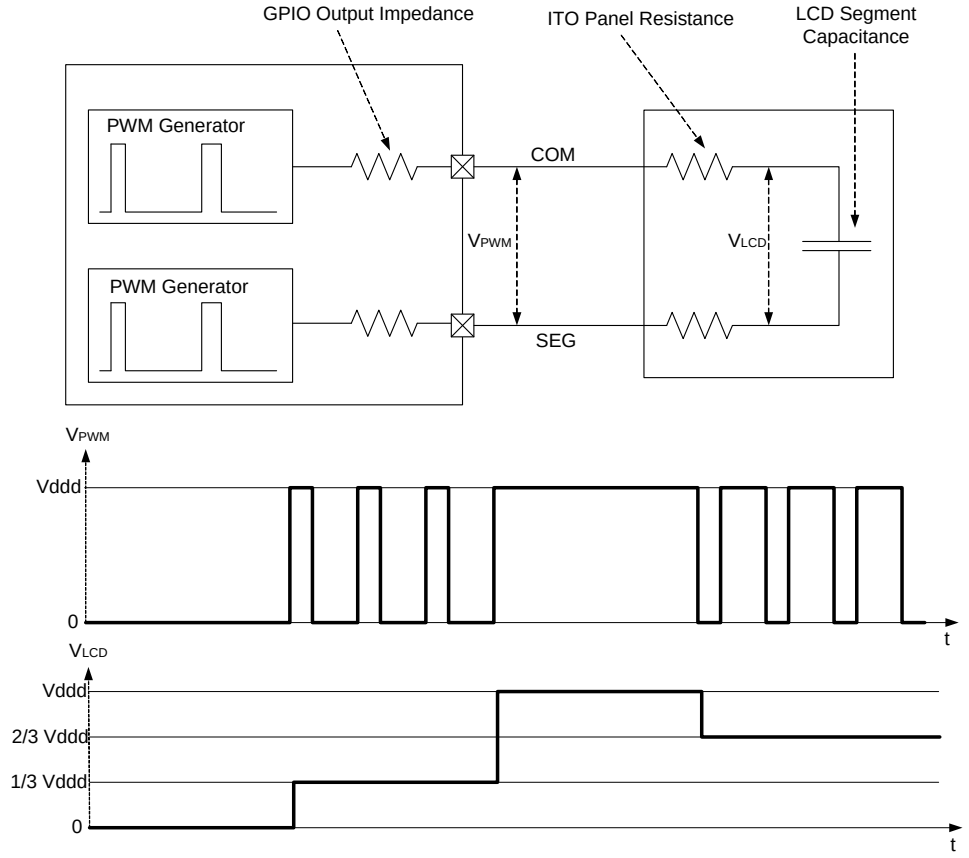
PRoC supports the following drive modes.

- PWM Drive at 1/2 bias
- PWM Drive at 1/3 bias
- Digital correlation

22.2.1.1 PWM Drive

In PWM drive mode, multi-voltage drive signals are generated using a PWM output signal together with the intrinsic resistance and capacitance of the LCD. Figure 22-1 illustrates this.

Figure 22-1. PWM Drive (at 1/3 Bias)



The output waveform of the drive electronics is a PWM waveform. With the Indium Tin Oxide (ITO) panel resistance and the segment capacitance to filter the PWM, the voltage across the LCD segment is an analog voltage, as shown in Figure 22-1. This figure illustrates the generation of a 1/3 bias waveform (four commons and voltage steps of $V_{DD}/3$).

The PWM is derived from either ILO (32 kHz) or IMO. The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 22-2 and Figure 22-3 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/2 bias and 1/4 duty. Only COM0/COM1 and SEG0/SEG1 are drawn for demonstration purpose. Similarly, Figure 22-4 and Figure 22-5 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/3 bias and 1/4 duty.

