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

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

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

PSoC 4200L Family

PSoC® 4 Architecture Technical Reference Manual (TRM)

Document No. 001-97952 Rev. *E

June 3, 2024

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

Copyrights

© Cypress Semiconductor Corporation, 2015–2024. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("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 a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (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, and (2) 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 for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach").Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, 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. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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.

Content Overview

Section A: Overview **14**

1. Introduction 15
2. Getting Started 20
3. Document Construction 22

Section B: CPU System **25**

4. Cortex-M0 CPU 26
5. DMA Controller Modes 31
6. Interrupts 45

Section C: System Resources Subsystem (SRSS) **54**

7. I/O System 55
8. Clocking System 72
9. Power Supply and Monitoring 86
10. Chip Operational Modes 90
11. Power Modes 91
12. Watchdog Timer 97
13. Reset System 101
14. Device Security 104

Section D: Digital System **106**

15. Serial Communications Block (SCB) 107
16. Universal Digital Blocks (UDB) 148
17. Controller Area Network (CAN) 190
18. Universal Serial Bus (USB) 207
19. Timer, Counter, and PWM 225

Section E: Analog System **248**

20. Precision Reference 249
21. SAR ADC 252
22. Low-Power Comparator 285
23. Continuous Time Block mini (CTBm) 291
24. LCD Direct Drive 301
25. CapSense 313
26. Temperature Sensor 314

Section F: Program and Debug **317**
27. Program and Debug Interface 318
28. Nonvolatile Memory Programming 325

Glossary **338**

Contents

Section A: Overview **14**

1. Introduction 15
   - 1.1 Top Level Architecture 15
   - 1.2 Features 16
   - 1.3 CPU System 16
     - 1.3.1 Processor 16
     - 1.3.2 Interrupt Controller 16
     - 1.3.3 Direct Memory Access 17
   - 1.4 Memory 17
     - 1.4.1 Flash 17
     - 1.4.2 SRAM 17
   - 1.5 System-Wide Resources 17
     - 1.5.1 Clocking System 17
     - 1.5.2 Power System 17
     - 1.5.3 GPIO 18
   - 1.6 Programmable Digital 18
   - 1.7 Fixed-Function Digital 18
     - 1.7.1 Timer/Counter/PWM Block 18
     - 1.7.2 Serial Communication Blocks 18
     - 1.7.3 Controller Area Network 18
   - 1.8 Analog System 18
     - 1.8.1 SAR ADC 18
     - 1.8.2 Continuous Time Block mini 18
     - 1.8.3 Low-Power Comparators 18
   - 1.9 Special Function Peripherals 19
     - 1.9.1 LCD Segment Drive 19
     - 1.9.2 CapSense 19
   - 1.10 Program and Debug 19
2. Getting Started 20
   - 2.1 PSoC 4 resources 20
   - 2.2 PSoC Creator 20
3. Document Construction 22
   - 3.1 Major Sections 22
   - 3.2 Documentation Conventions 22
     - 3.2.1 Register Conventions 22
     - 3.2.2 Numeric Naming 22
     - 3.2.3 Units of Measure 23
     - 3.2.4 Acronyms 23

Section B: CPU System **25**

4. Cortex-M0 CPU 26
   - 4.1 Features 26

4.2 Block Diagram 27
4.3 How It Works 27
4.4 Address Map 27
4.5 Registers 28
4.6 Operating Modes 29
4.7 Instruction Set 29
4.7.1 Address Alignment 30
4.7.2 Memory Endianness 30
4.8 Systick Timer 30
4.9 Debug 30
5. DMA Controller Modes 31
5.1 Block Diagram Description 31
5.1.1 Trigger Sources and Multiplexing 32
5.1.2 Pending Triggers 35
5.1.3 Output Triggers 35
5.1.4 Channel Prioritization 35
5.1.5 Data Transfer Engine 35
5.2 Descriptors 36
5.2.1 Address Configuration 36
5.2.2 Transfer Size 37
5.2.3 Descriptor Chaining 38
5.2.4 Transfer Mode 38
5.3 Operation and Timing 42
5.4 Arbitration 43
5.5 Register List 44
6. Interrupts 45
6.1 Features 45
6.2 How It Works 45
6.3 Interrupts and Exceptions - Operation 46
6.3.1 Interrupt/Exception Handling 46
6.3.2 Level and Pulse Interrupts 46
6.3.3 Exception Vector Table 47
6.4 Exception Sources 47
6.4.1 Reset Exception 47
6.4.2 Non-Maskable Interrupt (NMI) Exception 48
6.4.3 HardFault Exception 48
6.4.4 Supervisor Call (SVCall) Exception 48
6.4.5 PendSV Exception 48
6.4.6 SysTick Exception 48
6.5 Interrupt Sources 49
6.6 Exception Priority 50
6.7 Enabling and Disabling Interrupts 51
6.8 Exception States 51
6.8.1 Pending Exceptions 51
6.9 Stack Usage for Exceptions 52
6.10 Interrupts and Low-Power Modes 52
6.11 Exceptions – Initialization and Configuration 53
6.12 Registers 53
6.13 Associated Documents 53

Section C: System Resources Subsystem (SRSS) 54

7. I/O System 55
7.1 Features 55
7.2 GPIO Interface Overview 55
7.3 I/O Cell Architecture 56
7.3.1 Digital Input Buffer 58
7.3.2 Digital Output Driver 58
7.4 GPIO-OVT Pin 60
7.5 Special I/Os (SIOs) 63
7.5.1 Reference Generator 64
7.5.2 Differential Input Buffer 65
7.5.3 Regulated Output Driver 65
7.5.4 Overvoltage Tolerance 66
7.6 High-Speed I/O Matrix 66
7.7 I/O State on Power Up 67
7.8 Behavior in Low-Power Modes 67
7.9 Input and Output Synchronization 67
7.10 Interrupt 67
7.11 Peripheral Connections 69
7.11.1 Firmware Controlled GPIO 69
7.11.2 Analog I/O 69
7.11.3 LCD Drive 69
7.11.4 CapSense 70
7.11.5 Serial Communication Block (SCB) 70
7.12 Port Restrictions 70
7.13 Registers 71

8. Clocking System 72
8.1 Block Diagram 72
8.2 Clock Sources 73
8.2.1 Internal Main Oscillator 73
8.2.2 Internal Low-speed Oscillator 75
8.2.3 External Clock (EXTCLK) 76
8.2.4 External Crystal Oscillator (ECO) 76
8.2.5 PLL0 and PLL1 77
8.2.6 Watch Crystal Oscillator (WCO) 78
8.3 Clock Distribution 78
8.3.1 HFCLK and PLL Input Selection 79
8.3.2 HFCLK Input Selection 79
8.3.3 PLL Input Selection 80
8.3.4 LFCLK Input Selection 80
8.3.5 SYSCLK Prescaler Configuration 81
8.3.6 Peripheral Clock Divider Configuration 81
8.3.7 Peripheral Clock Configuration 84
8.4 Low-Power Mode Operation 85
8.5 Register List 85

9. Power Supply and Monitoring 86
9.1 Block Diagram 87
9.2 How It Works 87
9.2.1 Regulator Summary 87
9.3 Voltage Monitoring 88
9.3.1 Power-On-Reset (POR) 88
9.4 Register List 89

10. Chip Operational Modes 90
10.1 Boot 90
10.2 User 90
10.3 Privileged 90
10.4 Debug 90
11. Power Modes 91
11.1 Active Mode 92
11.2 Sleep Mode 92
11.3 Deep-Sleep Mode 92
11.4 Hibernate Mode 93
11.5 Stop Mode 93
11.6 Power Mode Summary 94
11.7 Low-Power Mode Entry and Exit 95
11.8 Register List 96
12. Watchdog Timer 97
12.1 Features 97
12.2 Block Diagram 97
12.3 How It Works 98
12.3.1 Enabling and Disabling WDT 99
12.3.2 WDT Operating Modes 99
12.3.3 WDT Interrupts and Low-Power Modes 100
12.3.4 WDT Reset Mode 100
12.4 Register List 100
13. Reset System 101
13.1 Reset Sources 101
13.1.1 Power-on Reset 101
13.1.2 Brownout Reset 101
13.1.3 Watchdog Reset 101
13.1.4 Software Initiated Reset 102
13.1.5 External Reset 102
13.1.6 Protection Fault Reset 102
13.1.7 Hibernate Wakeup Reset 102
13.1.8 Stop Wakeup Reset 102
13.2 Identifying Reset Sources 102
13.3 Register List 103
14. Device Security 104
14.1 Features 104
14.2 How It Works 104
14.2.1 Device Security 104
14.2.2 Flash Security 105

Section D: Digital System 106

15. Serial Communications Block (SCB) 107
15.1 Features 107
15.2 Serial Peripheral Interface (SPI) 107
15.2.1 Features 107
15.2.2 General Description 108
15.2.3 SPI Modes of Operation 109
15.2.4 Using SPI Master to Clock Slave 113
15.2.5 Easy SPI Protocol 113
15.2.6 SPI Registers 115
15.2.7 SPI Interrupts 116

15.2.8 Enabling and Initializing SPI....116
15.2.9 Internally and Externally Clocked SPI Operations....118
15.3 UART....121
15.3.1 Features....121
15.3.2 General Description....121
15.3.3 UART Modes of Operation....121
15.3.4 UART Registers....128
15.3.5 UART Interrupts....129
15.3.6 Enabling and Initializing UART....129
15.4 Inter Integrated Circuit (I2C)....131
15.4.1 Features....131
15.4.2 General Description....131
15.4.3 Terms and Definitions....132
15.4.4 I2C Modes of Operation....132
15.4.5 Easy I2C (EZI2C) Protocol....134
15.4.6 I2C Registers....135
15.4.7 I2C Interrupts....136
15.4.8 Enabling and Initializing the I2C....136
15.4.9 Internal and External Clock Operation in I2C....137
15.4.10 Wake up from Sleep....139
15.4.11 Master Mode Transfer Examples....140
15.4.12 Slave Mode Transfer Examples....142
15.4.13 EZ Slave Mode Transfer Example....144
15.4.14 Multi-Master Mode Transfer Example....146
16. Universal Digital Blocks (UDB)....148
16.1 Features....148
16.2 How It Works....149
16.2.1 PLDs....149
16.2.2 Datapath....151
16.2.3 Status and Control Module....170
16.2.4 Reset and Clock Control Module....177
16.2.5 UDB Addressing....185
16.2.6 System Bus Access Coherency....185
16.3 Port Adapter Block....186
16.3.1 PA Data Input Logic....186
16.3.2 PA Port Pin Clock Multiplexer Logic....187
16.3.3 PA Data Output Logic....187
16.3.4 PA Output Enable Logic....188
16.3.5 PA Clock Multiplexer....189
16.3.6 PA Reset Multiplexer....189
17. Controller Area Network (CAN)....190
17.1 Features....190
17.2 Block Diagram....191
17.3 CAN Message Frames....191
17.3.1 Data Frames....191
17.3.2 Remote Frame....192
17.3.3 Error Frame....193
17.3.4 Overload Frame....193
17.4 Transmitting Messages in CAN....193
17.4.1 Message Arbitration....194
17.4.2 Message Transmit Process....194
17.4.3 Message Abort....194

17.4.4 Single Shot Transmission....195
17.4.5 Transmitting Extended Data Frames....195
17.5 Receiving Messages in CAN....195
17.5.1 Message Receive Process....195
17.5.2 Acceptance Filter....196
17.5.3 DeviceNet Filtering....197
17.5.4 Filtering of Extended Data Frames....198
17.5.5 Receiver Message Buffer Linking....198
17.6 Remote Frames....199
17.6.1 Transmitting a Remote Frame by the Requesting Node....199
17.6.2 Receiving a Remote Frame....199
17.6.3 RTR Auto Reply....200
17.6.4 Remote Frames in Extended Format....200
17.7 Time-Triggered CAN....200
17.7.1 TTCAN Timer....200
17.8 Bit Time Configuration....200
17.8.1 Allowable Bit Rates and System Clock (SYSCLK)....200
17.8.2 Setting Bit Rate TSEG1 and TSEG2....201
17.9 Errors and Interrupts in CAN....203
17.9.1 Types of Errors....203
17.9.2 Error Capture Register....203
17.9.3 Error States in CAN....204
17.9.4 Interrupt Sources in CAN....204
17.10 Operating Modes in CAN....206
17.10.1Run/Stop Mode....206
17.10.2Listen Only Mode....206
17.10.3Loopback Test Mode....206
18. Universal Serial Bus (USB)....207
18.1 Features....207
18.2 Block Diagram....208
18.2.1 USB Physical Layer (USB PHY)....208
18.2.2 Serial Interface Engine (SIE)....208
18.2.3 Arbiter....208
18.3 How it Works....209
18.3.1 USB Physical Layer (USB PHY)....209
18.3.2 Endpoints....212
18.3.3 Transfer Types....212
18.3.4 Interrupts....212
18.3.5 DMA Support....214
18.4 Logical Transfer Modes....215
18.4.1 Store and Forward Mode....217
18.4.2 Cut Through Mode....219
18.4.3 Control Endpoint Logical Transfer....221
18.5 Register Summary....223
19. Timer, Counter, and PWM....225
19.1 Features....225
19.2 Block Diagram....226
19.2.1 Enabling and Disabling Counter in TCPWM Block....226
19.2.2 Clocking....226
19.2.3 Events Based on Trigger Inputs....227
19.2.4 Output Signals....228
19.2.5 Power Modes....229

19.3 Modes of Operation 230
19.3.1 Timer Mode 231
19.3.2 Capture Mode 234
19.3.3 Quadrature Decoder Mode 236
19.3.4 Pulse Width Modulation Mode 239
19.3.5 Pulse Width Modulation with Dead Time Mode 243
19.3.6 Pulse Width Modulation Pseudo-Random Mode 245
19.4 TCPWM Registers 247

Section E: Analog System 248

20. Precision Reference 249
20.1 Features 249
20.2 Block Diagram 249
20.3 How it Works 250
20.3.1 Precision Bandgap 250
20.3.2 Trim Buffer 250
20.3.3 Low-Power Buffers 250
20.3.4 Current Mirrors 251
20.3.5 Temperature-Controlled Voltage Generator 251
20.3.6 Temperature-Controlled Current Generator 251
20.4 Configuration 251

21. SAR ADC 252
21.1 Features 252
21.2 Block Diagram 253
21.3 How it Works 254
21.3.1 SAR ADC Core 254
21.3.2 SARMUX 257
21.3.3 SARREF 264
21.3.4 SARSEQ 265
21.3.5 Interrupt 268
21.3.6 Trigger 270
21.3.7 SAR ADC Status 271
21.3.8 Low-Power Mode 271
21.3.9 System Operation 272
21.3.10 Register Mode 274
21.3.11 DSI Mode 277
21.3.12 Analog Routing Configuration Example 280
21.3.13 Temperature Sensor Configuration 283
21.4 Registers 284

22. Low-Power Comparator 285
22.1 Features 285
22.2 Block Diagram 286
22.3 How It Works 286
22.3.1 Input Configuration 286
22.3.2 Output and Interrupt Configuration 286
22.3.3 Power Mode and Speed Configuration 288
22.3.4 Hysteresis 289
22.3.5 Wakeup from Low-Power Modes 289
22.3.6 Comparator Clock 289
22.3.7 Offset Trim 289
22.4 Register Summary 290

23. Continuous Time Block mini (CTBm) 291
23.1 Features 291
23.2 Block Diagram 292
23.3 How It Works 292
23.3.1 Power Mode Configuration 293
23.3.2 Output Strength Configuration 293
23.3.3 Compensation 294
23.3.4 Switch Control 294
23.4 Register Summary 300
24. LCD Direct Drive 301
24.1 Features 301
24.2 LCD Segment Drive Overview 301
24.2.1 Drive Modes 302
24.2.2 Recommended Usage of Drive Modes 310
24.2.3 Digital Contrast Control 310
24.3 Block Diagram 311
24.3.1 How it Works 311
24.3.2 High-Speed and Low-Speed Master Generators 311
24.3.3 Multiplexer and LCD Pin Logic 312
24.3.4 Display Data Registers 312
24.4 Register List 312
25. CapSense 313
26. Temperature Sensor 314
26.1 Features 314
26.2 How it Works 314
26.3 Temperature Sensor Configuration 315
26.4 Algorithm 316
26.5 Registers 316

Section F: Program and Debug 317

27. Program and Debug Interface 318
27.1 Features 318
27.2 Functional Description 318
27.3 Serial Wire Debug (SWD) Interface 319
27.3.1 SWD Timing Details 320
27.3.2 ACK Details 320
27.3.3 Turnaround (Trn) Period Details 320
27.4 Cortex-M0 Debug and Access Port (DAP) 321
27.4.1 Debug Port (DP) Registers 321
27.4.2 Access Port (AP) Registers 321
27.5 Programming the PSoC 4 Device 322
27.5.1 SWD Port Acquisition 322
27.5.2 SWD Programming Mode Entry 322
27.5.3 SWD Programming Routines Executions 322
27.6 PSoC 4 SWD Debug Interface 323
27.6.1 Debug Control and Configuration Registers 323
27.6.2 Breakpoint Unit (BPU) 323
27.6.3 Data Watchpoint (DWT) 323
27.6.4 Debugging the PSoC 4 Device 323
27.7 Registers 324
28. Nonvolatile Memory Programming 325
28.1 Features 325

28.2 Functional Description 325
28.3 System Call Implementation 326
28.4 Blocking and Non-Blocking System Calls 326
28.4.1 Performing a System Call 326
28.5 System Calls 327
28.5.1 Silicon ID 327
28.5.2 Configure Clock 328
28.5.3 Load Flash Bytes 328
28.5.4 Write Row 329
28.5.5 Program Row 330
28.5.6 Erase All 330
28.5.7 Checksum 331
28.5.8 Write Protection 332
28.5.9 Non-Blocking Write Row 333
28.5.10 Non-Blocking Program Row 333
28.5.11 Resume Non-Blocking 334
28.6 System Call Status 335
28.7 Non-Blocking System Call Pseudo Code 336

Glossary **338**

Section A: Overview

This section encompasses the following chapters:

- Introduction chapter on page 15
- Getting Started chapter on page 20
- Document Construction chapter on page 22

Document Revision History

Revision	Issue Date	Description of Change
**	July 27, 2015	Initial version of PSoC 4200L TRM
*A	November 27, 2015	Initial public release of the PSoC 4200L TRM
*B	March 23, 2017	Updated logo and copyright. Added the missing SCB chapter to the TRM.
*C	August 02, 2018	Updated the features section in Continuous Time Block mini (CTBm) chapter on page 291 Updated section 8.3.6 Peripheral Clock Divider Configuration
*D	July 30, 2019	Updated TCPWM block diagram. Fixed typo in datasheet link Updated CapSense chapter
*E	June 03, 2024	Fixed the broken links.

1. Introduction

PSoC® 4 is a programmable embedded system controller with an Arm® Cortex®-M0 CPU. It combines programmable analog, programmable interconnect, user-programmable digital logic, and commonly used fixed-function peripherals with a high-performance Arm Cortex-M0 subsystem. The PSoC 4200L is an enhanced version of the PSoC 4200 family and is upward-compatible with larger members of PSoC 4.

PSoC 4 devices have these characteristics:

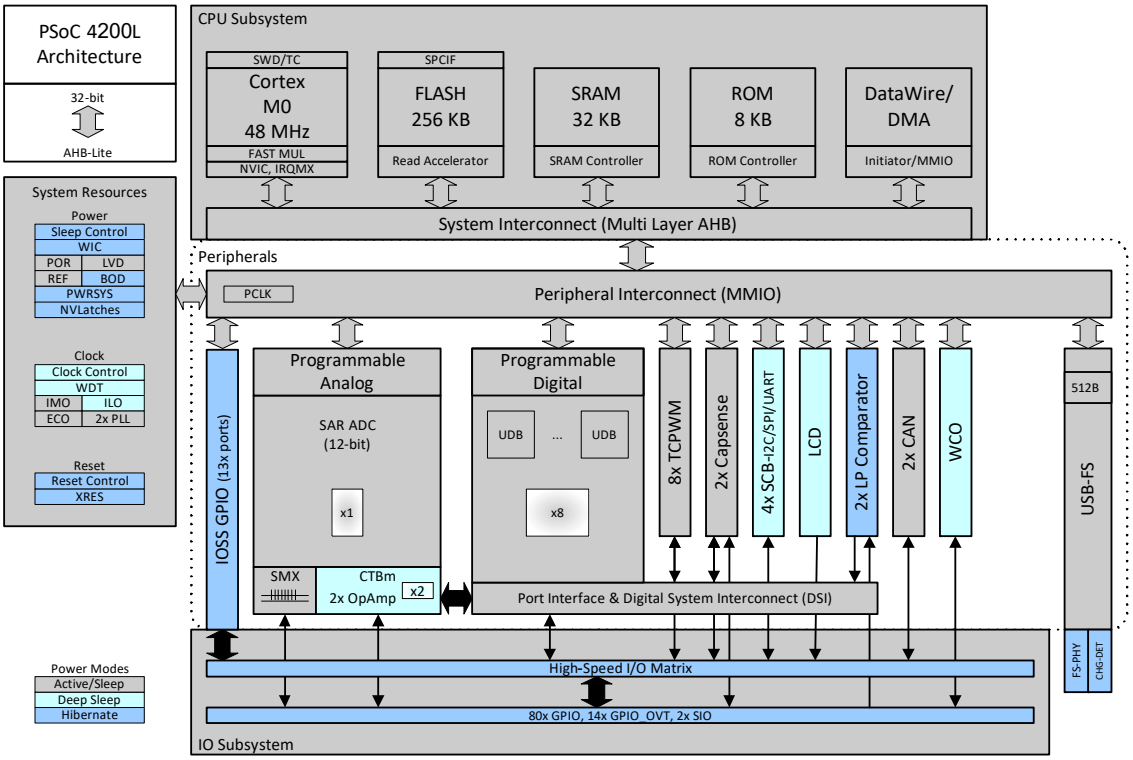
- High-performance, 32-bit single-cycle Cortex-M0 CPU core
- Fixed-function and configurable digital blocks
- Programmable digital logic
- High-performance analog system
- Flexible and programmable interconnect
- Full-Speed USB
- Capacitive touch sensing (CapSense®)
- Low-power operating modes – Sleep, Deep-Sleep, Hibernate, and Stop modes

This document describes each functional block of the PSoC 4200L device in detail. This information will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of the PSoC 4200L architecture.

Figure 1-1. PSoC 4200L Family Block Diagram



1.2 Features

The PSoC 4200L family has these major components:

- 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)
- Eight center-aligned pulse-width modulators (PWMs) with complementary, dead-band programmable outputs
- Twelve-bit SAR ADC (with a sampling rate of 1 Msps) with hardware sequencing for multiple channels
- Up to four opamps that can be used for analog signal conditioning and as a comparator
- Two low-power comparators
- Four serial communication blocks (SCB) that can work as SPI, UART, I^2C, and local interconnect network (LIN) slave serial communication channels
- Two controller area network (CAN) blocks
- Up to eight programmable logic blocks, known as universal digital blocks (UDBs)
- CapSense
- Segment LCD direct drive
- Low-power operating modes: Sleep, Deep-Sleep, Hibernate, and Stop
- Programming and debugging system through serial wire debug (SWD)
- Fully supported by PSoC Creator™ IDE tool

1.3 CPU System

1.3.1 Processor

The heart of the PSoC 4 is a 32-bit Cortex-M0 CPU core running up to 48 MHz for PSoC 4200L. It is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set. This instruction set enables fully compatible binary upward migration of the code to higher performance processors such as Cortex M3 and M4.

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

1.3.2 Interrupt Controller

The CPU subsystem includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode. The Cortex-M0 CPU of PSoC 4 implements a non-maskable interrupt (NMI) input, which can be tied to digital routing for general-purpose use.

1.3.3 Direct Memory Access

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

1.4 Memory

The PSoC 4 memory subsystem consists of flash and SRAM. A supervisory ROM, containing boot and configuration routines, is also present.

1.4.1 Flash

The PSoC 4 has a flash module, with a flash accelerator tightly coupled to the CPU, to improve average access times from the flash block. The flash accelerator delivers 85 percent of single-cycle SRAM access performance on an average.

1.4.2 SRAM

The PSoC 4 provides SRAM, which is retained during Hibernate mode.

1.5 System-Wide Resources

1.5.1 Clocking System

The clocking system for the PSoC 4 device consists of the internal main oscillator (IMO) and internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock, 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 device. The default IMO frequency is 24 MHz and it can be adjusted between 3 MHz and 48 MHz in steps of 1 MHz. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a low-power, less accurate oscillator and is used 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 used to generate the clock derivatives for the functional blocks instead of the IMO.

The ECO generates a highly accurate clock of up to 33 MHz frequency using an external crystal. There are also two phase-locked loops (PLLs), which can be used to generate frequencies up to 48 MHz.

The WCO is used as a source for LFCLK. WCO is used to accurately maintain the time interval during Deep Sleep mode. Similar to the ILO, WCO is also available in all modes, except Hibernate and Stop modes.

1.5.2 Power System

PSoC 4 provides multiple power supply domains – V_{DDD} to power digital section, V_{DDA} for noise isolation of analog section, and V_{DDIO} to allow separate voltage levels for one bank of I/Os. V_{DDD} and V_{DDA} should be shorted externally, whereas V_{DDIO} can be independently controlled.

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

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

1.5.3 GPIO

Every GPIO in PSoC 4 has the following characteristics:

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

PSoC 4200L also has 14 over-voltage tolerant pins, which enable I2C Fast Mode power down specification compliance and have the ability to connect to higher voltage buses while operating at lower V_{DD}.

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

PSoC 4200L has two special I/Os (SIOs). These pins allow hot-swap insertion and configurable input thresholds and output switching voltage levels.

1.6 Programmable Digital

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

The UDB arrays 4200L enable custom logic or additional timers/PWMs and communication interfaces such as I^2C, SPI, I2S, and UART.

1.7 Fixed-Function Digital

1.7.1 Timer/Counter/PWM Block

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

1.7.2 Serial Communication Blocks

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

The features of each SCB include:

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

1.7.3 Controller Area Network

Two CAN blocks are provided in PSoC 4200L, which support CAN 2.0A and 2.0B. These blocks have 16 receive buffers each with its own message filter, as well as eight transmit buffers. The PHY interface supports the industry standard Philips CAN PHY.

1.8 Analog System

1.8.1 SAR ADC

The PSoC 4200L has a configurable 12-bit 1-Msps SAR ADC. The ADC provides three internal voltage references (V_{DDA}, $V_{DDA}/2$, and V_{REF}) and an external reference through a GPIO pin. The SAR hardware sequencer is available, which scans multiple channels without CPU intervention.

1.8.2 Continuous Time Block mini

The Continuous Time Block mini (CTBm) provides continuous time functionality at the entry and exit points of the analog subsystem. The CTBm has two highly configurable and high-performance opamps with a switch routing matrix. The opamps can also work in comparator mode. PSoC 4200L has two CTBm blocks.

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

1.8.3 Low-Power Comparators

The PSoC 4 has a pair of low-power comparators, which can operate in all device power modes. This functionality allows the CPU and other system blocks to be disabled while retaining the ability to monitor external voltage levels during low-power modes. Two input voltages can both come from pins, or one from an internal signal through the AMUX-BUS.

1.9 Special Function Peripherals

1.9.1 LCD Segment Drive

The PSoC 4 has an LCD controller, which can drive up to eight commons and every GPIO can be configured to drive common or segment. It uses full digital methods (digital correlation and PWM) to drive the LCD segments, and does not require generation of internal LCD voltages.

1.9.2 CapSense

PSoC 4 devices have the CapSense feature, which allows you to use the capacitive properties of your fingers to toggle buttons and sliders. CapSense functionality is supported on all GPIO pins in PSoC 4 through a CapSense Sigma-Delta (CSD) block. The CSD also provides waterproofing capability. The PSoC 4200L device has two such CapSense blocks.

1.9.2.1 IDACs and Comparator

The CapSense block has two IDACs and a comparator with a 12-V reference, which can be used for general purposes, if CapSense is not used. PSoC 4200L has four IDACs and two comparators for general-purpose use. These are part of the two CapSense blocks.

1.10 Program and Debug

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

2. Getting Started

2.1 PSoC 4 resources

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

- Overview: PSoC portfolio
- Product Selectors: PSoC 4 MCU
- Application Notes cover a broad range of topics, from basic to advanced level, and include the following:
  - AN79953: Getting Started with PSoC 4
  - AN88619: PSoC 4 MCU Hardware Design Considerations
  - AN73854: PSoC Creator - Introduction to Bootloaders
  - AN89610: Arm® Cortex® Code Optimization
  - AN86233: PSoC 4 MCU low-power modes and power reduction techniques
  - AN57821: PSoC 3, PSoC 4, and PSoC 5LP Mixed-Signal Circuit Board Layout Considerations
  - AN85951: PSoC 4 and PSoC 6 MCU CapSense Design Guide
- Code Examples demonstrate product features and usage, and are also available on GitHub repositories
- Datasheets describe and provide electrical specifications for each family
- PSoC 4 MCU Programming Specification provides the information necessary to program PSoC 4 MCU non-volatile memory
- Development Tools
  - PSoC Creator is a free Windows-based IDE. It enables concurrent hardware and firmware design of PSoC 3, PSoC 4, PSoC 5LP, and PSoC 6 MCU based systems. Applications are created using schematic capture and over 150 pre-verified, production-ready peripheral Components
  - CY8CKIT-046 PSoC 4 L-Series Pioneer Kit, is an easy-to-use and inexpensive development platform. This kit includes connectors for Arduino compatible shields.
  - MiniProg4 (CY8CKIT-005-A) and MiniProg3 (CY8CKIT-002) all-in-one development programmers and debuggers.
  - PSoC 4 MCU CAD libraries provide footprint and schematic support for common tools. IBIS models are also available.
- Training videos are available on a wide range of topics including the PSoC MCUs.

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

2.2 PSoC Creator

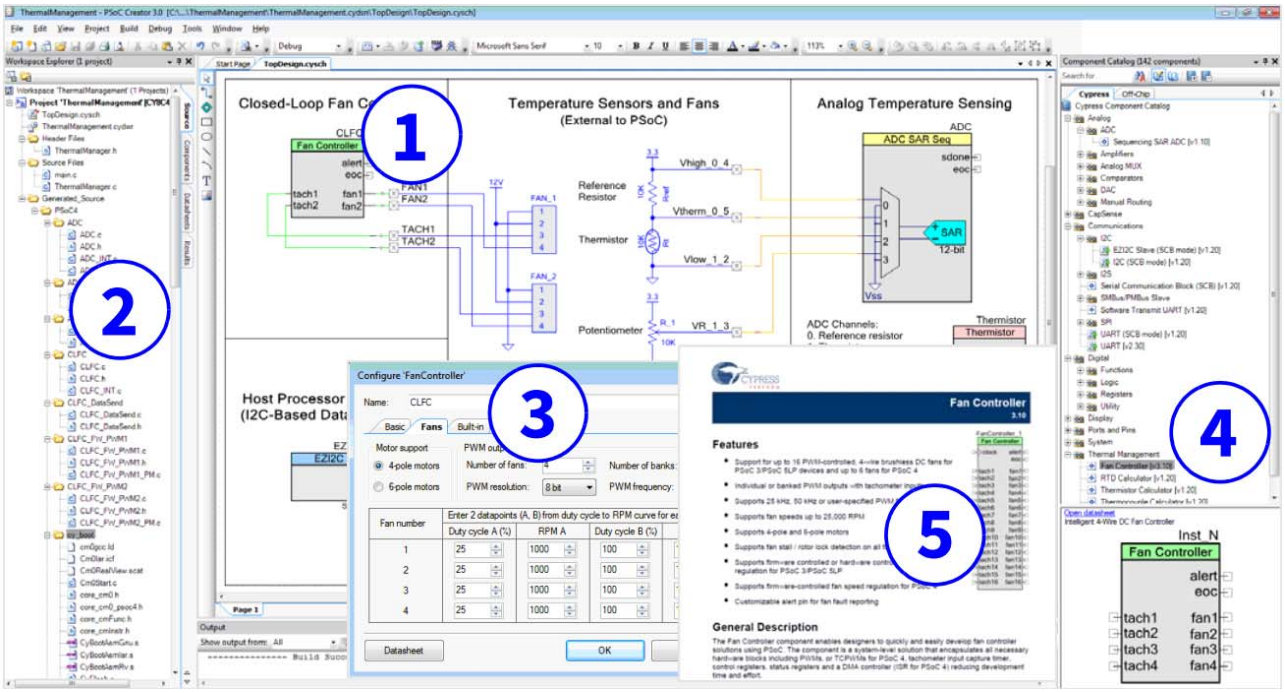
PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables you to design hardware and firmware systems concurrently, based on PSoC 4 MCU. As Figure 2-1 shows, with PSoC Creator you can:

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

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

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

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



3. Document Construction

This document includes the following sections:

- Section B: CPU System on page 25
- Section C: System Resources Subsystem (SRSS) on page 54
- Section D: Digital System on page 106
- Section E: Analog System on page 248
- Section F: Program and Debug on page 317

3.1 Major Sections

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

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

3.2 Documentation Conventions

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

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

3.2.1 Register Conventions

Register conventions are detailed in the PSoC 4200L Registers TRM.