Figure 22-2. PWM1/2 Type-A Waveform Example

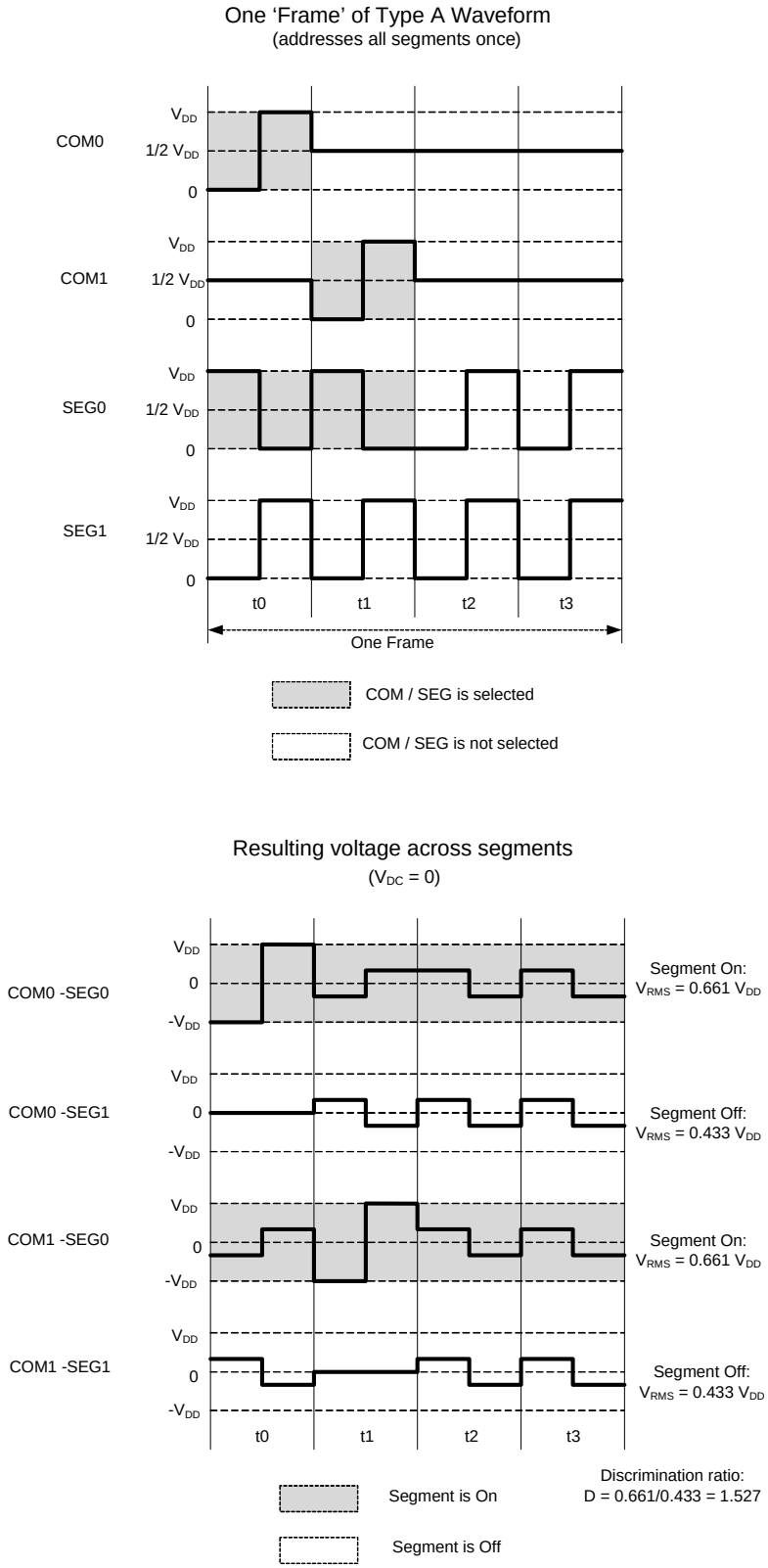


Figure 22-3. PWM1/2 Type-B Waveform Example

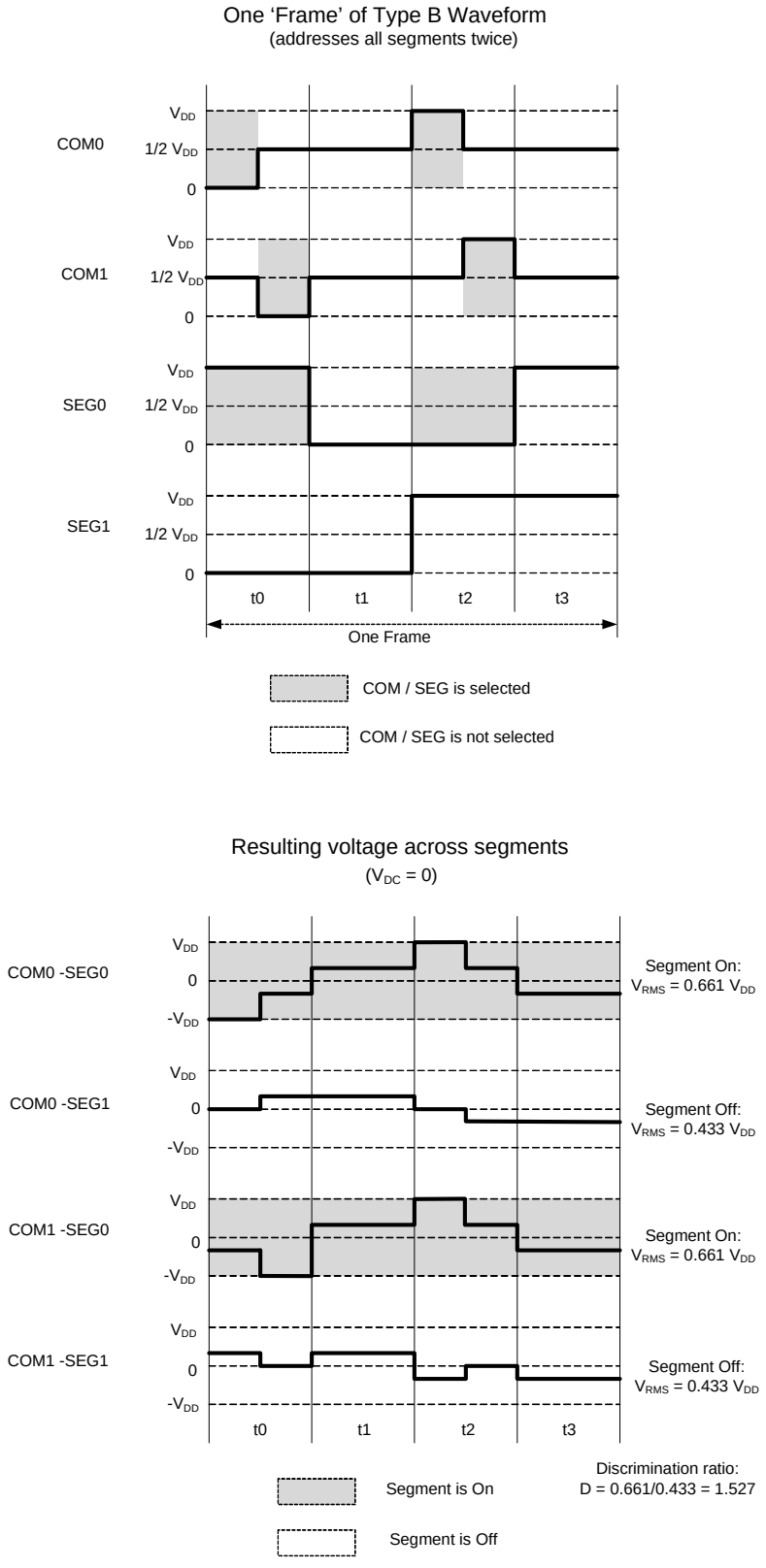


Figure 22-4. PWM1/3 Type-A Waveform Example

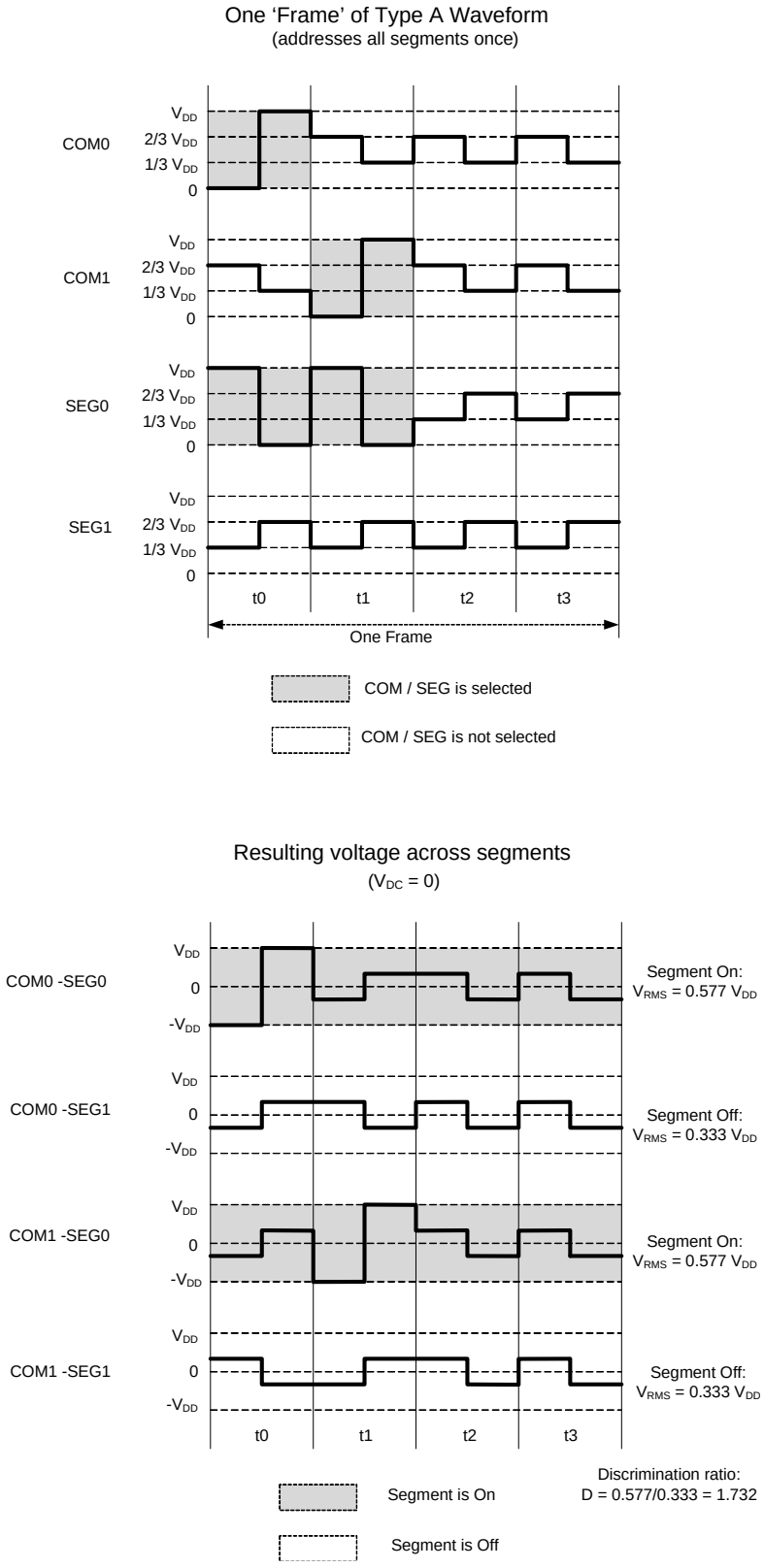
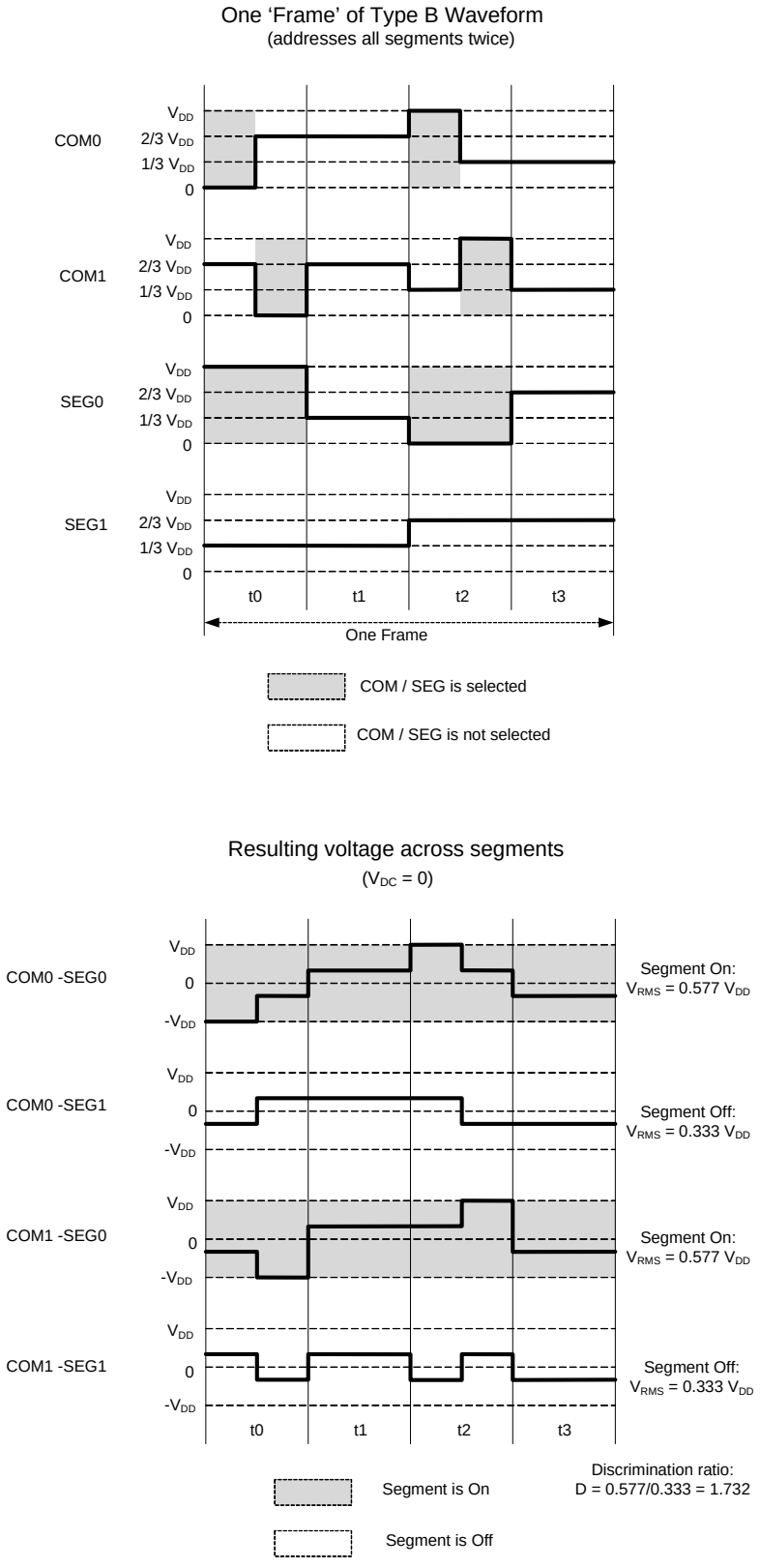


Figure 22-5. PWM1/3 Type-B Waveform Example



The effective RMS voltage for ON and OFF segments can be calculated easily using these equations:

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

Equation 22-1

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

Equation 22-2

Where B is the bias and M is the duty (number of COMs).

For example, if the number of COMs is 4, the resulting discrimination ratios (D) for 1/2 and 1/3 biases are 1.528 and 1.732, respectively. 1/3 bias offers better discrimination ratio in 2 and 3 COM drives also. Therefore, 1/3 bias offers better contrast than 1/2 bias and is recommended for most applications.

When the low-speed operation of LCD is used, the PWM signal is derived from the 32-kHz ILO. To drive a low-capacitance display with acceptable ripple and rise/fall times using a 32-kHz PWM, additional external series resistances of 100 k-1 MΩ should be used. External resistors are not required for PWM frequencies greater than ~1 MHz. The ideal PWM frequency depends on the capacitance of the display and the internal ITO resistance of the ITO routing traces.

The 1/2 bias mode has the advantage that PWM is only required on the COM signals; the SEG signals use only logic levels, as shown in Figure 22-2 and Figure 22-3.

22.2.1.2 Digital Correlation

The digital correlation mode, instead of generating bias voltages between the rails, takes advantage of the characteristic of LCDs that the contrast of LCD segments is determined by the RMS voltage across the segments. In this approach, the correlation coefficient between any given pair of COM and SEG signals determines whether the corresponding LCD segment is on or off. Thus, by doubling the base drive frequency of the COM signals in their inactive sub-frame intervals, the phase relationship of the COM and SEG drive signals can be varied to turn segments on and off. This is different from varying the DC levels of the signals as in the PWM drive approach. Figure 22-8 and Figure 22-9 are example waveforms that illustrate the principles of operation.

Figure 22-6. Digital Correlation Type-A Waveform

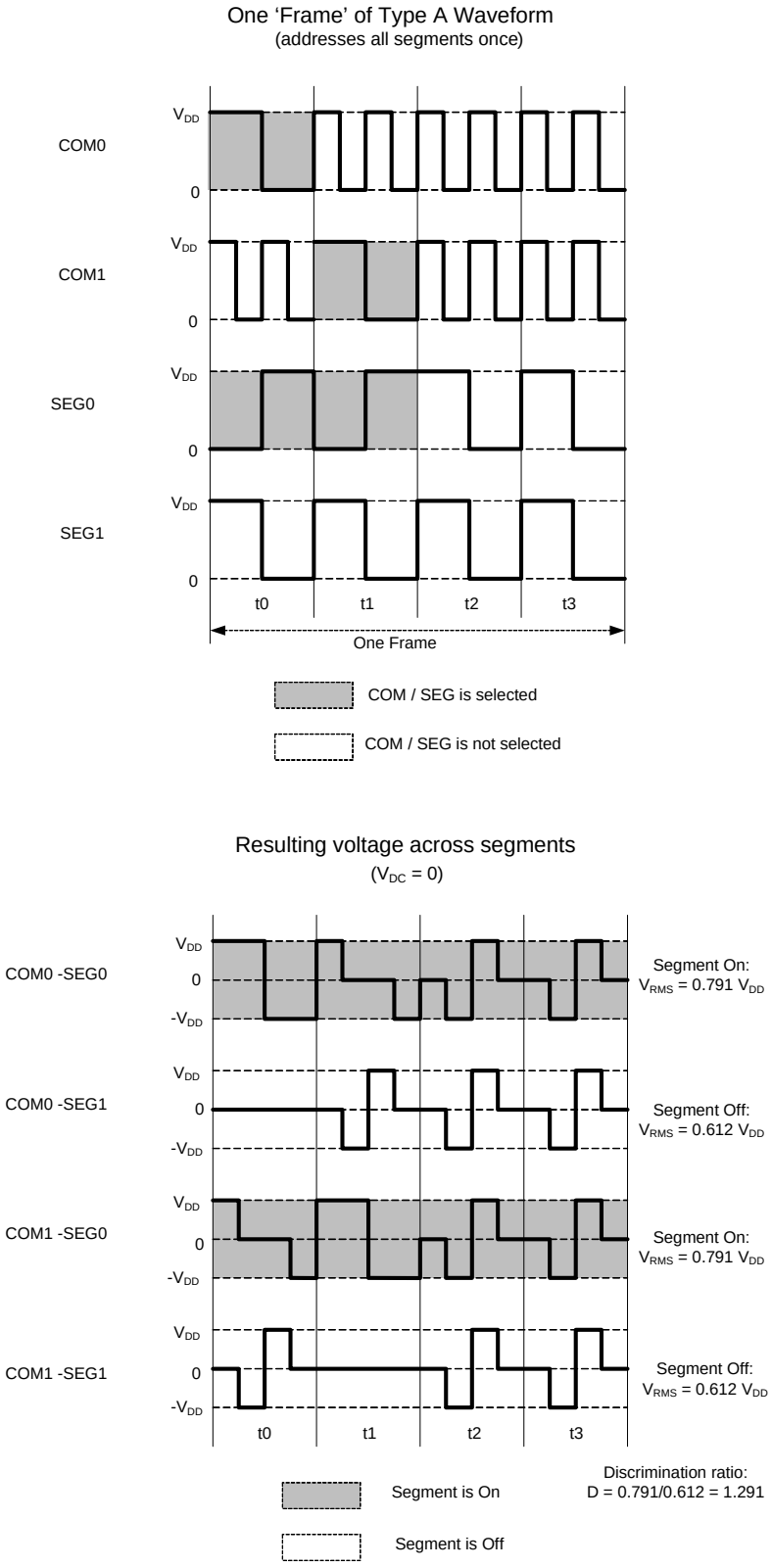
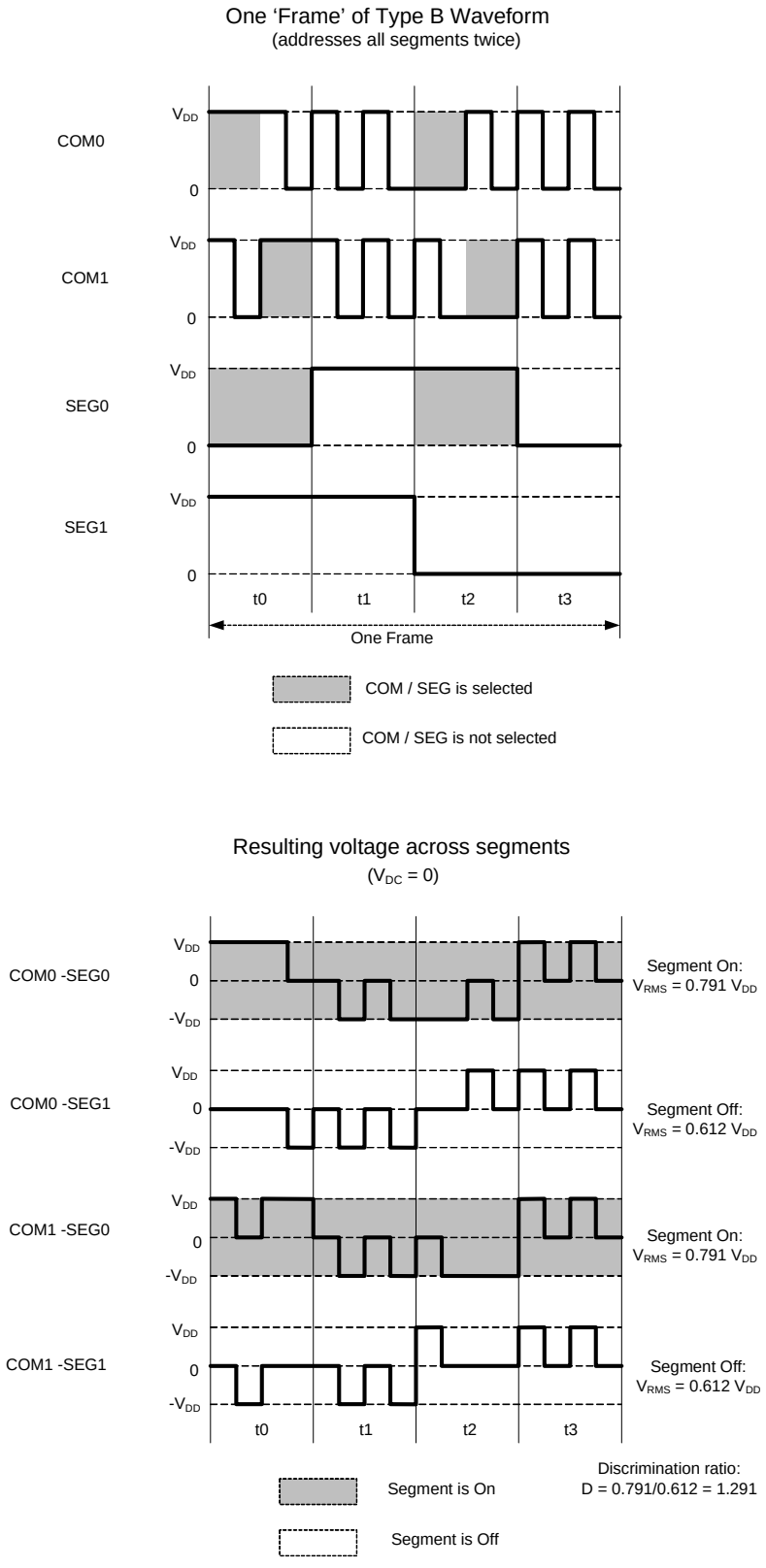


Figure 22-7. Digital Correlation Type-B Waveform



The RMS voltage applied to on and off segments can be calculated as follows:

$$V_{RMS(OFF)} = \sqrt{\frac{(M-1)}{2M}} x (V_{DD})$$

$$V_{RMS(ON)} = \sqrt{\frac{2+(M-1)}{2M}} x (V_{DD})$$

Where B is the bias and M is the duty (number of COMs). This leads to a discrimination ratio (D) of 1.291 for four COMs.