3.2.2 Numeric Naming

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

3.2.3 Units of Measure

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

Table 3-1. Units of Measure

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

3.2.4 Acronyms

This table lists the acronyms used in this document

Table 3-2. Acronyms

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

Table 3-2. Acronyms *(continued)*

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

Table 3-2. Acronyms *(continued)*

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

Table 3-2. Acronyms *(continued)*

Acronym	Definition
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SYSCLK	system clock
SWD	single wire debug
TC	terminal count
TCPWM	timer, counter, PWM
TD	transaction descriptors
UART	universal asynchronous receiver/transmitter
UDB	universal digital block
USB	universal serial bus
USBIO	USB I/O
WCO	watch crystal oscillator
WDT	watchdog timer
WDR	watchdog reset
XRES	external reset
XRES_N	external reset, active low

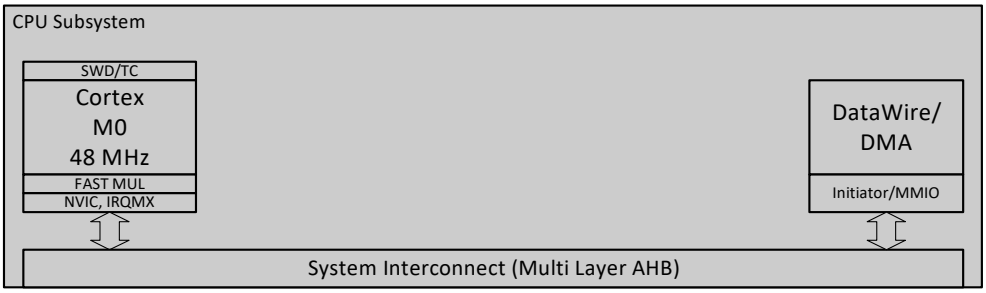
Section B: CPU System

This section encompasses the following chapters:

- Cortex-M0 CPU chapter on page 26
- DMA Controller Modes chapter on page 31
- Interrupts chapter on page 45

Top Level Architecture

CPU System Block Diagram



4. Cortex-M0 CPU

The PSoC® 4 Arm Cortex-M0 core is a 32-bit CPU optimized for low-power operation. It has an efficient three-stage pipeline, a fixed 4-GB memory map, and supports the Armv6-M Thumb instruction set. The Cortex-M0 also features a single-cycle 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

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

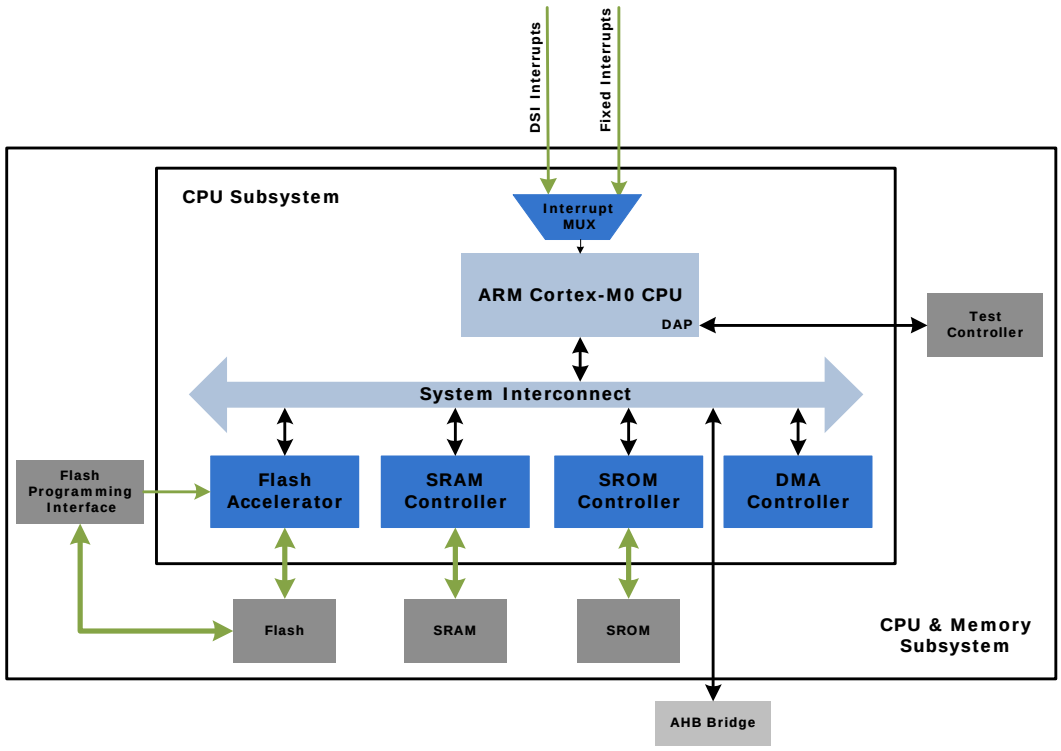
4.1 Features

The PSoC 4 Cortex-M0 has the following features:

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

4.2 Block Diagram

Figure 4-1. PSoC 4 CPU Subsystem Block Diagram



4.3 How It Works

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

The processor supports two operating modes (see “Operating Modes” on page 29). It has a single-cycle 32-bit multiplication instruction.

4.4 Address Map

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

Table 4-1. Cortex-M0 Address Map

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

4.5 Registers

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

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

Table 4-2. Cortex-M0 Registers

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

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

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

Table 4-3. Cortex-M0 PSR Bit Assignments

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

Table 4-3. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
27 – 25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23 – 6	–	–	Reserved
5 – 0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = NMI 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 47 = 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 45 for a list of exceptions.

4.6 Operating Modes

The Cortex-M0 processor supports two operating modes:

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

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

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

4.7 Instruction Set

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

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

Table 4-4. Thumb Instruction Set

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

Table 4-4. Thumb Instruction Set

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

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

4.7.1 Address Alignment

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

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

4.7.2 Memory Endianness

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

4.8 Systick Timer

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

4.9 Debug

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

5. 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 32 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.

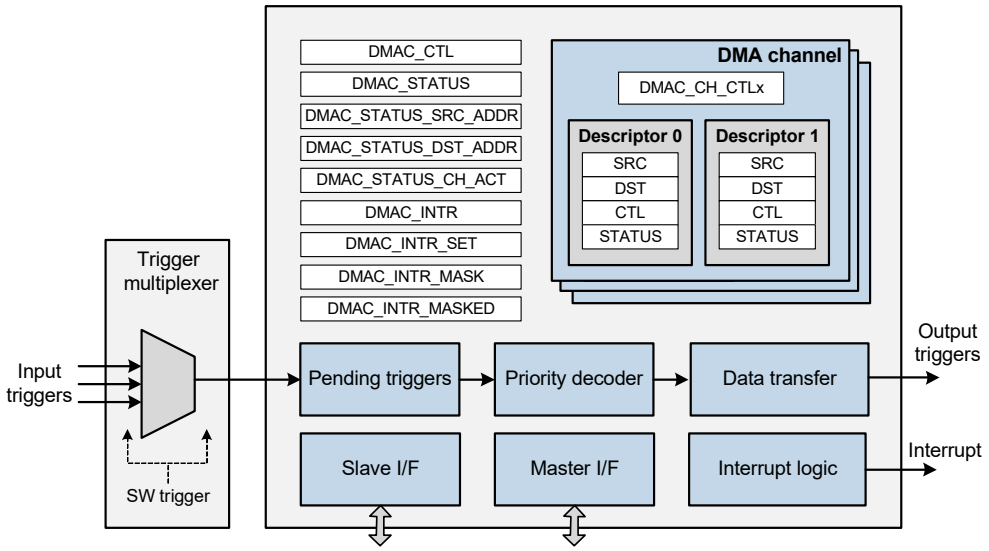
The DMA controller provides Active/Sleep functionality and is not available in the Deep-Sleep and Hibernate power modes.

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 PSoC main CPU to access the DMA controller's control/status registers and to access the descriptor structure. CPU is generally the master for this bus.

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

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

5.1.1 Trigger Sources and Multiplexing

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

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

5.1.1.1 Trigger Multiplexer

The DMA channels can have trigger inputs from different peripheral sources in the PSoC. This is routed to the individual DMA channel trigger inputs through the trigger multiplexer. In addition, the trigger multiplexer is also used to route the output triggers from the DMA to certain peripherals in PSoC.

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 device trigger input options can come from TCPWM, SAR ADC, SCB, CAN, CapSense ADC

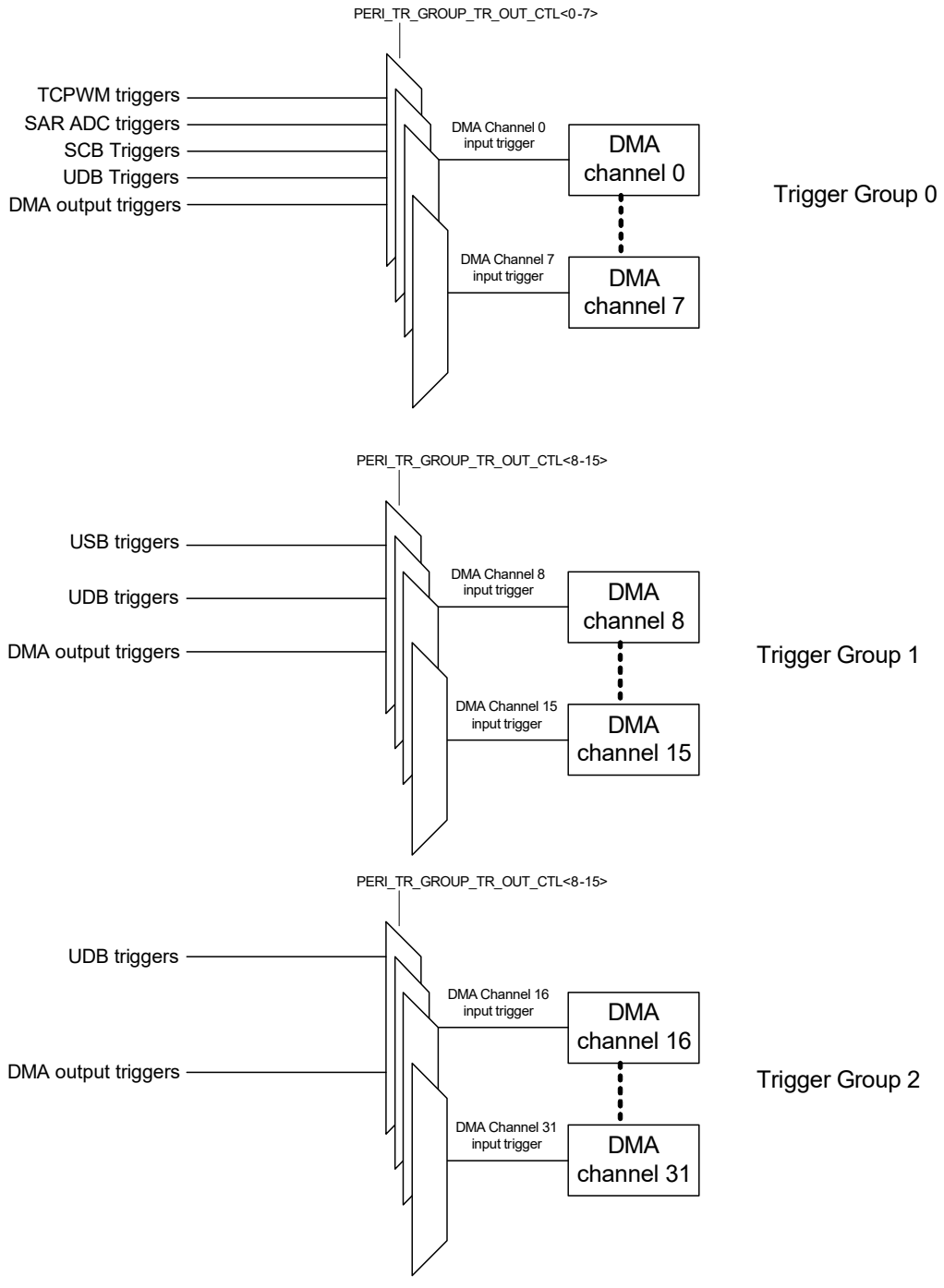
The PSoC 4200L implements four trigger groups:

- Trigger group 0 multiplexes the general-purpose peripherals such as TCPWM, SCB, UDB, and DMA trigger sources. This trigger group only provide trigger inputs to DMA channels 0 to 7.

- Trigger group 1 multiplexes the USB and UDB triggers to the DMA channels 8 to 15
- Trigger group 2 multiplexes 16 more UDB triggers to the DMA channels 16 to 31.
- Trigger group 3 implements multiplexers to route DMA trigger outputs back to the USB burst end signals.

Figure 5-2 illustrates trigger groups 0 to 2.

Figure 5-2. Trigger Groups 0 to 2



The input triggers of different DMA channels are grouped into four different trigger groups. Each groups has its own set of trigger source selections for the multiplexer. The setting in the PERI_TR_GROUP_TR_OUT_CTLx[5:0] register selects the trigger source for a specific trigger group. Table 5-3 provides details of the different source configuration for the trigger groups.

Table 5-3. Trigger Groups and Sources

PERI_TR_GROUP_TR_OUT_CTLx[5:0]	Trigger Group 0 (Trigger Sources)	Trigger Group 1 (Trigger Sources)	Trigger Group 2 (Trigger Sources)
0	Software Trigger Hardwired to zero	Software Trigger Hardwired to zero	Software Trigger Hardwired to zero
1	TCPWM 0 overflow	USB Trigger 0	UDB Trigger 16
2	TCPWM 1 overflow	USB Trigger 1	UDB Trigger 17
3	TCPWM 2 overflow	USB Trigger 2	UDB Trigger 18
4	TCPWM 3 overflow	USB Trigger 3	UDB Trigger 19
5	TCPWM 4 overflow	USB Trigger 4	UDB Trigger 20
6	TCPWM 5 overflow	USB Trigger 5	UDB Trigger 21
7	TCPWM 6 overflow	USB Trigger 6	UDB Trigger 22
8	TCPWM 7 overflow	USB Trigger 7	UDB Trigger 23
9	TCPWM 0 Compare	UDB Trigger 8	UDB Trigger 24
10	TCPWM 1 Compare	UDB Trigger 9	UDB Trigger 25
11	TCPWM 2 Compare	UDB Trigger 10	UDB Trigger 26
12	TCPWM 3 Compare	UDB Trigger 11	UDB Trigger 27
13	TCPWM 4 Compare	UDB Trigger 12	UDB Trigger 28
14	TCPWM 5 Compare	UDB Trigger 13	UDB Trigger 29
15	TCPWM 6 Compare	UDB Trigger 14	UDB Trigger 30
16	TCPWM 7 Compare	UDB Trigger 15	UDB Trigger 31
17	TCPWM 0 Underflow	DMA channel 8 trigger out	DMA channel 7 trigger out
18	TCPWM 1 Underflow	DMA channel 9 trigger out	
19	TCPWM 2 Underflow	DMA channel 10 trigger out	
20	TCPWM 3 Underflow	DMA channel 11 trigger out	
21	TCPWM 4 Underflow	DMA channel 12 trigger out	
22	TCPWM 5 Underflow	DMA channel 13 trigger out	
23	TCPWM 6 Underflow	DMA channel 14 trigger out	
24	TCPWM 7 Underflow	DMA channel 15 trigger out	
25	SAR ADC EOC		
26	SCB0 TX		
27	SCB0 RX		
28	SCB1 TX		
29	SCB1 RX		
30	SCB2 TX		
31	SCB2 RX		
32	SCB3 TX		
33	SCB3 RX		
34	UDB DSI request 0		
35	UDB DSI request 1		
36	DMA channel 0 trigger out		
37	DMA channel 1 trigger out		
38	DMA channel 2 trigger out		
39	DMA channel 3 trigger out		
40	DMA channel 4 trigger out		
41	DMA channel 5 trigger out		
42	DMA channel 6 trigger out		

Table 5-3. Trigger Groups and Sources

PERI_TR_GROUP_TR_OUT_CTLx[5:0]	Trigger Group 0 (Trigger Sources)	Trigger Group 1 (Trigger Sources)	Trigger Group 2 (Trigger Sources)
43	DMA channel 7 trigger out		
44	UDB Trigger 0		
45	UDB Trigger 1		
46	UDB Trigger 2		
47	UDB Trigger 3		
48	UDB Trigger 4		
49	UDB Trigger 5		
50	UDB Trigger 6		
51	UDB Trigger 7		

Trigger group 3 is dedicated for routing DMA output triggers back to the USB block to provide burst end signals to the USB. See the Universal Serial Bus (USB) chapter on page 207 for configuration details of trigger group 3.

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 32. 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.

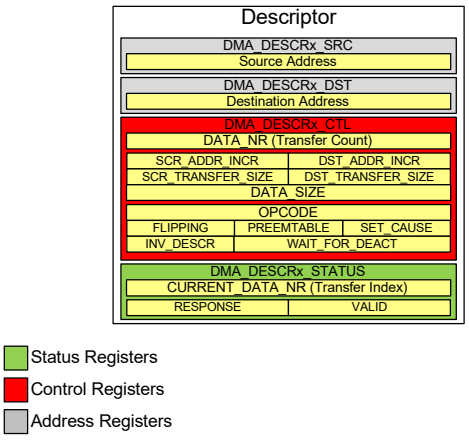
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two 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 37.

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 37.

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.

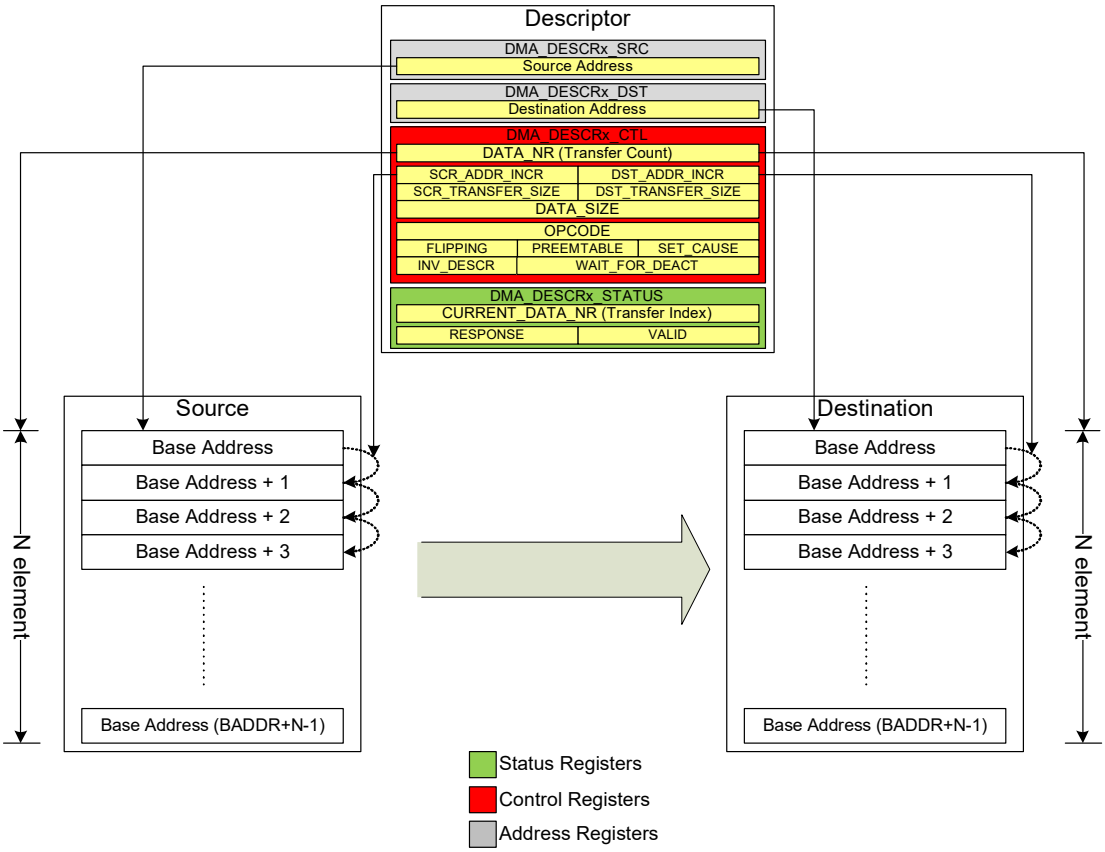
Setting Interrupt Cause (SET_CAUSE): When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

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 35. 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 PSoC 4 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 PSoC 4 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 5-4 summarizes the possible combinations of the transfer size settings and its description.

Table 5-4. 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 PSoC 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

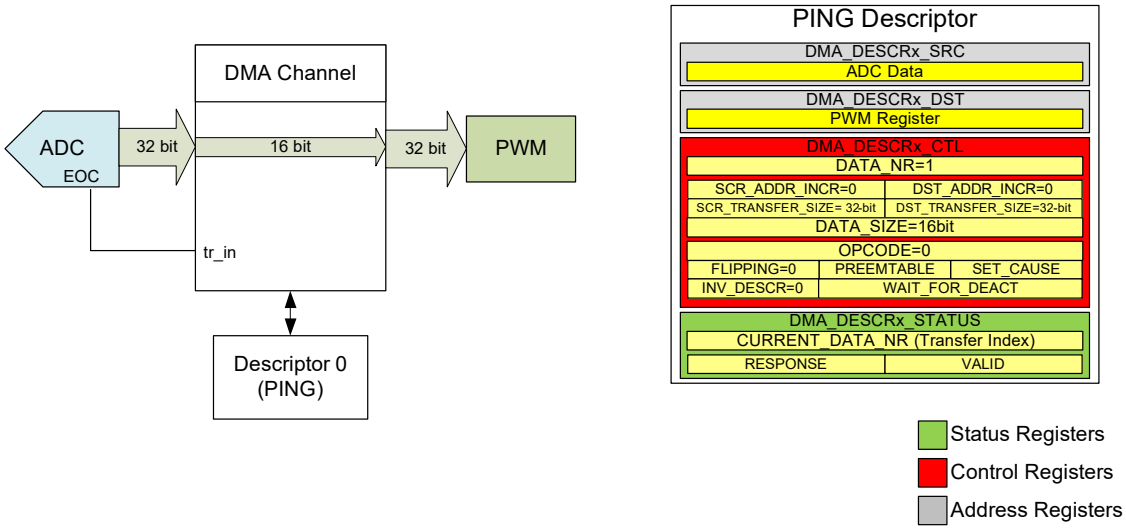
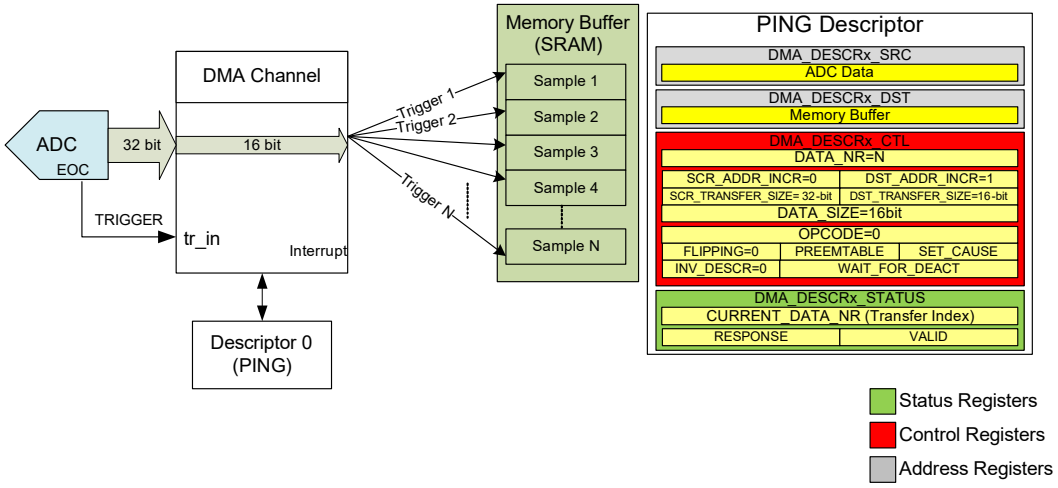


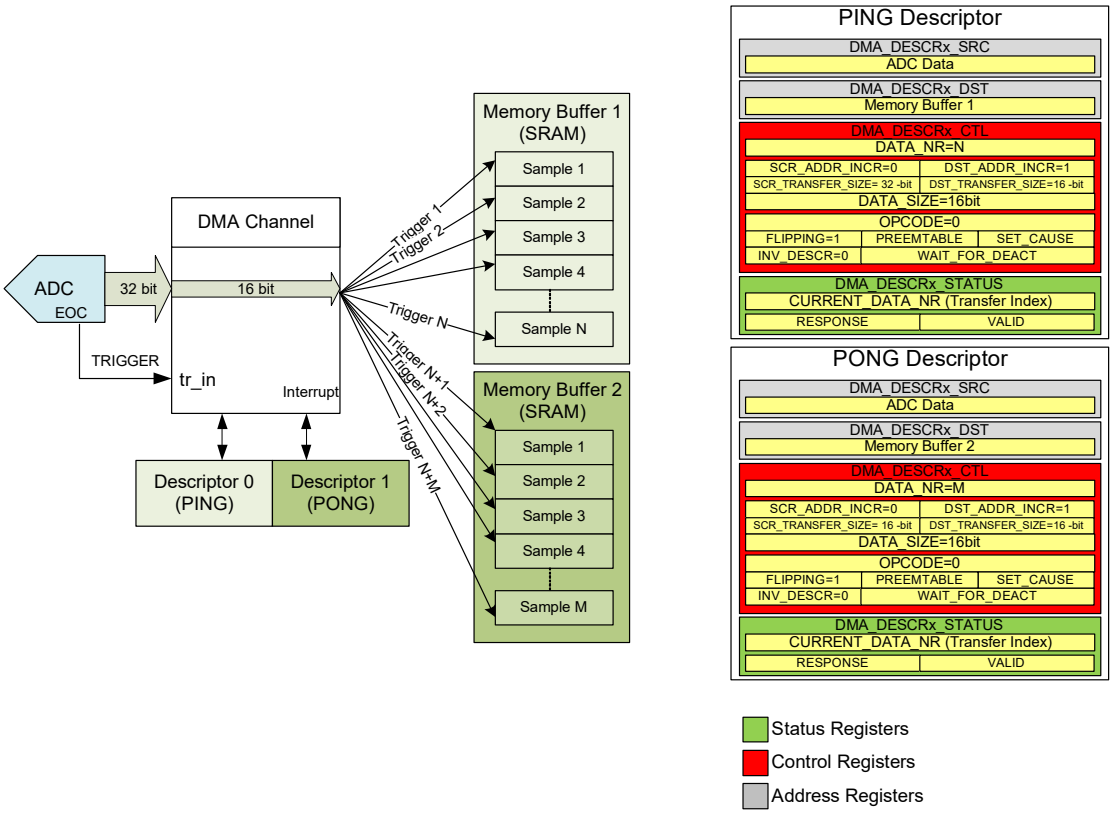
Figure 5-6 describes another use case where the data transfer is between the ADC data register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is an ADC. When the trigger is received, the transfer engine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

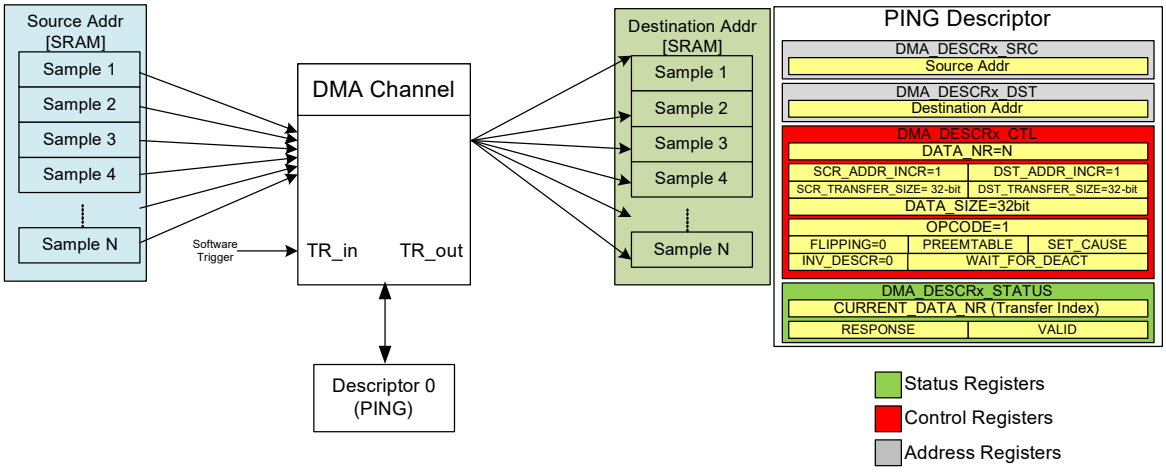


5.2.4.2 *Entire Descriptor Per Trigger (OPCODE 1)*

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

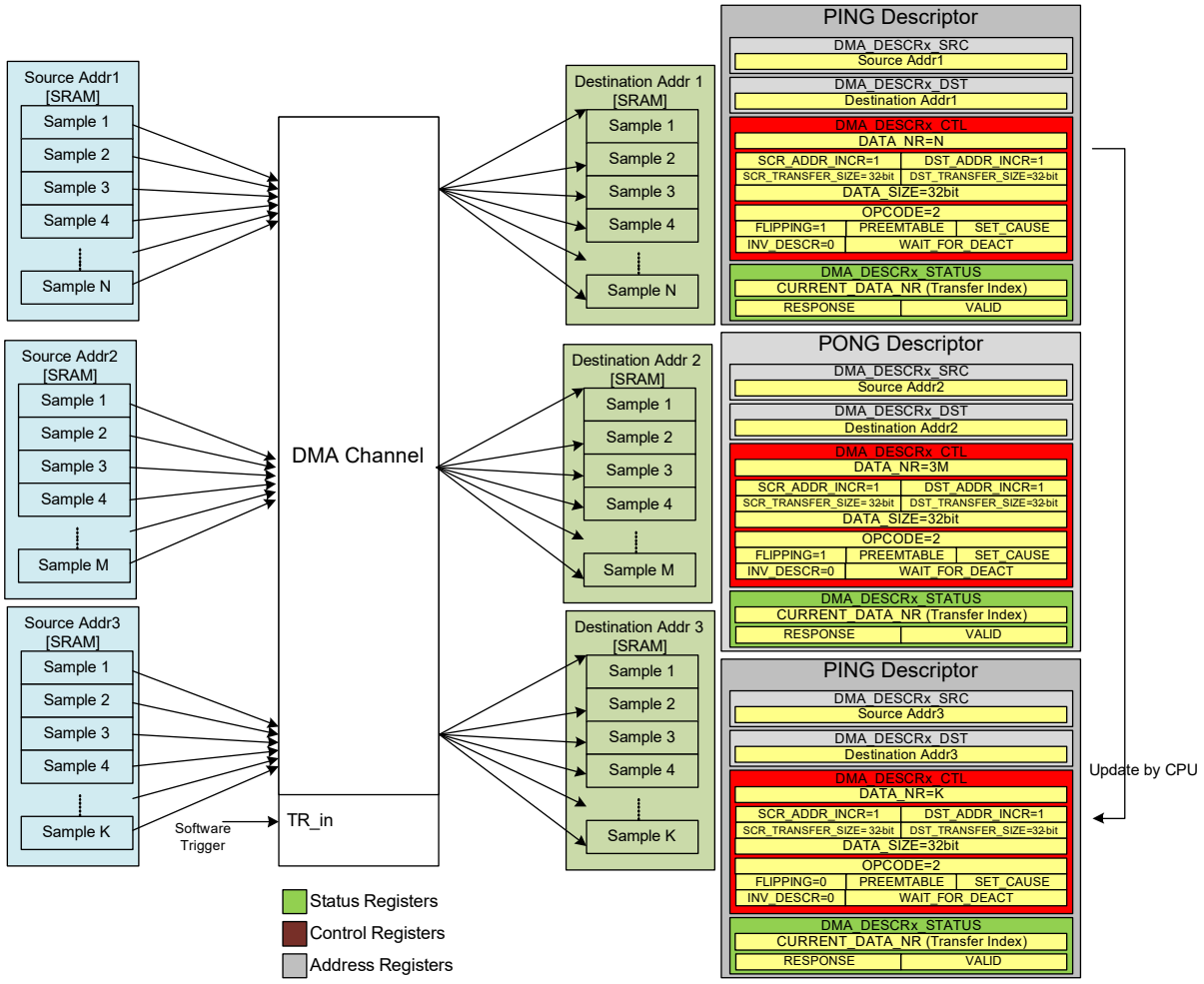


5.2.4.3 Entire Descriptor Chain Per Trigger (OPCODE 2)

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.