Digital correlation mode also has the ability to drive 3-V displays from 1.8 V V_{DD}.

22.2.2 Recommended Usage of Drive Modes

The PWM drive mode has higher discrimination ratios compared to the digital correlation mode, as explained in 22.2.1.1 PWM Drive and 22.2.1.2 Digital Correlation. Therefore, the contrast in digital correlation method is lower than PWM method but digital correlation has lower power consumption because its waveforms toggle at low frequencies.

The digital correlation mode creates reduced, but acceptable contrast on TN displays, but no noticeable difference in contrast or viewing angle on higher contrast STN displays.

Because each mode has strengths and weaknesses, recommended usage is as follows.

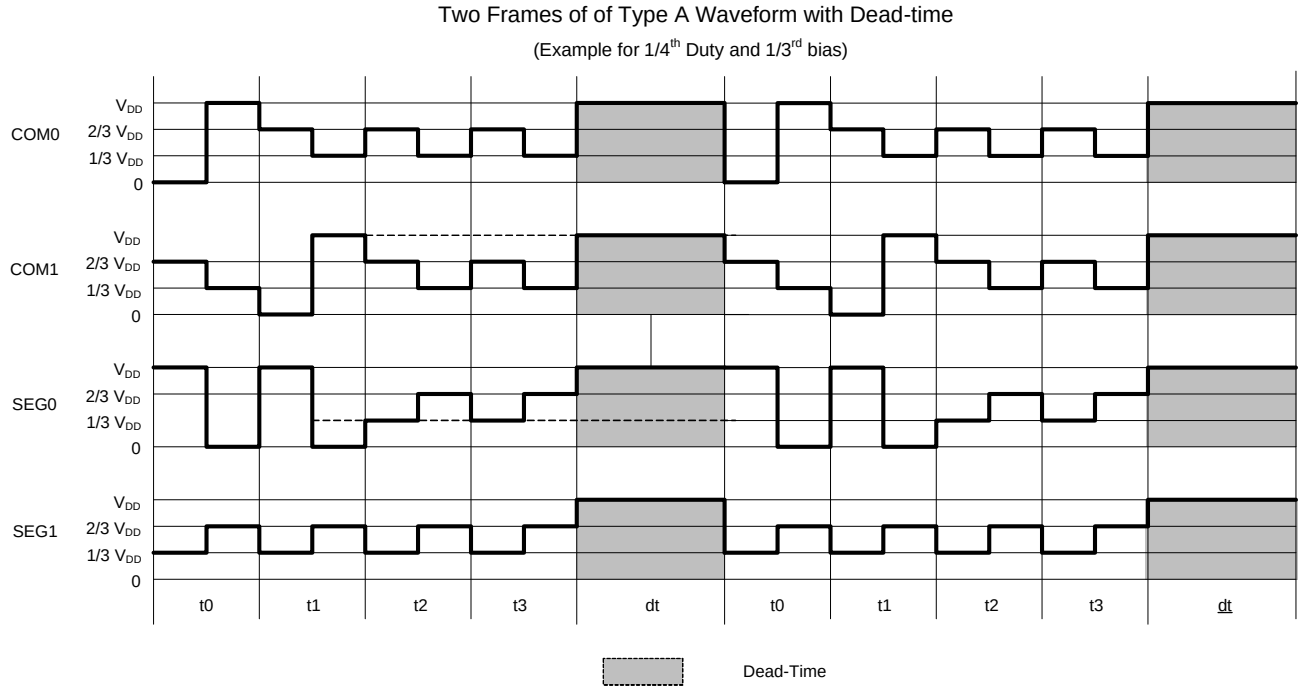
Table 22-1. Recommended Usage of Drive Modes

Display Type	Deep-Sleep Mode	Sleep/Active Mode	Notes
TN Glass	Digital Correlation	PWM 1/3 Bias	Firmware must switch between LCD drive modes before going to deep sleep or waking up.
STN Glass	Digital Correlation		No contrast advantage for PWM drive with STN glass.

22.2.3 Digital Contrast Control

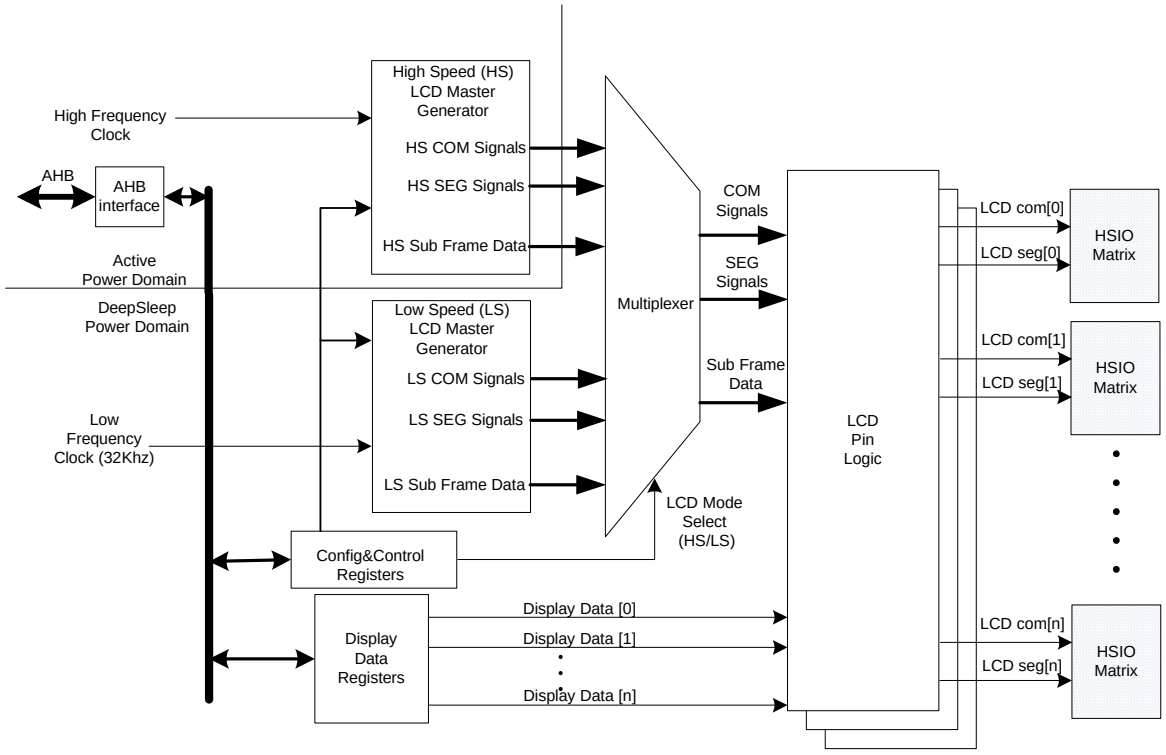
In all drive modes, digital contrast control can be used to change the contrast level of the segments. This method reduces contrast by reducing the driving time of the segments. This is done by inserting a ‘Dead-Time’ interval after each frame. During dead time, all COM and SEG signals are driven to a logic 1 state. The dead time can be controlled in fine resolution. Figure 22-8 illustrates the dead-time contrast control method for 1/3 bias and 1/4 duty implementation.

Figure 22-8. Dead-Time’ Contrast Control



22.3 Block Diagram

Figure 22-9. Block Diagram of LCD Direct Drive System



22.3.1 How it Works

The LCD controller block contains two generators; one with a high-speed clock source HFCLK and the other with a low-speed clock source (32 kHz) derived from the ILO. These are called high-speed LCD master generator and low-speed LCD master generator, respectively. Both the generators support PWM and digital correlation drive modes. PWM drive mode with low-speed generator requires external resistors, as explained in PWM Drive on page 234.

The multiplexer selects one of these two generator outputs to drive LCD, as configured by the firmware. The LCD pin logic block routes the COM and SEG outputs from the generators to the corresponding I/O matrices. Any GPIO can be used as either COM or SEG. This configurable pin assignment for COM or SEG is implemented in GPIO and I/O matrix; see High-Speed I/O Matrix on page 69. These two generators share the same configuration registers. These memory mapped I/O registers are connected to the system bus (AHB) using an AHB interface.

The LCD controller works in three device power modes: active, sleep, and deep-sleep. High-speed operation is supported in active and sleep modes. Low-speed operation is supported in active, sleep, and deep-sleep modes. The LCD controller is unpowered in hibernate and stop modes.

22.3.2 High-Speed and Low-Speed Master Generators

The high-speed and low-speed master generators are similar to each other. The only exception is that the high-speed version has larger frequency dividers to generate the frame and sub-frame periods. This is because the clock of the high-speed block (HFCLK) is derived from the IMO, which is typically at 30 to 100 times the frequency of the ILO (32 kHz) clock fed to the low-speed block. The high-speed generator is in the active power domain and the low-speed generator is in the deep-sleep power domain. A single set of configuration registers is provided to control both high-speed and low-speed blocks. Each master generator has the following features and characteristics:

- Register bit configuring the block for either Type A or Type B drive waveforms (LCD_MODE bit in LCD_CONTROL register).
- Register bits to select the number of COMs (COM_NUM field in LCD_CONTROL register). The available values are 2, 3, and 4.
- Operating mode configuration bits enabled to select one of the following:
  - Digital correlation
  - PWM 1/2 bias
  - PWM 1/3 bias

- Off/disabled. Typically, one of the two generators will be configured to be Off

OP_MODE and BIAS fields in LCD_CONTROL bits select the drive mode.

- A counter to generate the sub-frame timing. The SUBFR_DIV field in the LCD_DIVIDER register determines the duration of each sub-frame. If the divide value written into this counter is C, the sub-frame period is 4 × (C+1). The low-speed generator has an 8-bit counter. This generates a maximum half sub-frame period of 8 ms from the 32-kHz ILO clock. The high-speed generator has a 16-bit counter.
- A counter to generate the dead time period. These counters have the same number of bits as the sub-frame period counters and use the same clocks. DEAD_DIV field in the LCD_DIVIDER register controls the dead time period.

22.3.3 Multiplexer and LCD Pin Logic

The multiplexer selects the output signals of either high-speed or low-speed master generator blocks and feeds it to the LCD pin logic. This selection is controlled by the configuration and control register. The LCD pin logic uses the sub-frame signal from the multiplexer to choose the display data. This pin logic will be replicated for each LCD pin.

22.3.4 Display Data Registers

Each LCD segment pin is part of a LCD port with its own display data register, LCD_DATA0x. The device has eight such LCD ports. Note that these ports are not real pin ports but the ports/connections available in the LCD hardware for mapping the segments to commons. Each LCD segment configured is considered as a pin in these LCD ports. The LCD_DATA0x registers are 32-bit wide registers and store the ON/OFF data for all SEG-COM combination enabled in the design. The bits [4i+3:4i] (where 'i' is the pin number) of each LCD_DATA0x register represent the ON/OFF data for Pin[i] in Port[x] and COM[3,2,1,0] combinations, as shown in Table 22-2. The LCD_DATA0x register should be programmed according to the display data of each frame. The display data registers are Memory Mapped I/O (MMIO) and accessed through the AHB slave interface.