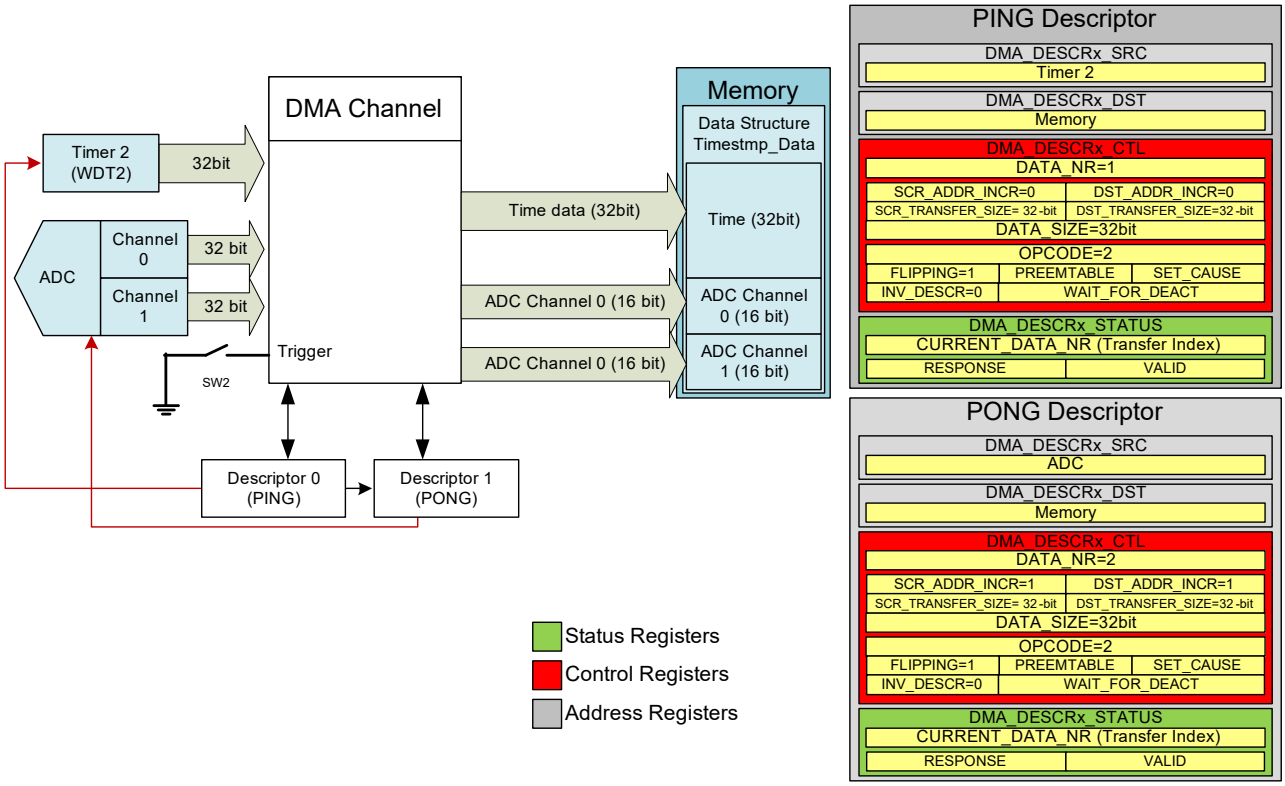
Figure 5-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

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.

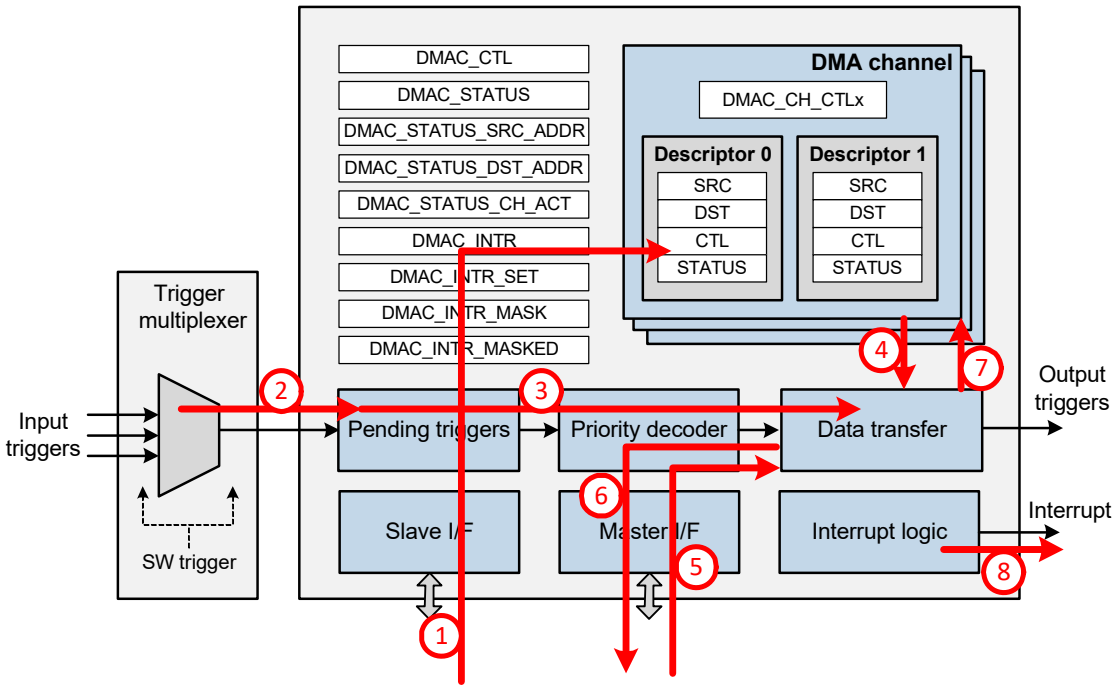
Figure 5-10. OPCODE 2: Multiple Sources to Memory



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

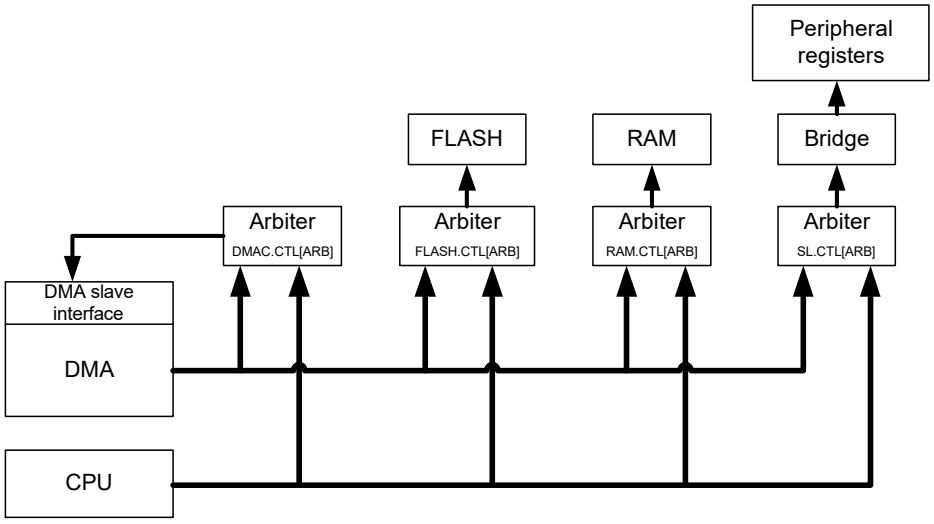
When transferring entire descriptors or chaining descriptor chains, 12 clock cycles are needed for the first data element. Subsequent elements need three cycles. This is also under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to transfer ‘N’ elements is:

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. PSoC 4 Bus Architecture

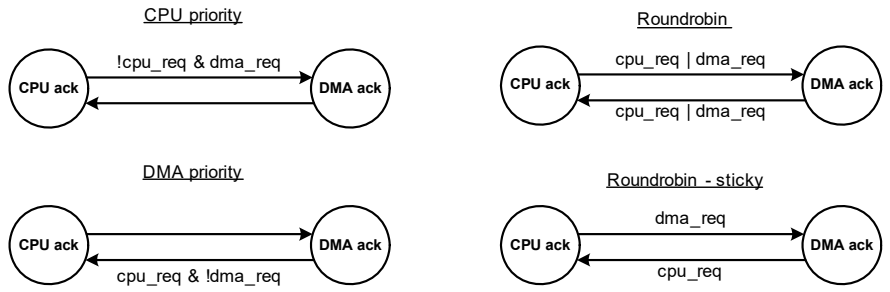


The arbitration policy for each slave can be one of the following:

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

The arbitration models are illustrated using the following diagrams.

Figure 5-13. Arbitration Models



5.5 Register List

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

6. Interrupts

The Arm Cortex-M0 (CM0) CPU in PSoC® 4 supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or interrupt service routine (ISR) being executed by the CPU. The device provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

6.1 Features

PSoC 4 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. PSoC 4 Interrupts Block Diagram

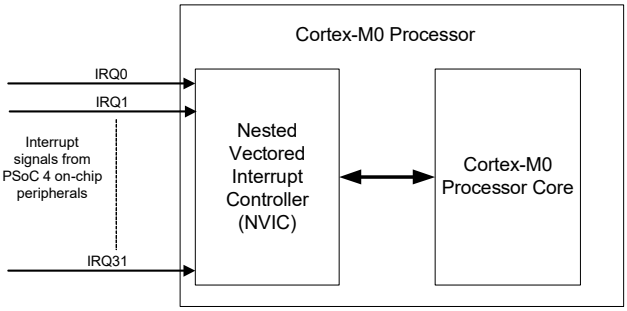


Figure 6-1 shows the interaction between interrupt signals and the Cortex-M0 CPU. PSoC 4 has 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in Figure 6-1 because they are part of CM0 core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

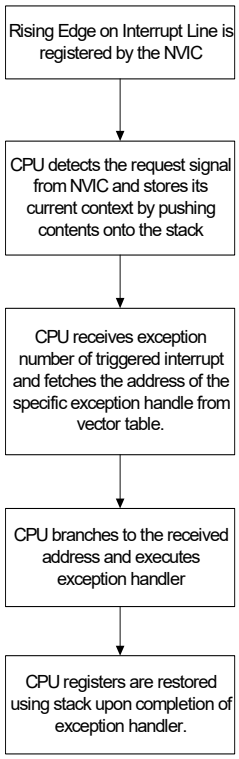
6.3 Interrupts and Exceptions - Operation

6.3.1 Interrupt/Exception Handling

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

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

Figure 6-2. Interrupt Handling When Triggered



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

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

6.3.2 Level and Pulse Interrupts

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

Figure 6-3. Level Interrupts

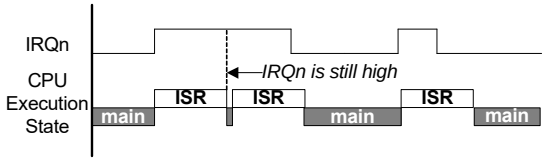


Figure 6-4. Pulse Interrupts

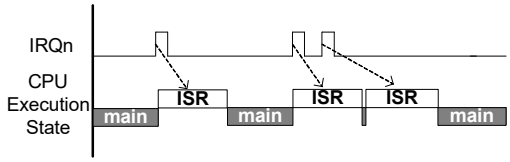


Figure 6-3 and Figure 6-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

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. The CPU fetches the appropriate address based on the exception number.

Table 6-1. Exception Vector Table

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

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

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

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

The exception sources (exception numbers 1 to 15) are explained in 6.4 Exception Sources. The exceptions marked as Reserved in Table 6-1 are not used, although they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 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 PSoC 4. It is always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of supervisory read-only memory (SROM). The boot code and other data in SROM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in Table 6-1) stores the location

of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in the SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted.

6.4.2 Non-Maskable Interrupt (NMI) Exception

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

- **NMI exception due to a hardware signal (user NMI exception):** PSoC 4 provides an option to trigger an NMI exception using a digital signal. This digital signal is referred to as irq_out[0] in Table 6-3. 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):** An NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

6.4.3 HardFault Exception

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

6.4.4 Supervisor Call (SVCall) Exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0 in PSoC 4 uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the Chip Operational Modes chapter on page 90 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in the device. The application developer must define the SVCall exception handler according to the end application requirements.

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

6.4.5 PendSV Exception

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

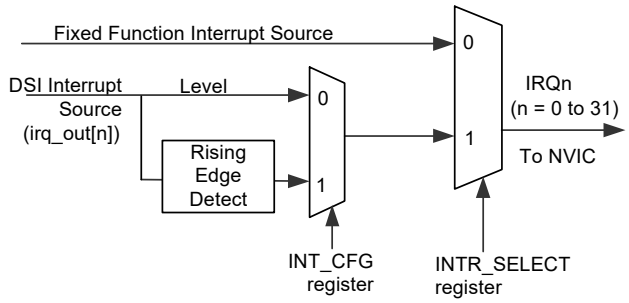
CM0 CPU in PSoC 4 supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI_15[31:30] of the System Handler Priority Register 3 (SHPR3). The SysTick exception can always be generated

in software at any instant by writing a one to the PENDSTSETb bit in the Interrupt Control State Register, CM0_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a one to the PENDSTCLR bit in the Interrupt Control State Register, CM0_ICSR.

6.5 Interrupt Sources

PSoC 4 supports 32 interrupts (IRQ0 to IRQ31 or exception numbers 16 – 47) from peripherals. The source of each interrupt is listed in . PSoC 4 provides flexible sourcing options for each interrupt line. 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.

See the I/O System chapter on page 55 for details on GPIO interrupts.

Table 6-2. List of PSoC 4 Interrupt Sources

Interrupt	Cortex-M0 Exception No.	Fixed Function Interrupt Source	DSI Interrupt Source
NMI (see "Exception Sources" on page 47)	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 13 (USB Wakeup)	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	SCB0 (Serial Communication Block 0)	udb.interrupts[1]:1
IRQ10	26	SCB1 (Serial Communication Block 1)	udb.interrupts[2]:1
IRQ11	27	SCB2 (Serial Communication Block 2)	udb.interrupts[3]:1
IRQ12	28	SCB3 (Serial Communication Block 3)	udb.interrupts[4]:1
IRQ13	29	CTBm Interrupt (all CTBms)	udb.interrupts[5]:1

Table 6-2. List of PSoC 4 Interrupt Sources

Interrupt	Cortex-M0 Exception No.	Fixed Function Interrupt Source	DSI Interrupt Source
IRQ14	30	DMA Interrupt	udb.interrupts[6]:1
IRQ15	31	SPCIF Interrupt	udb.interrupts[7]:1
IRQ16	32	LVD Interrupt	udb.interrupts[0]:2
IRQ17	33	SAR (Successive Approximation ADC)	udb.interrupts[1]:2
IRQ18	34	CSD0 (CapSense)	udb.interrupts[2]:2
IRQ19	35	CSD1 (CapSense)	udb.interrupts[3]:2
IRQ20	36	TCPWM0 (Timer/Counter/PWM 0)	udb.interrupts[4]:2
IRQ21	37	TCPWM1 (Timer/Counter/PWM 1)	udb.interrupts[5]:2
IRQ22	38	TCPWM2 (Timer/Counter/PWM 2)	udb.interrupts[6]:2
IRQ23	39	TCPWM3 (Timer/Counter/PWM 3)	udb.interrupts[7]:2
IRQ24	40	TCPWM4 (Timer/Counter/PWM 4)	udb.interrupts[0]:3
IRQ25	41	TCPWM5 (Timer/Counter/PWM 5)	udb.interrupts[1]:3
IRQ26	42	TCPWM6 (Timer/Counter/PWM 6)	udb.interrupts[2]:3
IRQ27	43	TCPWM7 (Timer/Counter/PWM 7)	udb.interrupts[3]:3
IRQ28	44	CAN0 Interrupt	udb.interrupts[4]:3
IRQ29	45	CAN1 Interrupt	udb.interrupts[5]:3
IRQ30	46	USB - Start of frame	udb.interrupts[6]:3
IRQ31	47	USB - EP1 to EP8 data	udb.interrupts[7]:3

a. Port 5 to Port 12 do not have dedicated interrupt vector numbers; these ports use IRQ 6.

6.6 Exception Priority

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

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

The registers to configure the priority of exception numbers 1 to 15 are explained in “Exception Sources” on page 47.

The priority of the 32 interrupts (IRQ0 to 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 6-3. The other bit fields in the register are not used.

Table 6-3. Interrupt Priority Register Bit Definitions

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

6.7 Enabling and Disabling Interrupts

The NVIC provides registers to individually enable and disable the 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 are 32-bit wide registers and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 6-4 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-4. Interrupt Enable/Disable Registers

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

The CM0_ISER and CM0_ICER registers are applicable only for interrupts IRQ0 to IRQ31. These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in “Exception Sources” on page 47.

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

6.8 Exception States

Each exception can be in one of the following states.

Table 6-5. Exception States

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

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

- The VECTACTIVE bits ([8:0]) in the CM0_ICSR store the exception number for the current executing exception. This value is zero if the CPU does not execute any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Program Status Register (IPSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0_ICSR indicates if a NVIC generated interrupt (IRQ0 to 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 are 32-bit wide registers and each bit corresponds to the same numbered interrupt.

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

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

Register	Operation	Bit Value	Comment
Interrupt Set-Pending Register (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 15. These 15 exceptions have their own support for pending, as explained in “Exception Sources” on page 47.

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 26 for details.

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

6.10 Interrupts and Low-Power Modes

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

The Wait For Interrupt (WFI) instruction, executed by the CM0 CPU, triggers the transition into Sleep, Deep-Sleep, and Hibernate modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 91. 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)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

6.11 Exceptions – Initialization and Configuration

This section covers the different steps involved in initializing and configuring exceptions in PSoC 4.

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

   It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents. PSoC Creator IDE uses the vector table in SRAM by default.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   b. Define the exception handler function and write the address of the function to the exception vector table. Table 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 50.
   d. Enable the exception, as explained in "Enabling and Disabling Interrupts" on page 51.