Table 22-2. SEG-COM Mapping in LCD_DATA0x Registers (each SEG is a pin of the LCD port)

BITS[31:28] = PIN_7[3:0]				**BITS[27:24] = PIN_6[3:0]**			
PIN_7-COM3	PIN_7-COM2	PIN_7-COM1	PIN_7-COM0	PIN_6-COM3	PIN_6-COM2	PIN_6-COM1	PIN_6-COM0
BITS[23:20] = PIN_5[3:0]				**BITS[19:16] = PIN_4[3:0]**			
PIN_5-COM3	PIN_5-COM2	PIN_5-COM1	PIN_5-COM0	PIN_4-COM3	PIN_4-COM2	PIN_4-COM1	PIN_4-COM0
BITS[15:12] = PIN_3[3:0]				**BITS[11:8] = PIN_2[3:0]**			
PIN_3-COM3	PIN_3-COM2	PIN_3-COM1	PIN_3-COM0	PIN_2-COM3	PIN_2-COM2	PIN_2-COM1	PIN_2-COM0
BITS[7:3] = PIN_1[3:0]				**BITS[3:0] = PIN_0[3:0]**			
PIN_1-COM3	PIN_1-COM2	PIN_1-COM1	PIN_1-COM0	PIN_0-COM3	PIN_0-COM2	PIN_0-COM1	PIN_0-COM0

22.4 Register List

Table 22-3. LCD Direct Drive Register List

Register Name	Description
LCD_ID	This register includes the information of LCD controller' ID and revision number
LCD_DIVIDER	This register controls the sub-frame and dead-time period
LCD_CONTROL	This register is used to configure high-speed and low-speed generators
LCD_DATA0x	LCD port pin data register for COM0 to COM3; x = port number, 8 ports are available

23. CapSense

PRoC uses a capacitive touch sensing method known as CapSense® Sigma Delta (CSD). The CapSense Sigma Delta touch sensing method provides the industry's best-in-class signal-to-noise ratio (SNR). CSD is a combination of hardware and firmware techniques. This chapter explains how the CSD hardware is implemented in PRoC.

See the PSoC 4 CapSense Design Guide for more details on the basics of CSD operation, available CapSense design tools, the easy-to-use PSoC Creator™ component, performance tuning using the tuner GUI, and PCB layout design considerations.

23.1 Features

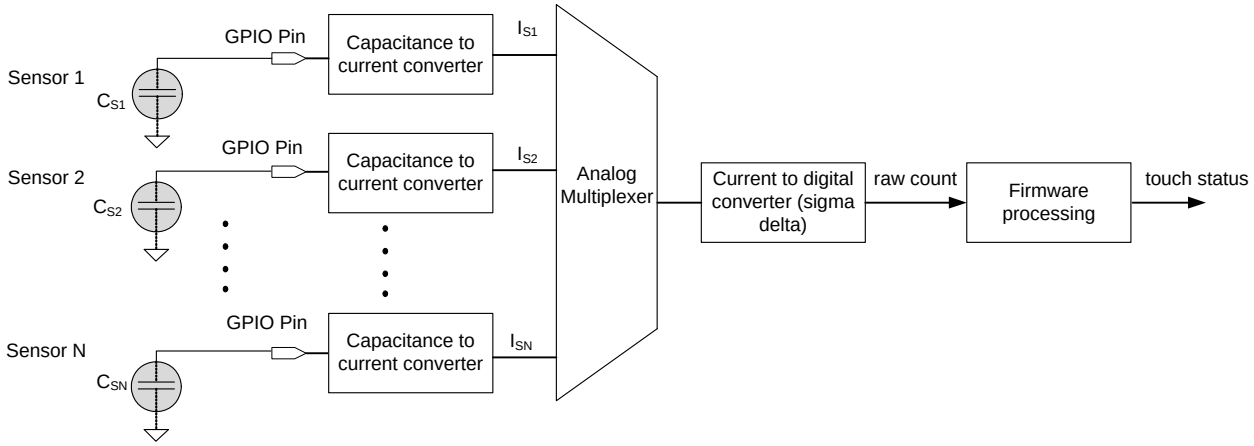
PRoC CapSense has the following features:

- Robust sensing technology
- CSD operation provides best-in-class SNR
- High-performance sensing across a variety of overlay materials and thicknesses
- SmartSense™ auto-tuning technology
- Supports as many as 35 sensors
- High-range proximity sensing
- Water tolerant operation using shield signal, available on all GPIOs
- Low power consumption
- Two IDAC operation for improved scan speed and SNR
- Any GPIO pin can be used for sensing or shielding
- Pseudo random sequence (PRS) clock source for lower electromagnetic interference (EMI)
- Dedicated charge tank capacitor for quick charge transfer on to shield lines
- GPIO cell precharge support to quickly initialize external tank capacitors

23.2 Block Diagram

Figure 23-1 shows the CSD system block diagram.

Figure 23-1. CapSense Module Block Diagram



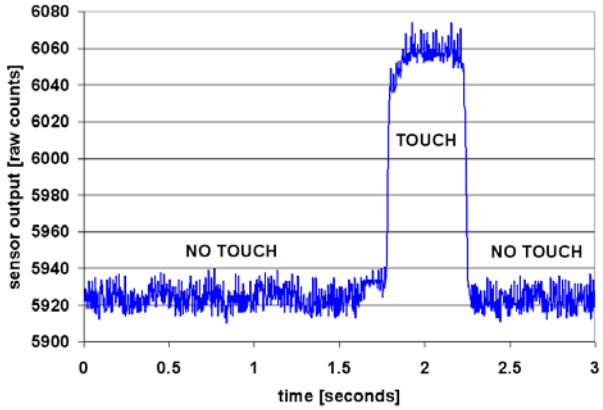
23.3 How It Works

With CSD, each GPIO has a switched capacitance circuit that converts the sensor capacitance into an equivalent current. An analog multiplexer then selects one of the currents and feeds it into the current-to-digital converter. The current-to-digital converter is similar to a sigma delta ADC.

The output count of the current-to-digital converter, known as raw count, is a digital value that is proportional to the sensor capacitance.

Figure 23-2 shows a plot of raw count over time. When a finger touches the sensor, the sensor capacitance increases; the raw count increases proportionally. By comparing the change in raw count to a predetermined threshold, logic in firmware can decide whether the sensor is active (finger is present).

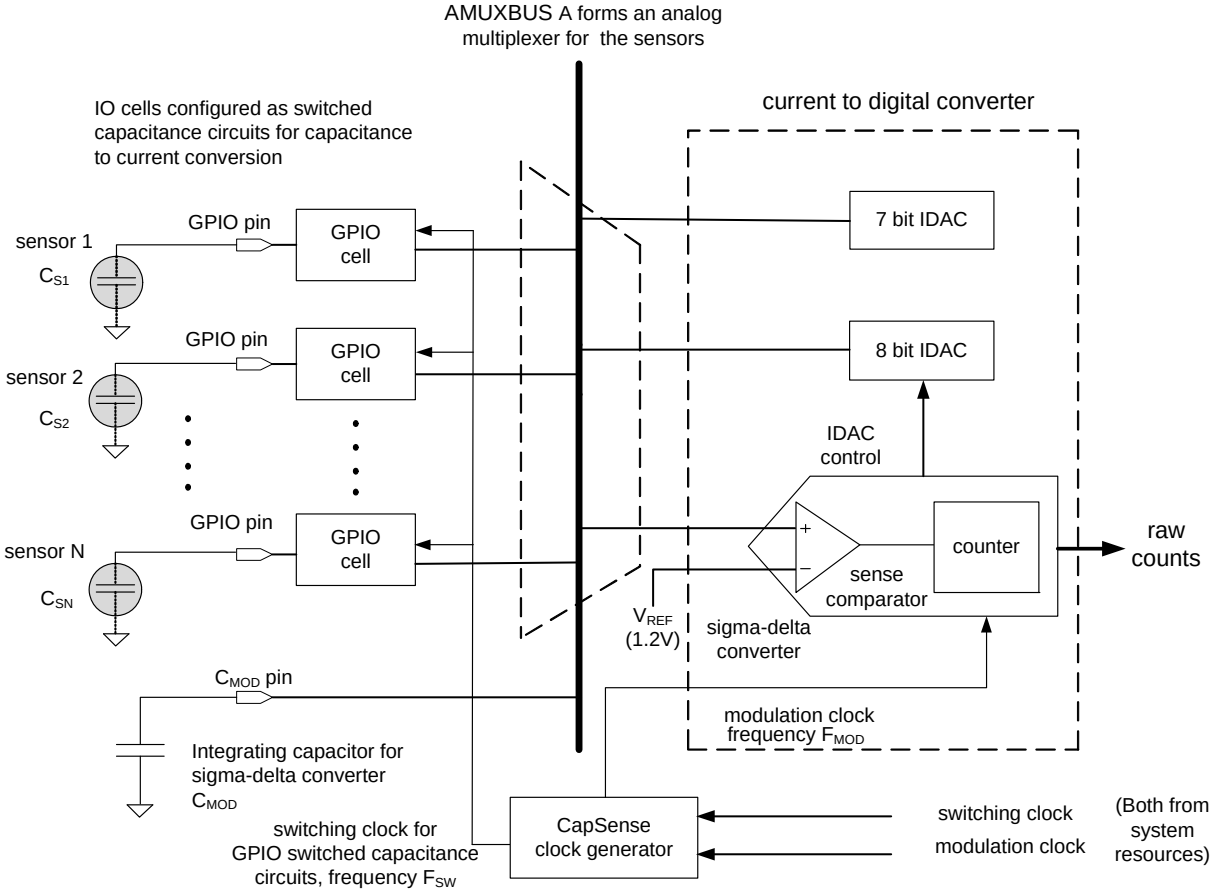
Figure 23-2. Raw Count Versus Time



23.4 CapSense CSD Sensing

Figure 23-3 shows the block diagram of the PRoC CapSense hardware.

Figure 23-3. PRoC CapSense CSD Sensing



23.4.1 GPIO Cell Capacitance to Current Converter

In the CapSense CSD system, the GPIO cells are configured as switched capacitance circuits, which convert the sensor capacitance to equivalent currents. Figure 23-4 shows a simplified diagram of the PRoC GPIO cell structure.

PRoC has two analog multiplexer buses: AMUXBUS A is used for CSD sensing and AMUXBUS B is used for CSD shielding. The GPIO switched capacitance circuit has two possible configurations: source current to AMUXBUS A or sink current from AMUXBUS A. Figure 23-5 shows the switched capacitance configuration for sourcing current to AMUXBUS A.

Figure 23-4. PRoC GPIO Cell

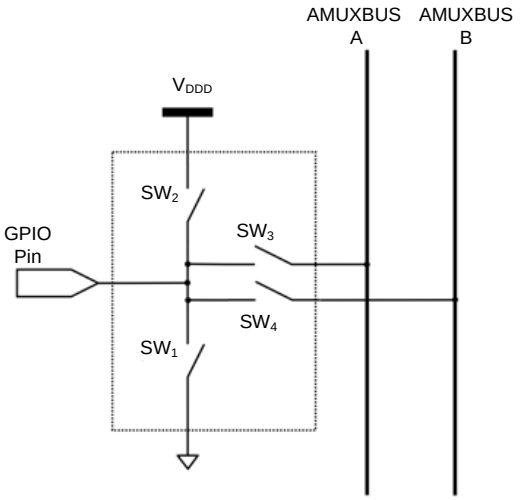
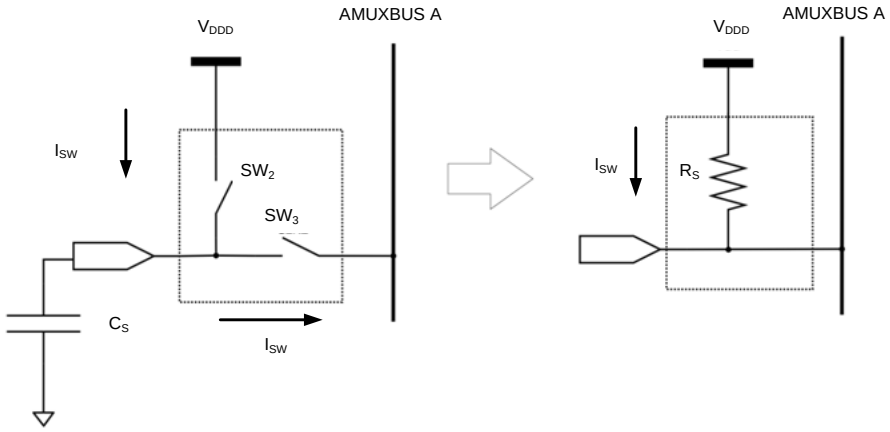


Figure 23-5. Sourcing Current to AMUXBUS A



Two non-overlapping, out of phase clocks of frequency F_{SW} (see Figure 23-3) control the switches SW_2 and SW_3. The continuous switching of SW_2 and SW_3 forms an equivalent resistance R_S, as Figure 23-5 shows. The value of the equivalent resistance R_S is:

$$R_S = \frac{1}{C_S F_{SW}}$$

Equation 23-1

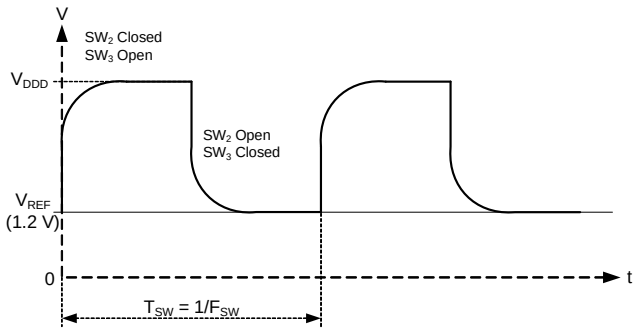
Where:

C_S = Sensor capacitance

F_{SW} = Frequency of the switching clock

The sigma delta converter maintains the voltage of AMUX-BUS A at a constant V_{REF} (this process is explained in Sigma Delta Converter on page 249). Figure 23-6 shows the voltage waveform across the sensor capacitance.

Figure 23-6. Voltage Across Sensor Capacitance



Equation 23-3 gives the value of average current supplied to AMUXBUS A.

$$I_S = C_S F_{SW}(V_{DDD} - V_{REF})$$

Equation 23-2

Figure 23-7 shows the switched capacitance configuration for sinking current from AMUXBUS A. Figure 23-8 shows the resulting voltage waveform across C_S.

Figure 23-7. Sinking Current From AMUXBUS A

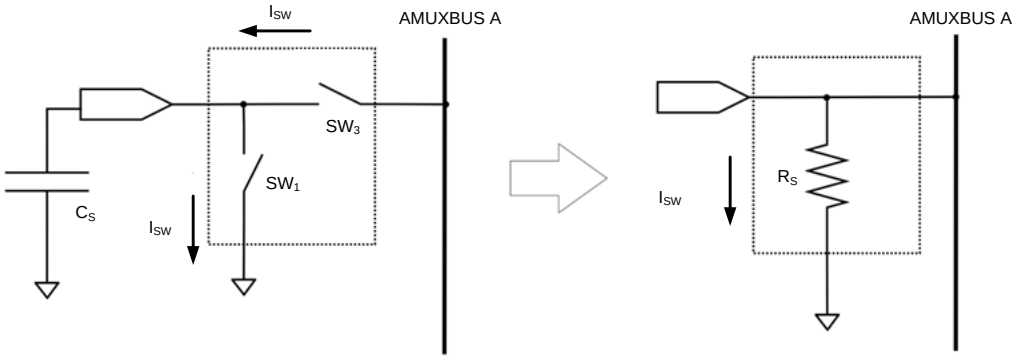
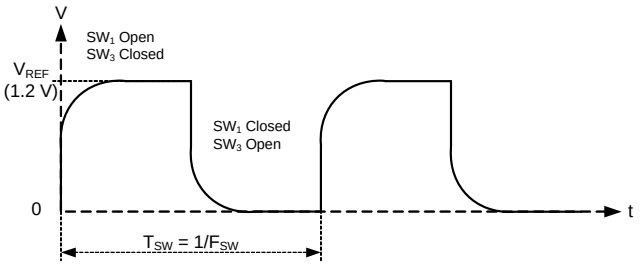


Figure 23-8. Voltage Across Sensor Capacitance



Equation 23-4 gives the value of average current taken from AMUXBUS A.

$$I_S = C_S F_{SW} V_{REF}$$

Equation 23-3

The sigma delta converter scans one sensor at a time. AMUXBUS A is used to select one of the GPIO cells and connects it to the input of the sigma delta converter, as Figure 23-3 shows. The AMUXBUS A and the GPIO cell switches (see SW_3 in Figure 23-4) form this analog multiplexer. AMUXBUS A can connect toall PRoC pins that support CSD. See the device datasheet to know the CSD capable pins.

See the I/O System chapter on page 65 to know how to configure a GPIO cell for sensing, shielding, and connecting C_{MOD}.

23.4.2 CapSense Clock Generator

This block, together with the peripheral clock from the system resources, generates the switching clock F_{SW} and the modulation clock F_{MOD}, as Figure 23-3 shows. For details, see the Clocking System chapter on page 73.

The switching clock is required for the GPIO cell switched capacitance circuits. The sigma delta converter uses the modulation clock for timing.

Two peripheral clocks CSDCLK0 and CSDCLK1 from the system resources can be used to generate the required frequencies. See the Clocking System chapter on page 73 for details. CSDCLK0 generates modulation clock and CSDCLK1 generates switching clock.

However, the final switching clock frequency depends on the CapSense clock generator. It has the following output options:

- Direct: Uses the output of programmable clock dividers directly. To select this option, set the BYPASS_SEL bit in the CSD_CONFIG register ‘1’.
- Divide by 2. Divides the clocks by two. To select this option, clear the PRS_SELECT and BYPASS_SEL bits in the CSD_CONFIG register.
- Pseudo random sequence (PRS): Reduces the EMI in the CapSense system by spreading the switching frequency over a broader range. To select this option, set the PRS_SELECT bit and clear the BYPASS_SEL bit in the CSD_CONFIG register. You can select between 8- and 12- bit pseudo random sequence using the PRS_12_8 bit in the same register. Set this bit to select a 12- bit sequence; clear it for 8- bit PRS.

If PRS is selected, the maximum switching frequency is

$$F_{SW(maximum)} = \frac{F_{in}}{2}$$

Equation 23-4

Where F_{in} is the frequency output of the CSDCLK1. The minimum frequency is:

$$F_{SW(minimum)} = \frac{F_{in}}{\text{PRS length-1}}$$

Equation 23-5

Where PRS length is either 12 or 8 bits. The average switching frequency is:

$$F_{SW(average)} = \frac{F_{in}}{4}$$

Equation 23-6

The PRS_CLEAR bit in CSD_CONFIG can be used to clear the PRS; when set, this bit forces the pseudo-random generator to its initial state.

23.4.3 Sigma Delta Converter

The sigma delta converter converts the input current to a corresponding digital count. It consists of a comparator, a voltage reference V_{REF}, a counter, and two current sourcing/sinking digital-to-analog converters (IDACs), as Figure 23-3 shows.

The sigma delta modulator controls the current of the 8-bit IDAC in an on/off manner. This IDAC is known as the modulation IDAC. The 7-bit IDAC, known as the compensation IDAC, is either always on or always off.

The sigma delta converter can operate in either single IDAC mode or dual IDAC mode. In the single IDAC mode, the compensation IDAC is always off. In the dual IDAC mode, the compensation IDAC is always on.

The sigma delta converter also requires an external integrating capacitor C_{MOD}, as Figure 23-1 shows. The recommended value of C_{MOD} is 2.2 nF. PRoC has a dedicated C_{MOD} pin. See the pinout in the device datasheet for details.

The sigma delta modulator maintains the voltage across C_{MOD} at V_{REF}. It works in one of the following modes:

- IDAC sourcing mode: If the switched capacitor circuit sinks current from AMUXBUS A, the IDACs source current to AMUXBUS A to balance its voltage.

- IDAC sinking mode: In this mode, the IDACs sink current from C_{MOD} and the switched capacitor circuit sources current to C_{MOD}.

In both cases, the modulation IDAC current is switched on and off corresponding to the small voltage variations across C_{MOD} to maintain the C_{MOD} voltage at V_{REF}.

The sigma delta converter can operate from 8-bit to 16-bit resolutions. In the single IDAC mode, the raw count is proportional to the sensor capacitance. If 'N' is the resolution of the sigma delta converter and I_{MOD} is the value of the modulation IDAC current, the approximate value of raw count in IDAC sourcing mode is given by Equation 16-7.

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S$$

Equation 23-7

Similarly, the approximate value of raw count in IDAC sinking mode is:

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S$$

Equation 23-8

In both cases, the raw count is proportional to sensor capacitance C_S. This raw count can be processed by the firmware to detect touches. You can use both the IDACs in a dual IDAC mode to improve the CapSense performance.

In this dual IDAC mode, the compensation IDAC is always on. If I_{COMP} is the compensation IDAC current, the equation for the raw count in IDAC sourcing mode is:

$$Rawcount = 2^N \frac{V_{REF} F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}}$$

Equation 23-9

Raw count in IDAC sinking mode is given by equation 16-10.

$$Rawcount = 2^N \frac{(V_{DD} - V_{REF}) F_{SW}}{I_{MOD}} C_S - 2^N \frac{I_{COMP}}{I_{MOD}}$$

Equation 23-10

Note that raw count values are always positive.

The hardware parameters such as I_{COMP}, I_{MOD}, and F_{SW}, should be tuned to optimum values for reliable touch detection. For a detailed discussion of the tuning process, see the PSoC 4 CapSense Design Guide.

Registers CSD_CONFIG, CSD_COUNTER, and CSD_IDAC control the operation of the sigma delta converter. The important bits in the CSD_CONFIG register are:

- ENABLE in CSD_CONFIG: Master enable of the CSD block. Must be set to '1' for any CSD operation to function.
- POLARITY in CSD_CONFIG: Selects between IDAC sinking mode and IDAC sourcing mode. 0: IDAC sourcing mode, 1: IDAC sinking mode.
- SENSE_COMP_BW in CSD_CONFIG: Selects the bandwidth of the sensing comparator. Setting this bit gives high bandwidth and clearing it gives low bandwidth. High bandwidth is recommended for CSD operation.
- SENSE_COMP_EN in CSD_CONFIG: Turns on the sense comparator circuit. 0: Sense comparator is powered off. 1: Sense comparator is powered on.
- SENSE_EN: Enables the sigma delta modulator output. Also turns on the IDACs.

The IDACs must be configured properly for CSD operation. See the CSD_IDAC register in PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

.

CSD_COUNTER register is used to initiate a sampling of the currently selected sensor and to read the result. The 16-bit COUNTER field in this register increments whenever the comparator is sampled (at the modulation clock frequency) and the sample is 1. Firmware typically writes ‘0’ to this field whenever a new sense operation is initiated. The 16-bit PERIOD field in the CSD_COUNTER register is used to initiate the capacitance to digital conversion. Writing a non-zero value to this register initiates a sensing operation. The value written to this field by the firmware determines the period during which the COUNTER field samples the comparator output.

The clocks, GPIOs, IDACs, and the sigma delta modulator must be properly configured before starting the CSD operation. The period field decrements after every modulation clock cycle. When it reaches 0, the COUNTER field stops incrementing. The value of this field at this time is the raw count corresponding to the value of sensor capacitance.

23.5 CapSense CSD Shielding

PRoC CapSense supports shield electrodes for waterproofing and proximity sensing. For waterproofing, the shield electrode is always kept at the same potential as the sensors. PRoC CapSense has a shielding circuit that drives the shield electrode with a replica of the sensor switching signal (see GPIO Cell Capacitance to Current Converter on page 247) to nullify the potential difference between sensors and the shield electrode. See the PSoC 4 CapSense Design Guide to understand the basics of shielding.

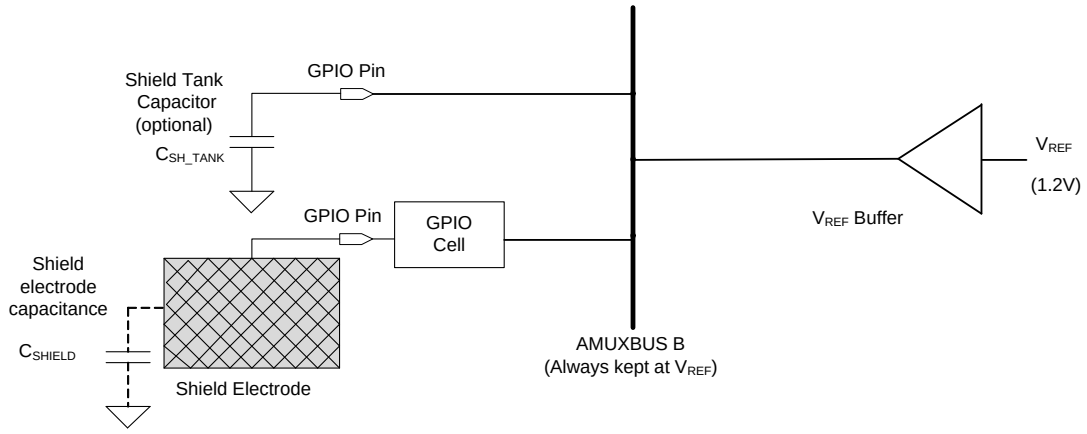
In the sensing circuit, the sigma delta converter keeps the AMUXBUS A at V_{REF} (see Sigma Delta Converter on

page 249). The GPIO cells generate the sensor waveforms by switching the sensor between AMUXBUS A and a supply rail (either V_{DD} or ground, depending on the configuration). The shielding circuit works in a similar way; AMUXBUS B is always kept at V_{REF}. The GPIO cell switches the shield between AMUXBUS B and a supply rail (either V_{DDD} or ground, the same configuration as the sensor). This process generates a replica of the sensor switching waveform on the shield electrode.

Depending on how AMUXBUS B is maintained at V_{REF}, two different configurations are possible.

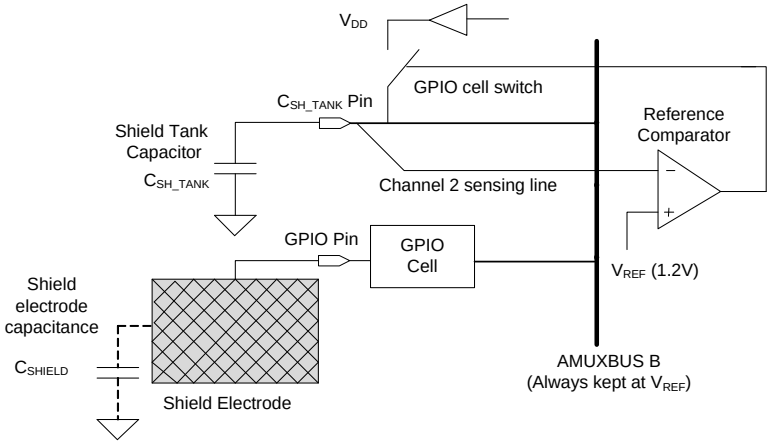
- Shield driving using V_{REF} buffer: In this configuration, a voltage buffer is used to drive AMUXBUS B to V_{REF}, as Figure 23-9 shows. An external C_{SH_TANK} capacitor is recommended to reduce switching transients. Setting the REBUF_OUTSEL bit in the CSD_CONFIG register connects the buffer output to AMUXBUS B. The REFBUF_DRV bit field in the same register can be used to set the drive strength of the buffer. Writing a '0' to this field disables the buffer; writing 1, 2, and 3 selects the low, mid, and high-current drive modes respectively.

Figure 23-9. Shield Driving Using V_{REF} Buffer



- Shield driving using GPIO cell precharge: This configuration requires an external C_{SH_TANK} capacitor, as Figure 23-10 shows. A special GPIO cell and a reference comparator is used to charge the C_{SH_TANK} capacitor and hence the AMUXBUS B to V_{REF}. The reference comparator always monitors the voltage on the CSH_TANK capacitor and controls the GPIO cell switch to keep the voltage at V_{REF}. The reference comparator connects to the C_{SH_TANK} capacitor using a dedicated sense line known as Channel 2 sensing line, as Figure 23-10 shows.