6.12 Registers

Table 6-7. List of Registers

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

6.13 Associated Documents

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

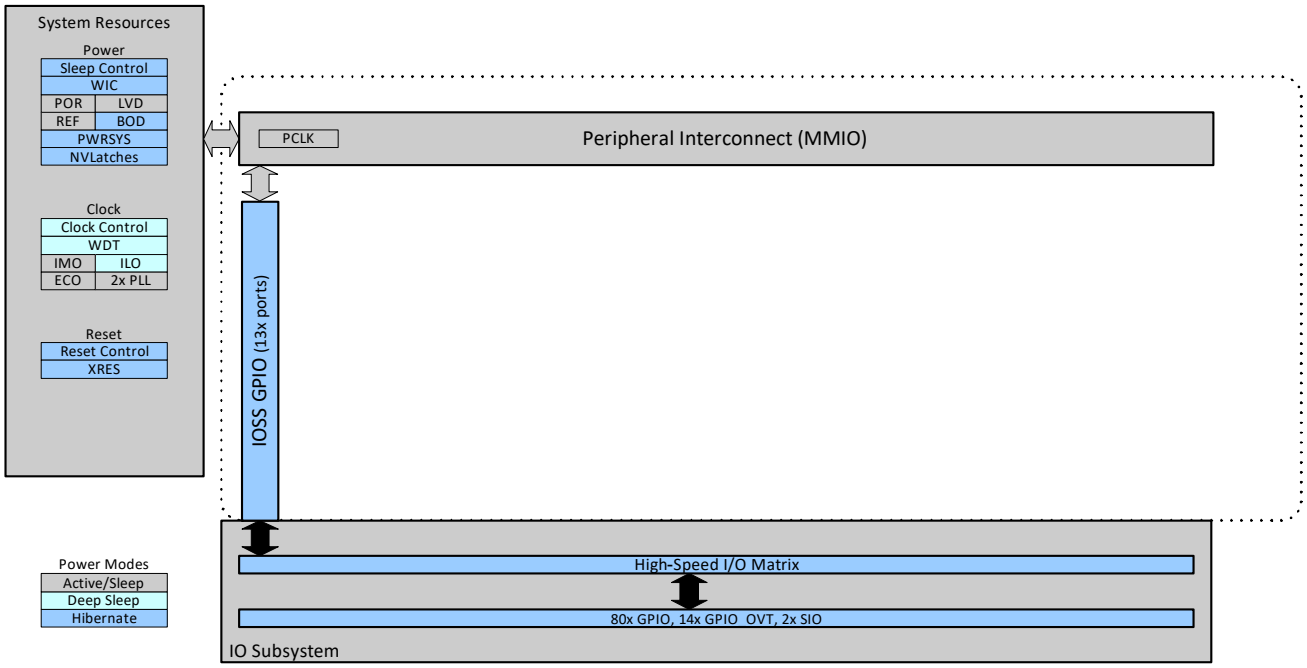
Section C: System Resources Subsystem (SRSS)

This section encompasses the following chapters:

- I/O System chapter on page 55
- Clocking System chapter on page 72
- Power Supply and Monitoring chapter on page 86
- Chip Operational Modes chapter on page 90
- Power Modes chapter on page 91
- Watchdog Timer chapter on page 97
- Reset System chapter on page 101
- Device Security chapter on page 104

Top Level Architecture

System-Wide Resources Block Diagram



7. I/O System

This chapter explains the PSoC® 4 I/O system, its features, architecture, operating modes, and interrupts. The GPIO pins in PSoC 4 are grouped into ports; a port can have a maximum of eight GPIOs. The PSoC 4200L family has a maximum of 98 GPIOs arranged in fourteen ports, which include two USB pins.

7.1 Features

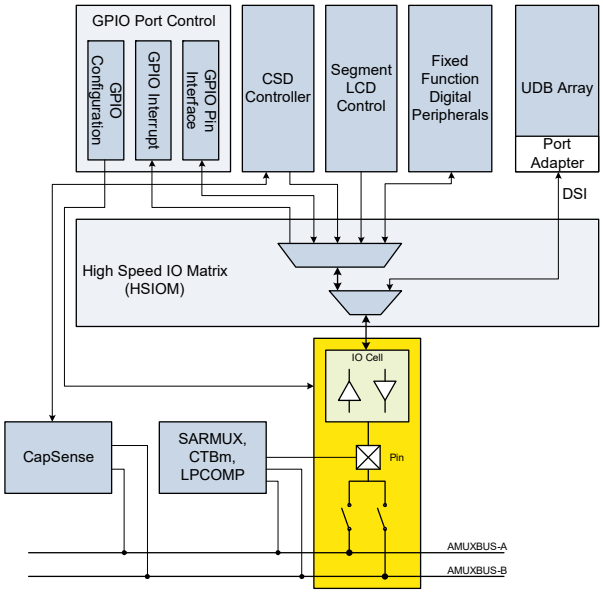
The PSoC 4 GPIOs have these features:

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

7.2 GPIO Interface Overview

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

Figure 7-1. GPIO Interface Overview

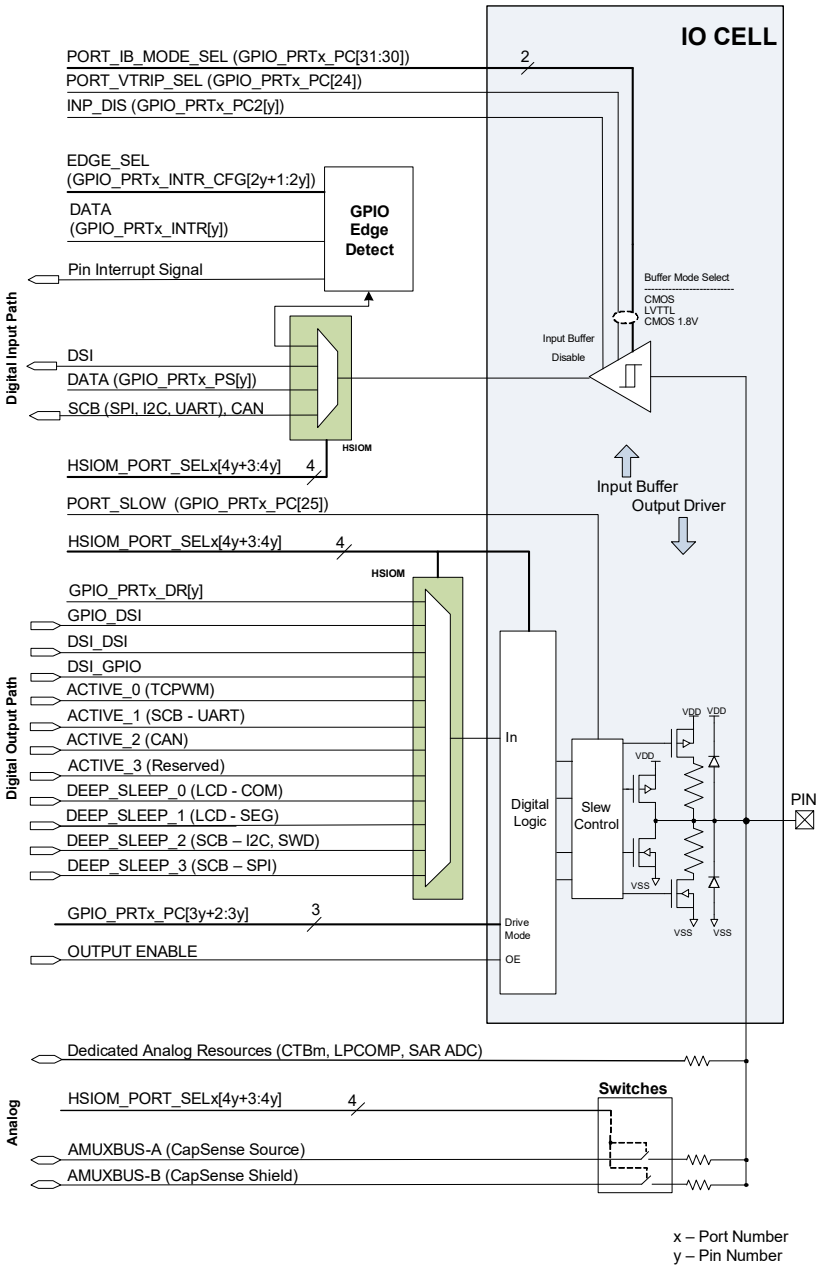


GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high input impedance and a driver for the digital output signals. The digital peripherals connect to the I/O cells via the high-speed I/O matrix (HSIOM). HSIOM contains multiplexers to connect between a peripheral selected by the user and the pin. HSIOM also bridges the connection between the digital system interconnect (DSI) and the pins. This enables routing of pin signals to the DSI-connected peripherals such as UDBs. The analog peripherals such as SAR ADC, Continuous Time Block-mini (CTBm), Low Power Comparator (LPCOMP), and CapSense are either connected to the GPIO pins directly or through the AMUX buses.

7.3 I/O Cell Architecture

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

Figure 7-2. I/O Cell Architecture in PSoC 4200L



7.3.1 Digital Input Buffer

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

- CMOS
- LVTTL
- 1.8 V CMOS

These buffer modes are selected by the PORT_VTRIP_SEL bit (GPIO_PRTx_PC[24]) and PORT_IB_MODE_SEL bits (GPIO_PRTx_PC[31:30]) of the Port Configuration register.

Table 7-1. Input Buffer Modes

PORT_VTRIP_SEL	PORT_IB_MODE_SEL	Input Buffer Mode
0b	0b or 10b	CMOS
1b	0b or 10b	LVTTL
x	1b or 11b	1.8V CMOS

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

7.3.2 Digital Output Driver

Pins are driven by the digital output driver. It consists of circuitry to implement different drive modes and slew rate control for the digital output signals. The peripheral connects to the digital output driver through the HSIOM; a particular peripheral is selected by writing to the HSIOM port select register (HSIOM_PORT_SELx). Three banks of I/Os are powered by the V_{DDD}, V_{DDA}, or V_{DDIO} source. The following table shows the ports in different banks and the I/O supply source.

Table 7-2. I/O Banks

Ports	IO Supply Source
Port 0, Port 7, Port 8, Port 9,	V_{DDD}/V_{SSD}
Port 1, Port 2, Port 5,	V_{DDA}/V_{SSA}
Port 3, Port 4, Port 6, Port 10, Port 11, Port 12	V_{DDIO}/V_{SSIO}

Each GPIO pin has ESD diodes to clamp the pin voltage to the I/O supply source. Ensure that the voltage at the pin does not exceed the I/O supply voltage $V_{DDIO}/V_{DDD}/V_{DDA}$ and drop below $V_{SSIO}/V_{SSD}/V_{SSA}$. For the absolute maximum and minimum GPIO voltage, see the device datasheet. The digital output driver can be enabled and disabled using the DSI signal from the peripheral or data register (GPIO_PRTx_DR) associated with the output pin. See 7.6 High-Speed I/O Matrix to know about the peripheral source selection for the data and to enable or disable control source selection.

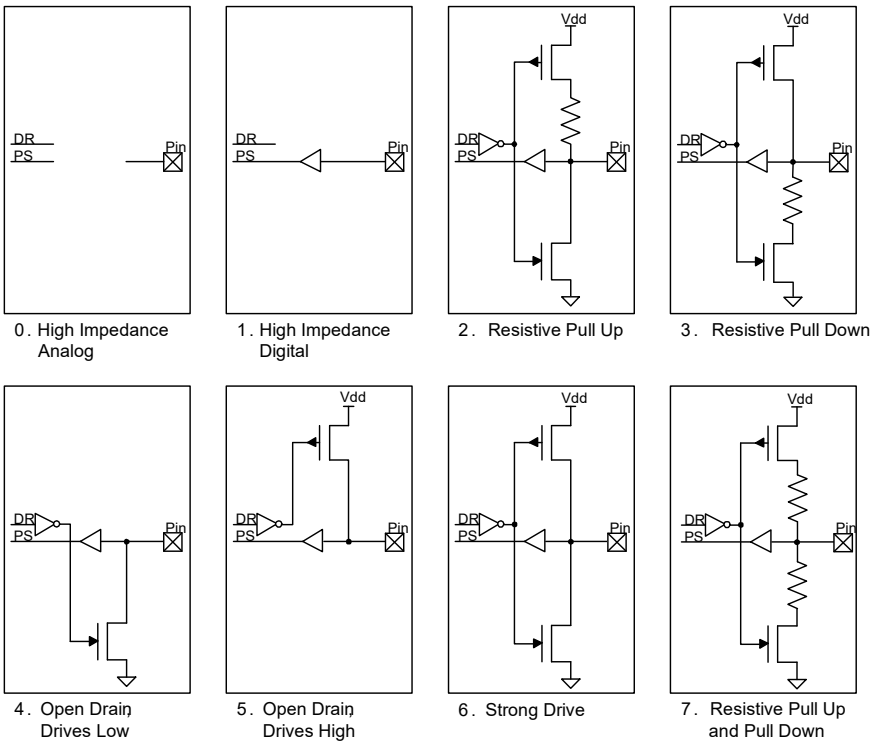
7.3.2.1 Drive Modes

Each I/O is individually configurable into one of eight drive modes using the Port Configuration register, GPIO_PRTx_PC. Table 7-3 lists the drive modes. Figure 7-2 is a simplified output driver diagram that shows the pin view based on each of the eight drive modes.

Table 7-3. Drive Mode Settings

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

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



- High-Impedance Analog

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

- High-Impedance Digital

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

- Resistive Pull-Up or Resistive Pull-Down

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

- Open Drain Drives High and Open Drain Drives Low

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

- Strong Drive

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

- Resistive Pull-Up and Resistive Pull-Down

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

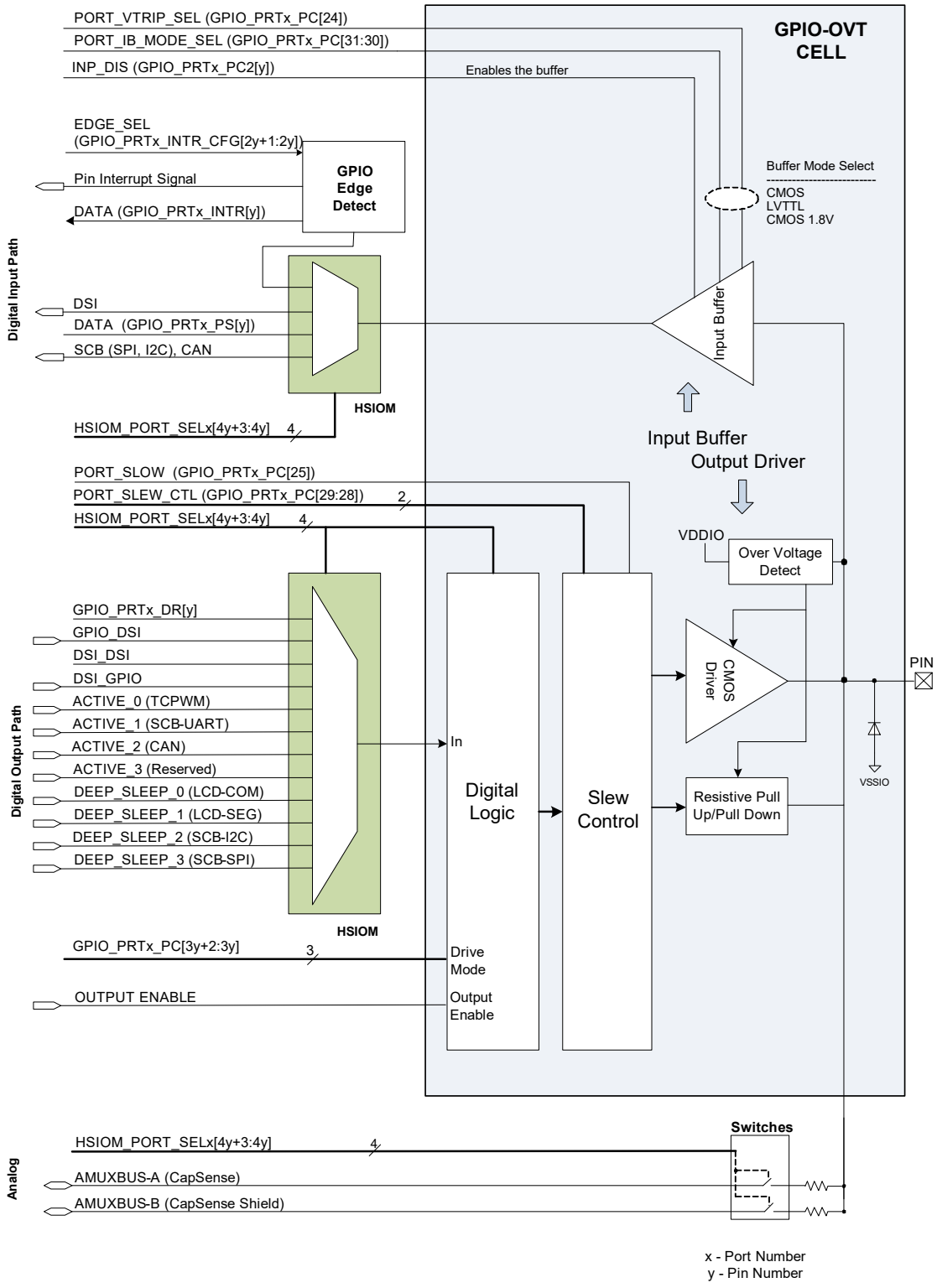
7.3.2.2 Slew Rate Control

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

7.4 GPIO-OVT Pin

Port 6 and Port 8 in the device have overvoltage tolerant (OVT) pins. Figure 7-4 shows the GPIO-OVT Pin Architecture.