Figure 23-10. Shield Driving Using GPIO Precharge



This GPIO cell precharge capability is available only on a fixed C_{SH_TANK} pin. See the device pinout in the device datasheet for details.

COMP_MODE bit in the CSD_CONFIG register selects between the reference buffer precharge and GPIO precharge; 0: reference buffer precharge, 1: GPIO precharge.

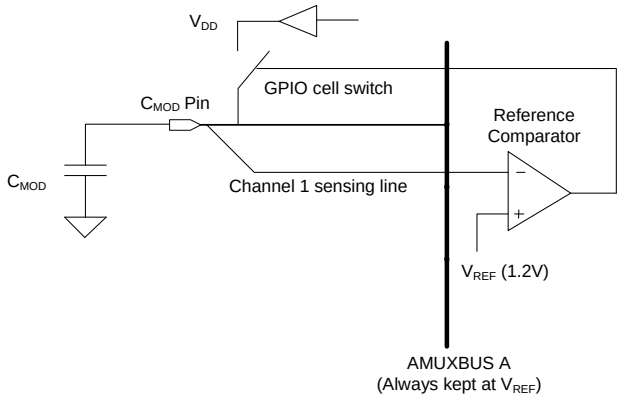
23.5.1 C_{MOD} Precharge

When the CapSense hardware is enabled for the first time, the voltage across C_{MOD} starts at zero. Then the sigma delta converter slowly charges the C_{MOD} to V_{REF}. The charging current is supplied by the IDACs in the IDAC sourcing mode and by the sensor switched capacitance circuit in the IDAC sinking mode. However, this is a slow process because C_{MOD} is a relatively large capacitor.

Precharging of C_{MOD} is the process of quickly initializing the voltage across C_{MOD} to V_{REF}. Precharging reduces the time required for the sigma delta converter to start its operation. There are two options for precharging C_{MOD}.

- Precharge using V_{REF} buffer: When the shield is enabled, the V_{REF} buffer output is always connected to AMUXBUS B (Figure 23-9). To precharge using the V_{REF} buffer, C_{MOD} is initially connected to AMUXBUS B. After the precharging process, C_{MOD} is connected to AMUXBUS A for normal sigma delta operation. When the shield is disabled, the V_{REF} buffer output is always connected to AMUXBUS A for precharging and disconnected afterwards.
- Precharge using GPIO cell: In this configuration, a special GPIO cell and a reference comparator is used to charge the C_{MOD} capacitor to V_{REF}. This GPIO cell precharge capability is available only on a fixed C_{MOD} pin. See the pinout in the device datasheet for details. The comparator used for this purpose is the same reference comparator used for CSH_TANK precharge. COMP_PIN bit in the CSD_CONFIG register is used to select which capacitor is connected to the reference comparator. If this bit is 0, the sense line designated as "Channel 1" is used to connect C_{MOD} to the reference comparator as Figure 23-11 shows; if this bit is 1, Channel 2 sense line is used to connect CSH_TANK to the reference comparator, as Figure 23-10 shows. Note that the GPIO cells must be configured properly for the GPIO cell precharge to work.

Figure 23-11. GPIO Cell Precharge



Precharge using a GPIO cell is faster than using the V_{REF} buffer. Therefore, GPIO precharge is the recommended precharge configuration. However, if you do not need a fast initialization of CapSense, use the V_{REF} buffer precharge.

The Channel 1 sense line can also be used to connect C_{MOD} to the sensing comparator in the sigma delta modulator. Setting the SENSE_INSEL bit in the CSD_CONFIG register to '1' enables this option. Clearing this bit connects C_{MOD} to the sensing comparator using AMUXBUS A.

23.6 General-Purpose Resources: IDACs and Comparator

If the CapSense block is not used for touch sensing, he sense comparator and the two IDACs can be used as general-purpose analog blocks.

You can use AMUXBUS A to connect any CSD-supported GPIO to the non-inverting input of the sense comparator. The inverting input is connected to the 1.2-V V_{REF} (see Figure 23-3). The AMUXBUS A can also be used as an analog multiplexer at the comparator input. The SENSE_COMP_EN, SENSE_COMP_BW, and ENABLE bits in the CSD_CONFIG register can be used to control the sense comparator, as explained in Sigma Delta Converter on page 249.

If AMUXBUS is required for other uses, the SENSE_INSEL bit in the CSD_CONFIG register can be used to connect the non-inverting input of the sense comparator to the fixed C_{MOD} pin, as explained in CMOD Precharge on page 252. The output of the comparator can connect to multiple GPIOs, see the I/O System chapter on page 65 for more details.

The 8-bit IDAC can operate in either 0 to 306 µA (1.2 µA/bit) or 0 to 612 µA (2.4 µA/bit) ranges. The 7-bit IDAC supports 0 to 152.4 µA (1.2 µA/bit) and 0 to 304.8 µA (2.4 µA/bit) ranges.

Both the 8-bit and 7-bit IDACs can connect to GPIOs using AMUXBUS A and AMUXBUS B. It is also possible to connect both IDACs to a single AMUXBUS. The IDACS can operate in three different modes: CSD-only mode, General-purpose (GP) mode, and CSD and GP mode. Table 23-1 describes how IDAC1 and IDAC2 are connected to AMUXBUS A and AMUXBUS B in each of these modes.

Table 23-1. IDAC Modes

Mode	AMUXBUS A	AMUXBUS B
CSD only	Both IDACs sink/source current at 1.2 V	No IDACs connected
General-purpose mode	8-bit IDAC sink/source current	7-bit IDAC sink/source current
CSD and GP mode	8-bit IDAC sink/source current at 1.2 V	7-bit IDAC sink/source current

See the CSD_IDAC register in PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

The CSD_CONFIG register can be used to enable the IDACs and set the polarity, as mentioned in Sigma Delta Converter on page 249. See the I/O System chapter on page 65 for details on how to connect GPIOs to AMUXBUS A and B.

23.7 Register List

Table 23-2. CapSense Register List

Register Name	Description
CSD_CONFIG	This register is used to configure and control the CSD block and its resources.
CSD_IDAC	This register is used to control the IDAC current settings.
CSD_COUNTER	This register is used to initiate a sampling of the selected capacitive sensor and read the result of conversion.
CSD_STATUS	This register allows the observation of key signals in the CSD block.
CSD_INTR	This is the CSD interrupt request register.

24. Temperature Sensor

PRoC has an on-chip temperature sensor that is used to measure the internal die temperature. The sensor consists of a transistor connected in diode configuration.

24.1 Features

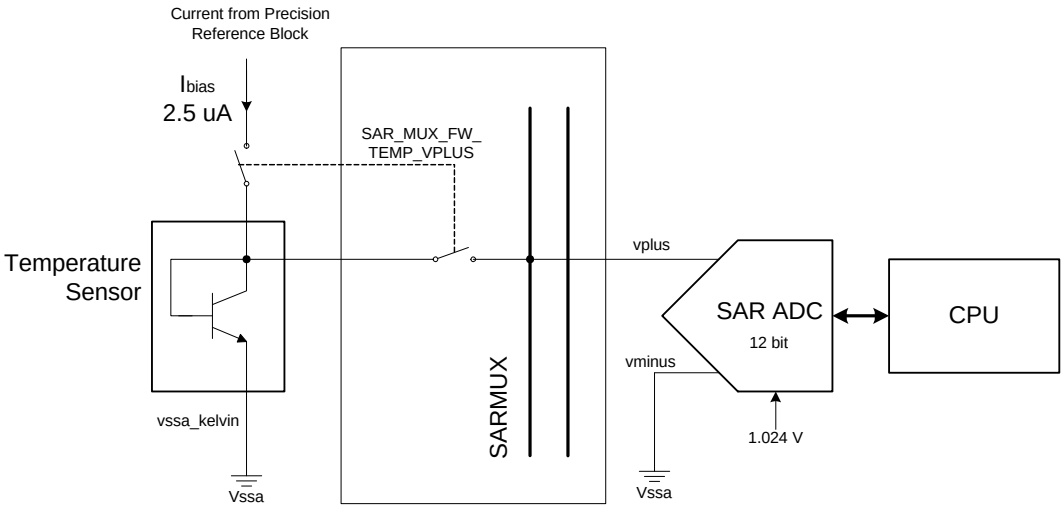
The temperature sensor has the following features:

- ± 5° Celsius accuracy over temperature range –40 °C to +85 °C
- 0.5° Celsius/LSB resolution (without amplification) when using a 12-bit SAR ADC with a 1.024-V reference
- 10 µs settling time

24.2 How it Works

The temperature sensor consists of a single bipolar junction transistor (BJT) in the form of a diode. Its base-to-emitter voltage (V_{BE}) has a strong dependence on temperature at a constant collector current and zero collector-base voltage. This property is used to calculate the die temperature by measuring the V_{BE} of the transistor using SAR ADC, as shown in Figure 24-1.

Figure 24-1. Temperature Sensing Mechanism



The analog output from the sensor (V_{BE}) is measured using the SAR ADC. Die temperature in °C can be calculated from the ADC results as given in the following equation:

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

Equation 24-1

- Temp is the slope compensated temperature in °C represented as Q16.16 fixed point number format.
- "A" is the 16-bit multiplier constant. The value of A is determined using the PRoC family characterization data of two point slope calculation. It is calculated as given in the following equation.

$$A = (\text{signedint})\left(2^{16}\left(\frac{100°C-(-40°C)}{SAR_{100°C}-SAR_{-40°C}}\right)\right)$$ **Equation 24-2**

Where,

SAR_{100C} = ADC counts at 100°C

SAR_{-40C} = ADC counts at –40°C

Constant 'A' is stored in a register SFLASH_SAR_TEMP_MULTIPLIER.

- "B" is the 16-bit offset value. The value of B is determined on a per die basis by taking care of all the process variations and the actual bias current (I_{bias}) present in the chip. It is calculated as given in the following equation.

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

Where,

SAR_{100C} = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH_SAR_TEMP_OFFSET.

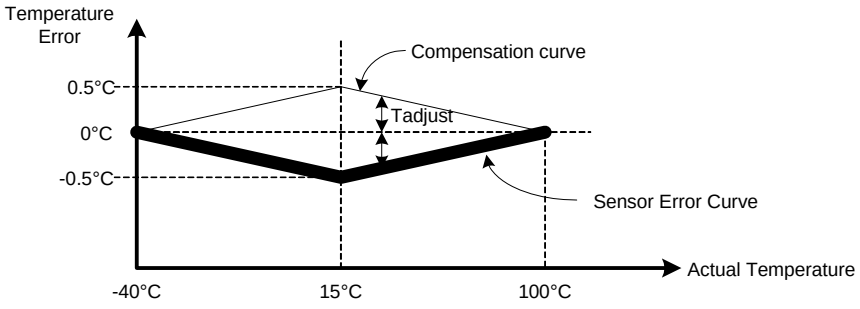
- T_{adjust} is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating $T_{initial} = (A\times SAR_{out}+ 2^{10}\times B)$. If it is greater than the center value (15°C), then T_{adjust} is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{100°C-15°C}\times(100°C\times 2^{16}-T_{initial})\right)$$ **Equation 24-4**

If less than center value, then Tadjust is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{40°C+15°C}\times(40°C\times 2^{16}-T_{initial})\right)$$ **Equation 24-5**

Figure 24-2. Temperature Error Compensation



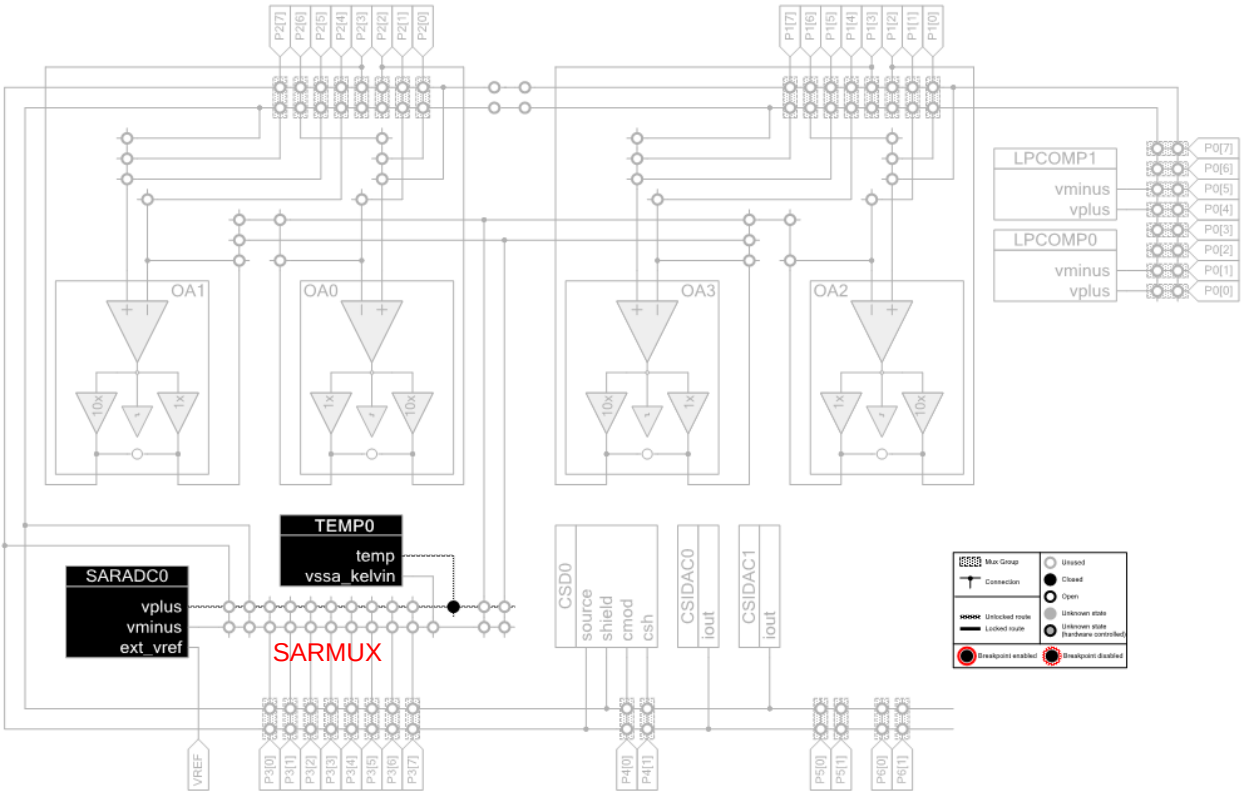
Note A and B are 16-bit constants stored in flash during factory calibration. Note that these constants are valid only when the SAR ADC is running at 12 bit resolution with a 1.024-V reference.

24.3 Temperature Sensor Configuration

As shown in Figure 24-3, the temperature sensor output is routed to the positive input of SAR ADC via dedicated switches, which can be controlled by sequencer, firmware, or digital system interconnect (DSI). The control signal for the switch (SAR_MUX_FW_TEMP_VPLUS shown in Figure 24-1) enables the temperature sensor by passing bias current from precision reference block and connecting the sensor output to the positive input of SAR ADC. The SAR_MUX_FW_TEMP_VPLUS

control bit is a part of the SAR_MUX_SWITCH0 register. The switch status can be read using the SAR_MUX_SWITCH_STATUS register.

Figure 24-3. Routing Temperature Sensor Output to SAR ADC



24.4 Algorithm

1. Enable the SARMUX and SAR ADC.
2. Configure SAR ADC in single-ended mode with $V_{NEG} = V_{SS}$, V_{REF} = 1.024 V, 12-bit resolution, and right-aligned result.
3. Enable the temperature sensor.
4. Get the digital output from the SAR ADC.
5. Fetch ‘A’ from SFLASH_SAR_TEMP_MULTIPLIER and ‘B’ from SFLASH_SAR_TEMP_OFFSET.
6. Calculate the die temperature using the linear equation (Equation 24-1).

   For example, let A = 0xBC4B and B = 0x65B4. Assume that the output of SAR ADC (V_{BE}) is 0x595 at a given temperature.

   Firmware does the following calculations:

   a. Multiply A and V_{BE}: 0xBC4B × 0x595 = $(–17333)_{10} \times (1429)_{10} = (–24768857)_{10}$
   b. Multiply B and 1024: 0x65B4 × 0x400 = $(26036)_{10} \times (1024)_{10} = (26660864)_{10}$
   c. Add the result of steps 1 and 2 to get $T_{initial}$: $(–24768857)_{10} + (26660864)_{10} = (1892007)_{10}$ = 0x1CDEA7
   d. Calculate T_{adjust} using $T_{initial}$ value: $T_{initial}$ is the upper 16 bits multiplied by 2^{16}, that is, 0x1C00 = $(1835008)_{10}$. It is greater than 15°C (0x1C - upper 16 bits). Use Equation 4 to calculate T_{adjust}. It comes to 0x6C6C = $(27756)_{10}$.
   e. Add T_{adjust} to $T_{initial}$: $(1892007)_{10} + (27756)_{10} = (1919763)_{10}$ = 0x1D4B13
   f. The integer part of temperature is the upper 16 bits = 0x001D = $(29)_{10}$
   g. The decimal part of temperature is the lower 16 bits = 0x4B13 = $(0.19219)_{10}$

h. Combining the result of steps f and g, Temp = 29.19219 °C ~ 29.2°C

24.5 Registers

Name	Description
SAR_MUX_SWITCH0	This register has the SAR_MUX_FW_TEMP_VPLUS field to connect the temperature sensor to the SAR MUX terminal.
SAR_MUX_SWITCH_STATUS	This register provides the status of the temperature sensor switch connection to SAR MUX.
SFLASH_SAR_TEMP_MULTIPLIER	Multiplier constant 'A' as defined in Equation 24-1.
SFLASH_SAR_TEMP_OFFSET	Constant 'B' as defined in Equation 24-1.

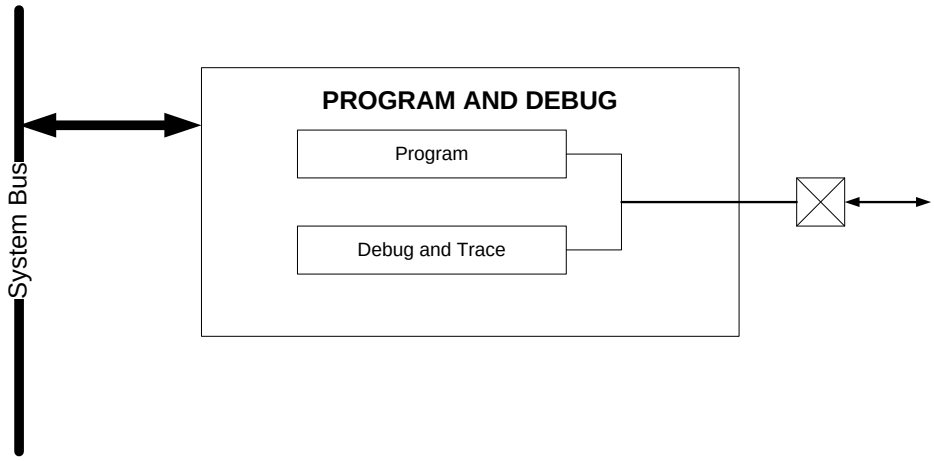
Section F: Program and Debug

This section encompasses the following chapters:

- Program and Debug Interface chapter on page 261
- Nonvolatile Memory Programming chapter on page 267

Top Level Architecture

Program and Debug Block Diagram



25. Program and Debug Interface

The PRoC 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 PRoC programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and PRoC.

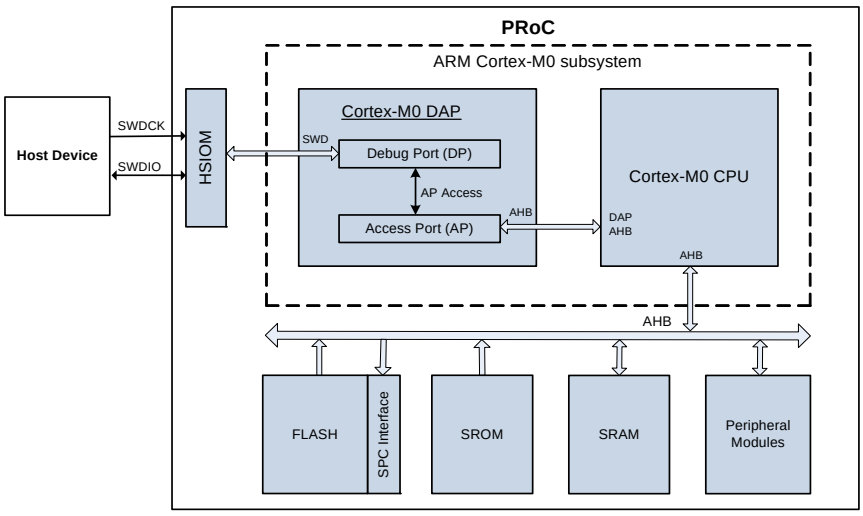
25.1 Features

- Programming and debugging through the SWD interface
- Four hardware breakpoints and two hardware watchpoints while debugging
- Read and write access to all memory and registers in the system while debugging, including the Cortex-M0 register bank when the core is running or halted

25.2 Functional Description

Figure 25-1 shows the block diagram of the program and debug interface in PRoC. 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 PRoC "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 65 for details on HSIOM.

Figure 25-1. 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 PRoC, which facilitates memory and peripheral register access by the AHB master. PRoC has two AHB masters – ARM CM0 CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

25.3 Serial Wire Debug (SWD) Interface

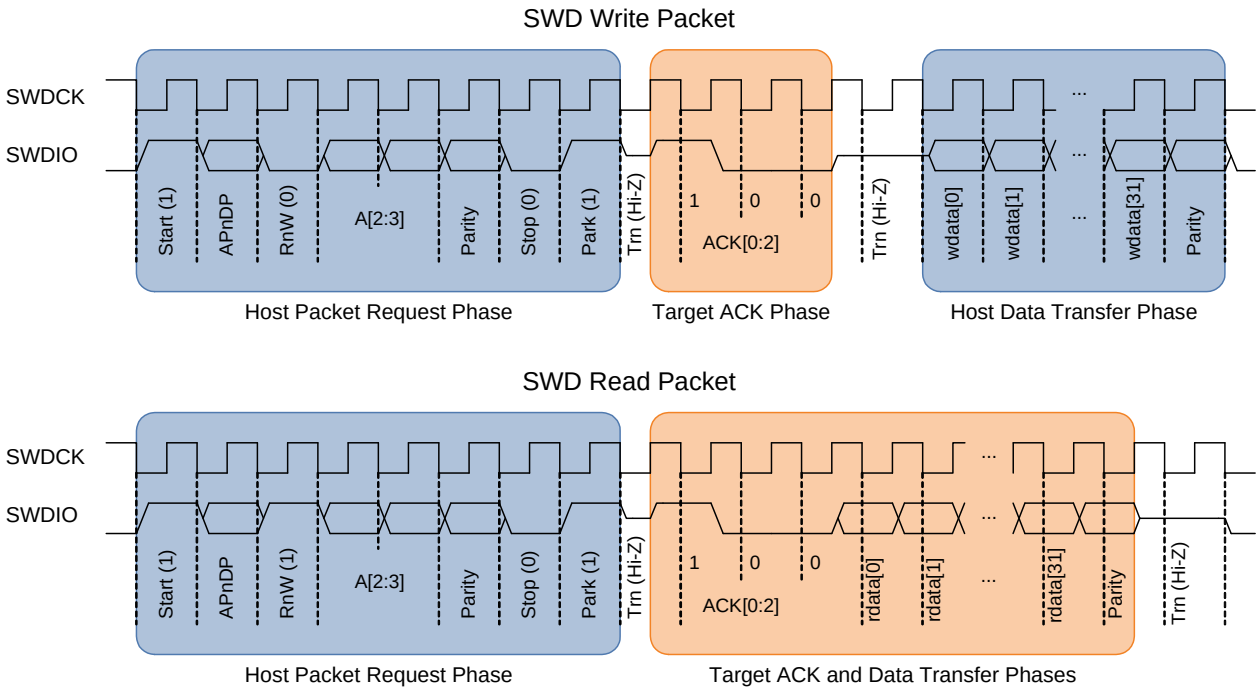
PRoC's Cortex-M0 supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, 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 PRoC target.
- **Target Acknowledge Response Phase** – The PRoC target sends an acknowledgement to the host.
- **Data Transfer Phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

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

Figure 25-2. 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 25-3 and Table 25-4 for definitions.
   Note Address bits are transmitted with the LSB first.
   e. The parity bit contains the parity of APnDP, RnW, and ADDR bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.
   If the parity bit is not correct, the header is ignored by PRoC; there is no ACK response (ACK = 3b111). The programming operation should be aborted and retried again by following a device reset.
   f. The stop bit is always logic 0.
   g. The park bit is always logic 1.
2. Target Acknowledge Response Phase: SWDIO driven by the target
   a. The ACK[2:0] bits represent the target to host response, indicating failure or success, among other results. See Table 25-1 for definitions. **Note** ACK bits are transmitted with the LSB first.

3. Data Transfer Phase: SWDIO driven by either target or host depending on direction
   a. The data for read or write is written to the bus, LSB first.
   b. The data parity bit indicates the parity of the data read or written. It is an even parity; this means when XORed with the data bits, the result will be 0.

      If the parity bit indicates a data error, corrective action should be taken. For a read packet, if the host detects a parity error, it must abort the programming operation and restart. For a write packet, if the target detects a parity error, it generates a FAULT ACK response in the next packet.

According to the SWD protocol, the host can generate any number of SWDCK clock cycles between two packets with SWDIO low. It is recommended to generate three or more dummy clock cycles between two SWD packets if the clock is not free-running or to make the clock free-running in IDLE mode.

The SWD interface can be reset by clocking the SWDCK line for 50 or more cycles with SWDIO high. To return to the idle state, clock the SWDIO low once.

25.3.1 SWD Timing Details

The SWDIO line is written to and read at different times depending on the direction of communication. The host drives the SWDIO line during the Host Packet Request Phase and, if the host is writing data to the target, during the Data Transfer phase as well. When the host is driving the SWDIO line, each new bit is written by the host on falling SWDCK edges, and read by the target on rising SWDCK edges. The target drives the SWDIO line during the Target Acknowledge Response Phase and, if the target is reading out data, during the Data Transfer Phase as well. When the target is driving the SWDIO line, each new bit is written by the target on rising SWDCK edges, and read by the host on falling SWDCK edges.

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

Table 25-1. SWDIO Bit Write and Read Timing

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

25.3.2 ACK Details

The acknowledge (ACK) bit-field is used to communicate the status of the previous transfer. OK ACK means that previous packet was successful. A WAIT response requires a data phase. For a FAULT status, the programming operation should be aborted immediately. Table 25-2 shows the ACK bit-field decoding details.

Table 25-2. SWD Transfer ACK Response Decoding

Response	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	3b111

Details on WAIT and FAULT response behaviors are as follows:

- For a WAIT response, if the transaction is a read, the host should ignore the data read in the data phase. The target does not drive the line and the host must not check the parity bit as well.
- For a WAIT response, if the transaction is a write, the data phase is ignored by the PRoC. 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 PRoC is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received. If it fails, then the programming operation should be aborted and retried again.
- For a FAULT response, the programming operation should be aborted and retried again by doing a device reset.