Figure 7-4. GPIO-OVT Architecture in PSoC 4200L



It is similar to regular GPIOs with the following additional features:

- Overvoltage tolerant - The GPIO-OVT cell has the hardware (represented by “Overvoltage Detect” block in Figure 7-4) to compare the V_{DDIO} and the pin voltage. If the pin voltage exceeds the V_{DDIO} voltage, the output driver is disabled and the pin is tristated. This results in negligible current sink at the pin. Also, there are ESD clamp diodes only between the pin and V_{SSIO} to clamp negative voltage spikes.

  Note that in overvoltage conditions at the pin, the input buffer data will not be valid if the external source’s specification of VOH and VOL do not match the trip points of the input buffer.
- Provides better pull-down drive strength when compared to a regular GPIO.
- 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

The minimum fall time specification is achieved with the additional slew control feature configured using PORT_SLEW_CTRL bits of Port Configuration register (GPIO_PRTx_PC[29:28]). Table 7-4 shows the PORT_SLEW_CTRL settings.

Table 7-4. Slew Rate Control

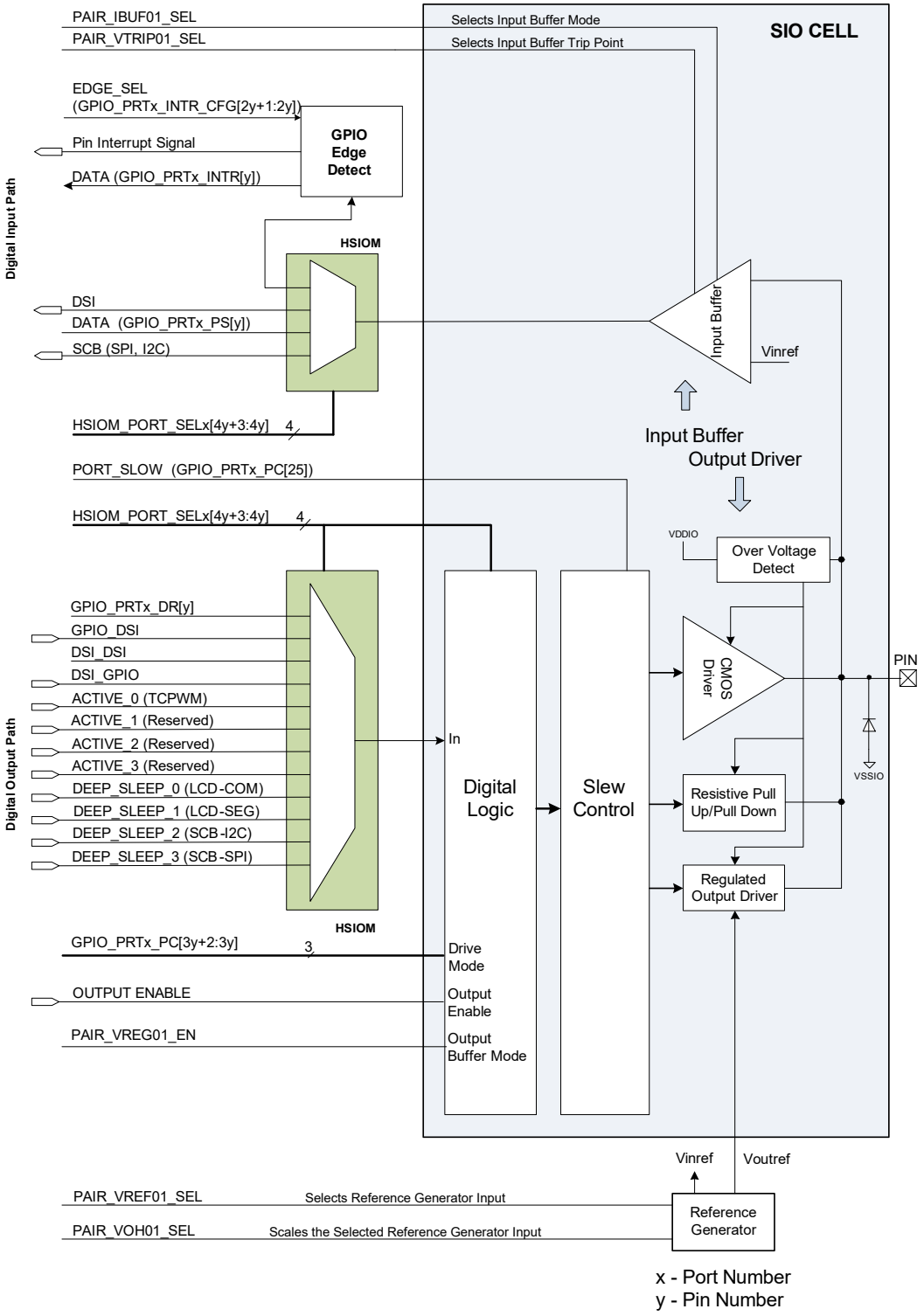
PORT_SLEW_CTRL (GPIO_PRTx_PC[29:28])	Usage
00b	Reserved
01b	I^2C Fast Mode Plus (FM+) for external I^2C bus voltage > 2.8 V
10b	Reserved
11b	I^2C Fast Mode Plus (FM+) for external I^2C bus voltage ≤ 2.8 V

Note that to use the PORT_SLEW_CTRL, it is required to configure the drive mode to open drain drive low mode. If other drive modes are selected, PORT_SLEW_CTRL has not effect.

7.5 Special I/Os (SIOs)

PSoC 4200L provides two SIO pins - P12[0] and P12[1]. Figure 7-5 shows the SIO architecture.

Figure 7-5. SIO Architecture



An SIO is similar to the regular GPIO, but it consists of the following additional features:

- Reference generator
- Differential input buffer
- Regulated output driver
- Overvoltage tolerance

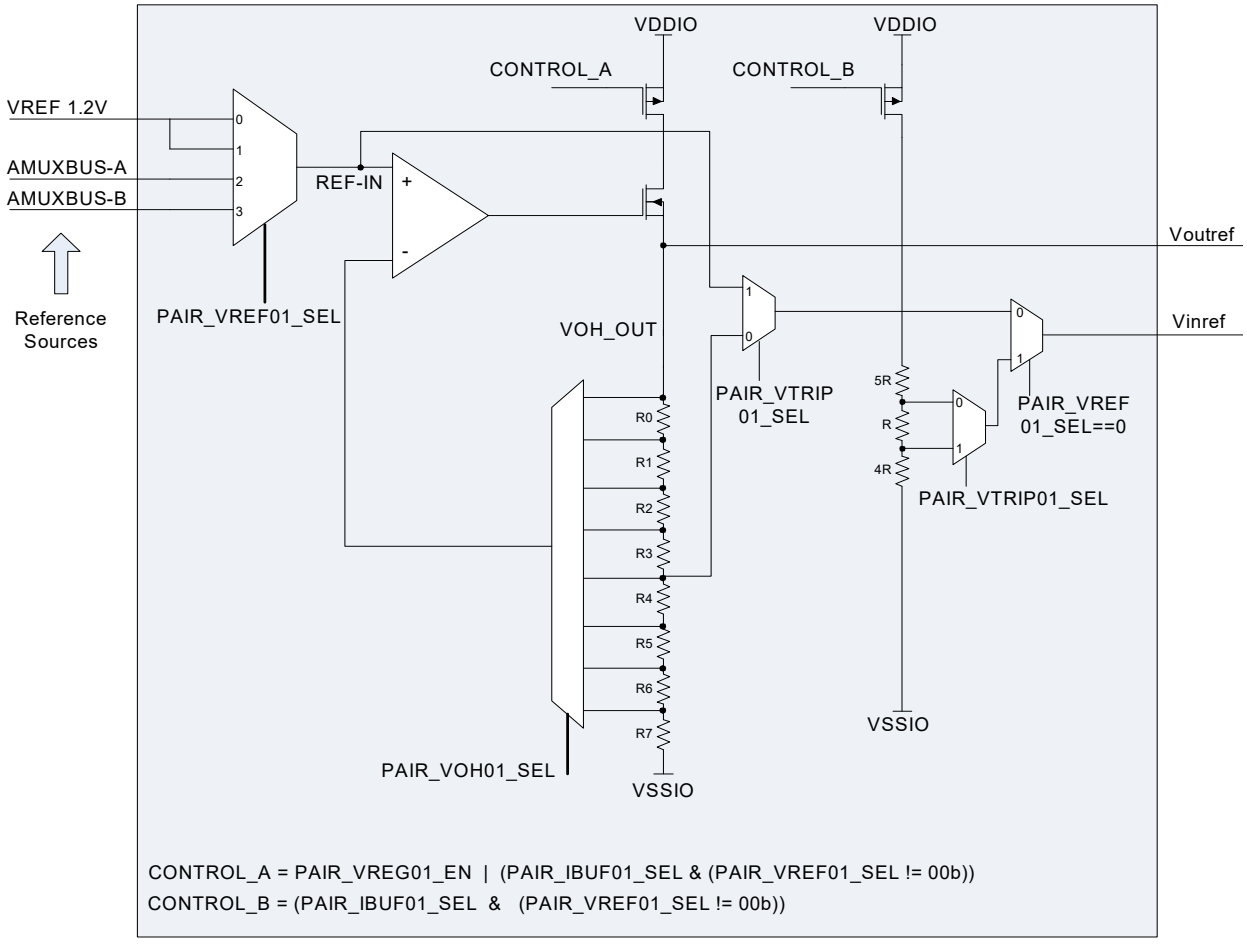
However, SIOs do not support 1.8-V CMOS mode and no connections are possible with analog peripherals.

SIOs are organized in pairs of pins. In PSoC 4200L, SIO pins P12[0] and P12[1] form the pair. Each SIO pair consists of a cell for each pin and a reference generator shared by both the pins.

7.5.1 Reference Generator

The reference generator block is an opamp connected in negative feedback loop configuration to generate the output reference voltage Voutref for the regulated output and the input buffer reference Vinref for trip setting as shown in Figure 7-6.

Figure 7-6. Reference Generator



The input to the opamp is selected by PAIR_VREF01_SEL bits of the SIO register as Table 7-5 shows.

Table 7-5. Reference Generator Input Selection

PAIR_VREF01_SEL	Selected Source REF-IN
00	VREF (1.2V)
01	VREF (1.2V)
10	AMUXBUS-A
11	AMUXBUS-B

The output reference voltage Voutref/VOH_OUT in the circuit is controlled by setting the resistor tap using PAIR_VOH01_SEL bits of the SIO register. Table 7-6 provides the gain at different values of PAIR_VOH01_SEL bits to the input selected by PAIR_VREF01_SEL bits.

Table 7-6. Gain Selection in Reference Generator

PAIR_VOH01_SEL	Rtap/Rtotal	Gain (Rtotal/Rtap)
000	1	1
001	0.8	1.25
010	0.67	1.49
011	0.6	1.66
100	0.48	2.08
101	0.4	2.5
110	0.36	2.77
111	0.25	4

The resistor values in the feedback divider network are skewed to generate standard Voutref voltages; that is, 1.2 V, 1.5 V, 1.8 V, 2 V, 2.5 V, 3.3 V, and 4.8 V when VREF (1.2 V) is selected as the input to the reference generator. Also, the upper value of Voutref is limited to (V_{DDIO} – 400 mV).

The same circuit is used to generate the reference Vinref for the input buffer. The PAIR_VREF01_SEL and PAIR_VTRIP01_SEL bits of the SIO register set the input buffer reference Vinref. VOH_OUT in Table 7-7 depends on the source selection through the PAIR_VREF01_SEL bit shown in Table 7-5 and the gain setting through PAIR_VOH01_SEL bit shown in Table 7-6. For example, if PAIR_VREF01_SEL is configured to 10b and PAIR_VOH01_SEL is set to 001b, then VOH_OUT is equal to AMUXBUS-A voltage x 1.25. Finally, if PAIR_VTRIP_SEL is set to 0, Vinref is equal to AMUXBUS-A voltage x 1.25 x 0.5. REF-IN is the reference source selected using the PAIR_VREF01_SEL bit as shown in Table 7-5.

Table 7-7. Vinref Generation

PAIR_VREF01_SEL	PAIR_VTRIP_SEL	Vinref
00	0	50% of V_{DDIO}
00	1	40% of V_{DDIO}
01/10/11	0	0.5*VOH_OUT
01/10/11	1	REF-IN

7.5.2 Differential Input Buffer

Regular GPIOs has a single-ended buffer with trip points selected using PORT_VTRIP_SEL, as described in section 7.3.1 Digital Input Buffer. SIOs have differential input buffer with the reference voltage Vinref from the reference generator described in section 7.5.1 Reference Generator. Vinref sets the trip point of the digital input buffer. See the device datasheet for the specifications on input voltage high threshold (VIH) and low threshold (VIL).

The buffer can be switched to a single-ended mode by writing to the PAIR_IBUF01_SEL bit of the SIO register as Table 7-8 shows.

Table 7-8. Input Buffer Mode Configuration

PAIR_INBUF01_SEL	Input Buffer Mode
0	Single Ended Mode
1	Differential Mode

Note that the differential Input buffer mode cannot be used in deep-sleep, hibernate, and stop modes.

7.5.3 Regulated Output Driver

Output driver of the SIO cell consists of a CMOS output driver, similar to a standard GPIO, supporting eight different drive modes and a regulated output driver supporting configurable output voltages at the pin. An SIO can behave as a regular GPIO by writing into PAIR_VREG01_EN bit of the SIO register as Table 7-9 shows.

Table 7-9. SIO Output Driver Mode Selection

PAIR_VREG01_EN	Output Driver Mode
0	CMOS output
1	Regulated Output

To set the output driver to regulated mode, the drive mode bits of the Port Configuration register (GPIO_PRTx_PC) should be set to strong drive mode. If other drive modes are selected, the output driver works in the standard CMOS mode and the regulated output driver is disabled.

In the regulated output mode, the output voltage is set by Voutref supplied by the reference generator, described in section 7.5.1 Reference Generator. Table 7-10 summarizes the SIO output configuration.

Table 7-10. SIO Output Driver Configuration

PAIR_VREG01_EN	PAIR_VREF01_SEL	PAIR_VOH01_SEL	SIO configuration
0	X	X	Standard CMOS mode
1	00/01	[0-7]	Regulated Output Mode with Voutref based on VREF of 1.2 V
1	10	[0-7]	Regulated Output Mode with Voutref based on AMUXBUS-A voltage
1	11	[0-7]	Regulated Output Mode with Voutref based on AMUXBUS-B voltage

The slew rate can also be configured for the SIO output, similar to a regular GPIO, using the PORT_SLOW bit of the Port Configuration register (GPIO_PRTx_PC[25]).

Note that the SIO cannot be used in regulated output mode in deep-sleep, hibernate, and stop modes.

7.5.4 Overvoltage Tolerance

The overvoltage tolerant feature in a SIO is the same as in a GPIO-OVT pin. The only difference is that there is no slew rate control feature to support fall time spec in fast mode plus of the I^2C.

7.6 High-Speed I/O Matrix

The high-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the peripherals inside the device. As the GPIOs are shared for multiple functions, HSIOM multiplexes the pin and connects to a particular peripheral selected by the user. The HSIOM_PORT_SELx register is provided to select the peripheral. It is a 32-bit wide register available for each port, with each pin occupying four bits. This register provides up to 16 different options for a pin as listed in Table 7-11.

Table 7-11. PSoC 4200L 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).
	ACTIVE_0	8	Pin-specific Active source #0 (TCPWM, EXT_CLK)
	ACTIVE_1	9	Pin-specific Active source #1 (SCB-UART)
	ACTIVE_2	10	Pin-specific Active source #2 (CAN)
	ACTIVE_3	11	Reserved
	DEEP_SLEEP_0	12	Pin-specific Deep-Sleep source #0 (LCD - COM)
	DEEP_SLEEP_1	13	Pin-specific Deep-Sleep source #1 (LCD - SEG)
	DEEP_SLEEP_2	14	Pin-specific Deep-Sleep source #2 (SCB-I^2C, SWD, LPCOMP, WAKEUP)
	DEEP_SLEEP_3	15	Pin-specific Deep-Sleep source #3 (SCB-SPI)

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

7.7 I/O State on Power Up

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

7.8 Behavior in Low-Power Modes

Table 7-12 shows the status of GPIOs in low-power modes.