25.3.3 Turnaround (Trn) Period Details

There is a turnaround period between the packet request and the ACK phases, as well as between the ACK and the data phases for host write transfers, as shown in Figure 25-2. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor PRoC should drive the SWDIO line during the Trn period.

25.4 Cortex-M0 Debug and Access Port (DAP)

The Cortex-M0 program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex-M0 DAP-AHB interface. Typically, the DP registers are used for a one time configuration or for error detection purposes, and the AP

registers are used to perform the programming and debugging operations. Complete architecture details of the DAP is available in the *ARM® Debug Interface v5 Architecture Specification*.

25.4.1 Debug Port (DP) Registers

Table 25-3 shows the Cortex-M0 DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the *ARM® Debug Interface v5 Architecture Specification* for details on all of the DP registers.

Table 25-3. Main Debug Port (DP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex-M0 CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In PRoC, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

25.4.2 Access Port (AP) Registers

Table 25-4 lists the main Cortex-M0 AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 25-4. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the PRoC Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register

25.5 Programming the PRoC Device

PRoC is programmed using the following sequence. Refer to the PRoC BLE Programming Specifications for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in PRoC.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

25.5.1 SWD Port Acquisition

25.5.1.1 Primary and Secondary SWD Pin Pairs

The first step in device programming is to acquire the SWD port in PRoC. Refer to the device datasheet for information on SWD pins.

If two SWD pin pairs are available in the device, the SWD_CONFIG register in the supervisory flash region is used to select between one of the two SWD pin pairs that can be used for programming and debugging. Note that only one of the SWD pin pairs can be used during any programming or debugging session. The default selection for devices coming from the factory is the primary SWD pin pair. To select the secondary SWD pin pair, it is necessary to program the device using the primary pair with the hex file

that enables the secondary pin pair configuration. Afterwards, the secondary SWD pin pair may be used.

25.5.1.2 SWD Port Acquire Sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After 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 < 1.5 ms); // retry connection until OK ACK or timeout

if (time_elapsed >= 1.5 ms) return FAIL; //check for acquire time out

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

In this 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 (1.5 ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex-M0 DAP.

25.5.2 SWD Programming Mode Entry

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

25.5.3 SWD Programming Routines Executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in the Nonvolatile Memory Programming chapter on page 267. The exact sequence of calling the programming routines is given in the PRoC BLE Programming Specifications document.

25.6 PRoC SWD Debug Interface

Cortex-M0 DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex-M0 Debug.

See the *ARMv6-M Architecture Reference Manual* for complete details on the debug architecture.

25.6.1 Debug Control and Configuration Registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the *ARMv6-M Architecture Reference Manual* for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (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. This 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.

25.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0 DAP in PRoC 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 PRoC).
- 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.

25.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. Trace is not supported by the Cortex-M0 in PRoC. 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.

25.6.4 Debugging the PRoC Device

The host debugs the target PRoC device by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0 DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PRoC device is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the correct CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the I/O System chapter on page 65 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as LCD and GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

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

Debugging the target device is also affected by the overall device protection setting, which is explained in the Device Security chapter on page 103. Only the OPEN protected mode supports device debugging. Also, the external debugger loses connection to the target device when the device enters either Hibernate or Stop modes. The connection must be re-established after the device enters the Active mode again. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

25.7 Registers

Table 25-5. 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

26. Nonvolatile Memory Programming

Nonvolatile memory programming refers to the programming of flash memory in the PRoC 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 PRoC device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

26.1 Features

- Supports programming through the debug and access port (DAP) and Cortex-M0 CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex-M0 CPU

26.2 Functional Description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0 CPU requests the system call by writing the function opcode and parameters to the 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.

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

External programmers program the flash memory in PRoC using the SWD protocol by sending the commands to the Debug and Access Port (DAP). The programming sequence for the PRoC device with an external programmer is given in the PRoC BLE 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 99) 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.

26.3 System Call Implementation

A system call consists of the following items:

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

  key1 = 0xB6

  key2 = 0xD3 + Opcode

  The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are 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.

26.4 Blocking and Non-Blocking System Calls

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

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

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

26.4.1 Performing a System Call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, additional parameters) are:
   a. Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b. Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits are not cleared after 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 26-1 for the complete list of status codes and their description.
5. Retrieve the return values: For system calls that return values such as silicon ID and checksum, the CPU or DAP should read the CPUSS_SYSREG and CPUSS_SYSARG registers to fetch the values returned.

26.5 System Calls

Table 26-1 lists all the system calls supported in PRoC along with the function description and availability in device protection modes. See the Device Security chapter on page 103 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

Table 26-1. List of System Calls

System Call	Description	DAP Access			CPU Access
		Open	Protected	Kill	
Silicon ID	Returns the device Silicon ID, Family ID, and Revision ID	✔	✔	–	✔
Load Flash Bytes	Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity	✔	–	–	✔
Write Row	Erases and then programs a row of flash with data in the page latch buffer	✔	–	–	✔
Program Row	Programs a row of flash with data in the page latch buffer	✔	–	–	✔
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	✔	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔	–	✔
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	✔	✔	–	
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	✔

26.5.1 Silicon ID

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

Parameters

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

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

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [7:0]	Silicon ID Lo	See the device datasheet for Silicon ID values for different part numbers
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision Id	See the PRoC BLE Programming Specifications for these values
Bits [23:20]	Major Revision Id	
Bits [27:24]	0xXX	Not used (don't care)
Bits [31:28]	0xA	Success status code
CPUSS_SYSREQ register		
Bits [11:0]	Family ID	Family ID is 0x093 for PRoC BLE
Bits [15:12]	Chip Protection	See the Device Security chapter on page 103
Bits [31:16]	0xXXXX	Not used

26.5.2 Configure Clock

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

Note This applies only to PRoC CYBL10x6x device.

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)

26.5.3 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 128 bytes for PRoC CYBL10x6x devices and 256 bytes for PRoC CYBL1xx7x devices. Data is loaded into the page latch buffer starting at the location specified by the “Byte Addr” input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS_SYSARG register. Note that the starting parameter address should be a word-aligned address.

Parameters

Address	Value to be Written	Description
SRAM Address - 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD7	Key2
Bits [23:16]	Byte Addr	Start address of page latch buffer to write data 0x00 – Byte 0 of latch buffer 0x7F - Byte 127 of latch buffer (applicable to PRoC CYBL10x6x devices) 0xFF - Byte 255 of latch buffer (applicable to PRoC CYBL1xx7x devices)
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 (Refer to the Cortex-M0 CPU chapter on page 33 for the number of flash macros in the device)
SRAM Address- 32’hYY + 0x04		
Bits [7:0]	Load Size	Number of bytes to be written to the page latch buffer. 0x00 – 1 byte 0x7F – 128 bytes (applicable to PRoC CYBL10x6x devices) 0xFF – 256 bytes (applicable to PRoC CYBL1xx7x devices)
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)

26.5.4 Write Row

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

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Note This applies only to PRoC CYBL10x6x device.

Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write protected.

Note that this system call disables the 36-MHz IMO output before performing the flash write operation. The 36-MHz IMO output can be used to source the analog switch pump or the CTBm pump. If the 36-MHz IMO output is used, it must be manually re-enabled after the system call completes. Specifically, the CLK_IMO_CONFIG EN_CLK36 and FLASHPUMP_SEL must be reset.

Refer to the CLK_IMO_CONFIG register in PRoC® BLE: CYBL1XXXX Family - Programmable Radio-on-Chip With BLE (PRoC BLE) Registers Technical Reference Manual (TRM).

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0005	Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

26.5.5 Program Row

This function programs the addressed row of the flash, with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The row must be in an erased state before calling this function. This 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. Note This applies only to PRoC CYBL10x6x device.

Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD9	Key2
Bits [31:16]	Row ID	Row number to program 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0006	Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

26.5.6 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in 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. Note This applies only to PRoC CYBL10x6x device.

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)

26.5.7 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDE	Key2
Bits [31:16]	Row ID	Selects the flash row number on which the checksum operation is done Row number – 16 bit flash row number or 0x8000 – Checksum is performed on entire flash memory
CPUSS_SYSREQ register		
Bits [15:0]	0x000B	Checksum opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	Checksum	24-bit checksum value of the selected flash region

26.5.8 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. Note This applies only to PRoC CYBL10x6x device.

The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

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

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE0	Key2
Bits [23:16]	Device Protection Byte	Parameter applicable only for Flash Macro 0 0x01 – OPEN mode 0x02 – PROTECTED mode 0x04 – KILL mode
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1
CPUSS_SYSREQ register		
Bits [15:0]	0x000D	Write Protection opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	0x000000	

26.5.9 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from 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 268.

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

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

Usage Requirements: Call the 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. Note This applies only to PRoC CYBL10x6x device.

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

Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0 CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

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

Return

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

26.5.10 Non-Blocking Program Row

This function is used when a flash row needs to be programmed by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from the SRAM when the program operation is being done. The explanation of non-blocking system calls is explained in Blocking and Non-Blocking System Calls on page 268. 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. Note This applies only to PRoC CYBL10x6x device.

Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0 CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

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

Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

26.5.11 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are explained in Blocking and Non-Blocking System Calls on page 268.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDC	Key2
Bits [31:16]	0xXXXX	Don't care. Not used by SROM
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0009	Resume Non-Blocking opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

26.6 System Call Status

At the end of every system call, a status code is written over the arguments in the CPUSS_SYSARG register. A success status is 0xAXXXXXXX, where X indicates don't care values or return data in the case of the system calls that return a value. A failure status is indicated by 0xF00000XX, where XX is the failure code.

Table 26-2. System Call Status Codes

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

26.7 Non-Blocking System Call Pseudo Code

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

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

/* Note: Use the right macro according to your device */
#define ROW_SIZE_128    (128)
#define ROW_SIZE_256    (256)
#define ROW_SIZE        (ROW_SIZE_128)

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

__flash int main(void)
{
    DoUserStuff();

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

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

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

    /* End Program */
    while(1);
}

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

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

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

    /* Number of times the ISR has triggered */
    iStatusInt ++;
}

__sram void NonBlockingWriteRow(void)
{
    int iter;

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

    /* Write load size param (128/256 bytes) to SRAM */
    #if (ROW_SIZE == ROW_SIZE_128)
        REG( 0x20000004 ) = 0x0000007F;
    #elif (ROW_SIZE == ROW_SIZE_256)
        REG( 0x20000004 ) = 0x000000FF;
    #endif  /* #if (ROW_SIZE == ROW_SIZE_128) */

    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 53 for details on configuring the CM0 exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PRoC 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 PRoC's M8CP is an 8-bit microcontroller, the PRoC 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 PRoC block or an analog PRoC 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 PRoC 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 PRoC 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 PRoC 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

I²C A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). I^2C is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. I^2C uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.

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

impedance
1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.
2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.

input A point that accepts data, in a device, process, or channel.

input/output (I/O) A device that introduces data into or extracts data from a system.

instruction An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.

instruction mnemonics A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.

integrated circuit (IC) A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.

interface The means by which two systems or devices are connected and interact with each other.

interrupt A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

interrupt service routine (ISR) A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter
1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams.
2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

L

latency — The time or delay that it takes for a signal to pass through a given circuit or network.

least significant bit (LSb) — The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.

least significant byte (LSB) — The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

Linear Feedback Shift Register (LFSR) — A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.

load — The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).

logic function — A mathematical function that performs a digital operation on digital data and returns a digital value.

lookup table (LUT) — A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.

low time — The amount of time the signal has a value of ‘0’ in one period, for a periodic digital signal.

low-voltage detect (LVD) — A circuit that senses V_{DDD} and provides an interrupt to the system when V_{DDD} falls below a selected threshold.

M

M8CP — An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PRoC device by interfacing to the flash, SRAM, and register space.

macro — A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.

mask —

1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference.
2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.

master device A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.

microcontroller An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.

mnemonic A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode A distinct method of operation for software or hardware. For example, the Digital PRoC block may be in either counter mode or timer mode.

modulation A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux)

1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

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

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise

1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR	See *Boolean Algebra*.
NOT	See *Boolean Algebra*.

O

OR	See *Boolean Algebra*.
oscillator	A circuit that may be crystal controlled and is used to generate a clock frequency.
output	The electrical signal or signals which are produced by an analog or digital block.

P

parallel	The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.
parameter	Characteristics for a given block that have either been characterized or may be defined by the designer.
parameter block	A location in memory where parameters for the SSC instruction are placed prior to execution.
parity	A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).
path	1. The logical sequence of instructions executed by a computer. 2. The flow of an electrical signal through a circuit.
pending interrupts	An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.
phase	The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).
pin	A terminal on a hardware component. Also called lead.
pinouts	The pin number assignment: the relation between the logical inputs and outputs of the PRoC 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 PRoC 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.
PRoC 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 base-line 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 PRoC device.
ripple divider	An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to 2^n - 1.
rising edge	*See positive edge*.
ROM	An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.
routine	A block of code, called by another block of code, that may have some general or frequent use.
routing	Physically connecting objects in a design according to design rules set in the reference library.

runt pulses In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling The process of converting an analog signal into a series of digital values or reversed.

schematic A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.

seed value An initial value loaded into a linear feedback shift register or random number generator.

serial
1. Pertaining to a process in which all events occur one after the other.
2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.

set To force a bit/register to a value of logic 1.

settling time The time it takes for an output signal or value to stabilize after the input has changed from one value to another.

shift The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.

shift register A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.

sign bit The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.

signal A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.

silicon ID A unique identifier of the PRoC 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 PRoC 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 PRoC 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 PRoC 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 PRoC 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

L

O

P

R

S

U

W