Table 7-12. PSoC 4200L I/Os in Low-Power Modes

Low-Power Mode	Status
Sleep	■ Standard GPIO, GPIO-OVT, and SIO pins are active and can be driven by peripherals such as CapSense, TCPWM, and SCB, which can work in sleep mode. ■ Inputs buffers are active; thus an interrupt on any I/O can be used to wake up the CPU.
Deep-Sleep	■ GPIO, GPIO-OVT, and SIO pins, connected to the deep-sleep domain peripherals, are functional. Other pins, with its output enabled, are in the frozen state. ■ Pin interrupts are functional on all I/Os.
Hibernate	■ Pin output states are latched and remain in the frozen state. ■ Pin interrupts are functional on all I/Os. Note that the input buffer of the GPIO, GPIO-OVT, and SIO pins should not be configured to 1.8 V CMOS mode. This mode is non-functional in hibernate mode.
Stop	■ GPIO and GPIO-OVT output states are latched and remain in the frozen state. ■ Interrupt on only port P2[2] wakes up the device; therefore, the input buffer is not configured to 1.8 V CMOS mode. Input buffer on other pins are inactive.

7.9 Input and Output Synchronization

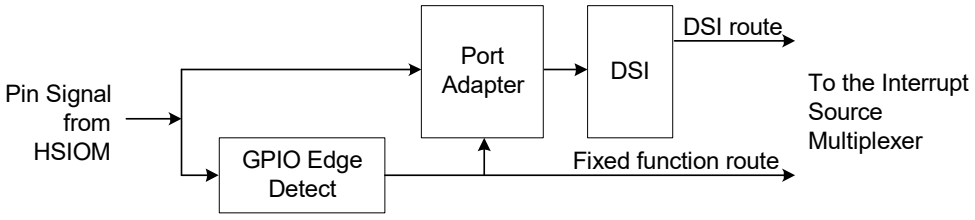
For digital input and output signals, the I/O provides synchronization with an internal clock or a digital signal as clock. By default, HFCLK is used for synchronization but any other clock can also be used.

This feature is implemented using the UDB port adapter. See the Universal Digital Blocks (UDB) chapter on page 148 for details on the port adapter.

7.10 Interrupt

In the PSoC 4 device, all the port pins have the capability to generate interrupts. There are three routing possibilities for the pin signals to generate the interrupt, as shown in Figure 7-7.

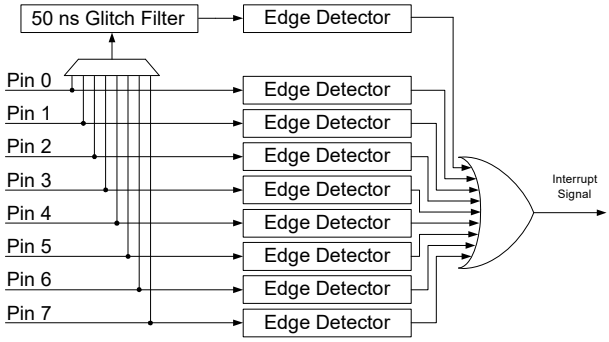
Figure 7-7. Interrupt Signal Routing



- Through the “GPIO Edge Detect” block and direct connection to the interrupt source multiplexer
- Through the “GPIO Edge Detect” block and to the interrupt source multiplexer via DSI
- Pin signal to the interrupt source multiplexer via DSI bypassing the “GPIO Edge Detect” block

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

Figure 7-8. GPIO Edge Detect Block Architecture



An edge detector is present at each pin. It is capable of detecting rising edge, falling edge, and both edges without reconfiguration. The edge detector is configured by writing into the EDGE_SEL bits of the Port Interrupt Configuration register, GPIO_PRTx_INTR_CFG, as shown in Table 7-13.

Table 7-13. Edge Detector Configuration

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

Besides the pins, edge detector is also present at the glitch filter output. This filter can be used on one of the pins of a port. The pin is selected by writing to the FLT_SEL field of the GPIO_PRTx_INTR_CFG register as shown in Table 7-14.

Table 7-14. Glitch filter Input Selection

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

The edge detector outputs of a port are ORed together and then routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port. On a pin interrupt, it is required to know which pin caused an interrupt. This is done by reading the Port Interrupt Status register, GPIO_PRTx_INTR. This register not only includes the information on which pin triggered the interrupt, it also includes the pin status; it allows the CPU to read both information in a single read operation. This register has one more important use – to clear the interrupt. Writing '1' to the corresponding status bit clears the pin interrupt. It is important to clear the interrupt status bit; otherwise, the interrupt will occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. Also, note that when the Port Interrupt Control Status register is read when an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code. Table 7-15 shows the Port Interrupt Status register bit fields.

Table 7-15. Port Interrupt Status Register

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

The edge detector block output is routed to the Interrupt Source Multiplexer shown in Figure 6-3 on page 46, which gives an option of Level and Rising Edge detect. If the Level option is selected, an interrupt is triggered repeatedly as long as the Port Interrupt Status register bit is set. If the Rising Edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the Edge Detect block is used.

There is a dedicated interrupt vector for each port when the interrupt signal is routed through the fixed-function route. However, when the signal is routed though the DSI, interrupt vector is flexible and can occupy any of the 32 interrupt lines of the NVIC. See the Interrupts chapter on page 45 for details.

When the signal is routed to the DSI, bypassing the Edge Detect block, the edge detection is configurable in the Interrupt Source Multiplexer block. It is important to note that if the multiplexer is configured as Level, the interrupt is triggered repeatedly as long as the pin signal is high. It is recommended to use the Rising Edge detect option when this route is selected.

7.11 Peripheral Connections

7.11.1 Firmware Controlled GPIO

See Table 7-11 to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

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

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

7.11.2 Analog I/O

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

To configure a GPIO as a dedicated analog I/O, it should be configured in high-impedance analog mode (see Table 7-3) and the respective connection should be enabled in the specific analog resource. This can be done via registers associated with the respective analog resources.

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

7.11.2.1 AMUXBUS Connection and DSI

In ports that support DSI connectivity, connecting a pin to AMUXBUS A/B requires the user to configure a couple of DSI signals in addition to configuring the HSIOM_PORT_SELx register. The DSI output and the DSI output enable signal of the pin should be set high to enable the AMUXBUS connection. This option allows the use of DSI signals to control the AMUXBUS connection on a pin, thus enabling the user to implement hardware AMUXBUS switching through DSI. To properly configure a pin as AMUXBUS input, follow these steps:

1. Route the DSI output and DSI output enable signals out of the pin in the design and connect them to logic '1' for static AMUXBUS connection. The signals can be connected to a selection signal from DSI, if dynamic AMUXBUS connection is required in the design. This can be done within the device.
2. Configure the GPIO_PRTx_PC register to set the pin in Hi-Z analog mode – this enables the analog connectivity on the pin by disabling the input buffer.
3. Configure the HSIOM_PRT_SELx register to connect the pin to AMUXBUS A or B.

7.11.3 LCD Drive

All GPIOs have the capability of driving an LCD common or segment. HSIOM_PORT_SELx registers are used to select the pins for LCD drive. See the LCD Direct Drive chapter on page 301 for details.

7.11.4 CapSense

The pins that support CSD can be configured as CapSense widgets such as buttons, slider elements, touchpad elements, or proximity sensors. CapSense also requires external tank capacitors and shield lines. Table 7-16 shows the GPIO and HSIOM settings required for CapSense. See the CapSense chapter on page 313 for more information.

Table 7-16. CapSense Settings

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

7.11.5 Serial Communication Block (SCB)

SCB, which can be configured as UART, I^2C, and SPI, has dedicated connections to the pin. See the device datasheet or details on these dedicated pins of PSoC 4. When the UART and SPI mode is used, the SCB controls the digital output buffer drive mode for the input pin to keep the pin in the high-impedance state. That is, the SCB block disables the output buffer at the UART Rx pin and MISO pin when configured as SPI master, and MOSI and select line when configured as SPI slave. This functionality overrides the drive mode settings, which is done using the GPIO_PRTx_PC register.

7.12 Port Restrictions

Port 7, Port 8, Port 9, and Port 10 pins do not have the port adapter resulting in the following restrictions:

- Cannot be routed through the DSI; thus UDB-based digital signals cannot be routed to the pins of these ports
- No input/output synchronization

However, these ports can be used in the following ways:

- As GPIO controlled in firmware
- Direct connection to TCPWM, SCB, or CAN
- LCD and CapSense pins
- Interrupts generation

7.13 Registers

Table 7-17. I/O Registers

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

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

8. Clocking System

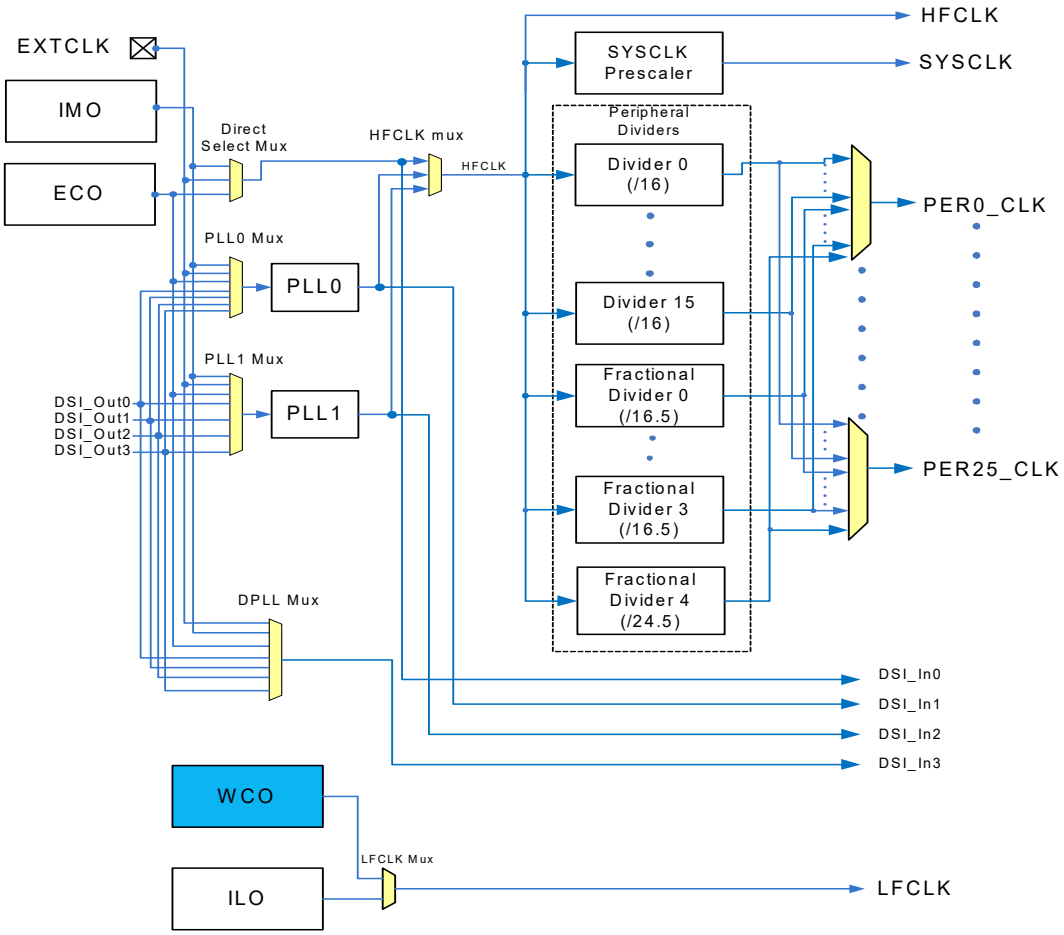
The PSoC® 4 clock system includes these clock resources:

- Two internal clock sources:
  - 3–48 MHz internal main oscillator (IMO) ±2 percent accuracy across all frequencies with trim
  - 32-kHz internal low-speed oscillator (ILO) with ±60 percent accuracy with trim (can be calibrated using the IMO)
- Three external clock sources:
  - External clock (EXTCLK) generated using a signal from an I/O pin
  - External 4- to 33-MHz crystal oscillator (ECO)
  - External 32-kHz watch crystal oscillator (WCO)
- High-frequency clock (HFCLK) of up to 49.152 MHz, selected from IMO, ECO,external clock, or one of two PLLs
- Low-frequency clock (LFCLK) sourced by ILO or WCO
- Dedicated prescaler for system clock (SYSCLK) of up to 49.152 MHz sourced by HFCLK
- Two PLLs that can generate up to 49.152 MHz output from the IMO, ECO, or external clock
- Sixteen 16-bit peripheral clock dividers
- Five fractional dividers (four 16.5 fractional dividers and one 24.5 fractional divider) for accurate clock generation
- Twenty-five digital and analog peripheral clocks

8.1 Block Diagram

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

Figure 8-1. Clocking System Block Diagram



The five clock sources in the device are IMO, ECO, EXTCLK, WCO, and ILO, as shown in Figure 8-1. There is a are two PLLs that can be configured to take clock sources from the IMO, ECO, EXTCLK, or one of the DSI_out clocks. The DSI_out clocks can be used to route clocks from the programmable digital logic through the DSI. The PLL output frequency can be as high as 108 MHz. Although the PLL output frequency can be as high as 108 MHz, not all blocks accept frequencies higher than 48 MHz. See 8.2.5 PLL0 and PLL1 on page 77 for details. The HFCLK mux selects the HFCLK source from the EXTCLK, ECO, PLL0, PLL1, or the IMO. The HFCLK frequency can be a maximum of 49.152 MHz. The SYSCLK prescaler generates the SYSCLK and the peripheral dividers generate the individual peripheral clocks. There are sixteen integral dividers and four fractional dividers. The clock tree has 25 clock outputs that route to specific peripherals in the device. Each clock output can select from any of the sixteen integral divider outputs or five fractional divider outputs. See "Clock Distribution" on page 78 for details. The clocking system has four clock outputs to the DSI. These clock outputs, called the DSI_In clocks, are fed into the DSI, which can be used to clock digital circuits. The LFCLK mux selects the LFCLK from the WCO or the ILO. The LFCLK is used as a source for one of the timers in the watchdog timer. See the Watchdog Timer chapter on page 97 for details.

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. Frequency selection follows an algorithm to ensure no intermediate state is programmed to a value higher than 48 MHz. Each PSoC device has IMO trim settings determined during manufacturing to meet datasheet

specifications; the trim is stored in manufacturing configuration data in SFLASH. There are TRIM values corresponding to the frequency selected by the user. The TRIM values from SFLASH are loaded in the corresponding trim register – 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.

To configure the IMO frequency, follow this algorithm:

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

Table 8-1. IMO Frequency Configuration

CLK_IMO_TRIM2						Frequency in MHz
Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
0	0	0	0	1	1	3
0	0	0	1	0	0	4
0	0	0	1	0	1	5
0	0	0	1	1	0	6
0	0	0	1	1	1	7
0	0	1	0	0	0	8
0	0	1	0	0	1	9
0	0	1	0	1	0	10
0	0	1	0	1	1	11
0	0	1	1	0	0	12
0	0	1	1	1	0	13
0	0	1	1	1	1	14
0	1	0	0	0	0	15
0	1	0	0	0	1	16
0	1	0	0	1	0	17
0	1	0	0	1	1	18
0	1	0	1	0	0	19
0	1	0	1	0	1	20
0	1	0	1	1	0	21
0	1	0	1	1	1	22

Table 8-1. IMO Frequency Configuration

CLK_IMO_TRIM2						Frequency in MHz
Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
0	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 8-2. The limits of the distribution are defined with register CLK_IMO_SPREAD bits SS_RANGE, which are shown in Table 8-3. All spread options are downspread, meaning that instantaneous clock frequency values are always at or below the configured frequency.

Table 8-2. 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 8-3. 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 can be calibrated periodically using a higher accuracy, high-frequency clock to improve accuracy. The ILO is available in all power modes except Hibernate and Stop modes. The ILO is used as the system low-frequency clock LFCLK in the device. The ILO is a relatively inaccurate (±60 percent overvoltage and temperature) oscillator, which is used to generate low-frequency clocks. If calibrated against the IMO when in operation, the ILO is accurate to ±10 percent for stable temperature and voltage. The ILO is enabled and disabled with register CLK_ILO_CONFIG bit ENABLE.

8.2.3 External Clock (EXTCLK)

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

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the I/O System chapter on page 55 for more details.

8.2.4 External Crystal Oscillator (ECO)

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

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

8.2.4.1 ECO Trimming

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

Maximum peak to peak value:

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

Transconductance:,

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

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

The ATRIM and WDTRIM settings control the trim for amplitude of the oscillator output. Amplitude trim (ATRIM) sets the crystal drive level when AGC is enabled (ECO_CONFIG.AGC_EN=1). AGC must be enabled for V_{PP} < 2 V and disabled for all other cases. Based on the V_{PP} value, the ATRIM and WDTRIM values are set as shown in Table 8-4.

Table 8-4. ATRIM and WDTRIM Setting

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

The GTRIM sets up the trim for amplifier gain and is set based on the g_m calculated as shown in Table 8-5.

Table 8-5. GTRIM Setting

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

The FTRIM and RTRIM sets up the trim for the filter characteristics and is set based on the g_m calculated as shown in Table 8-6.

Table 8-6. FTRIM and RTRIM Setting

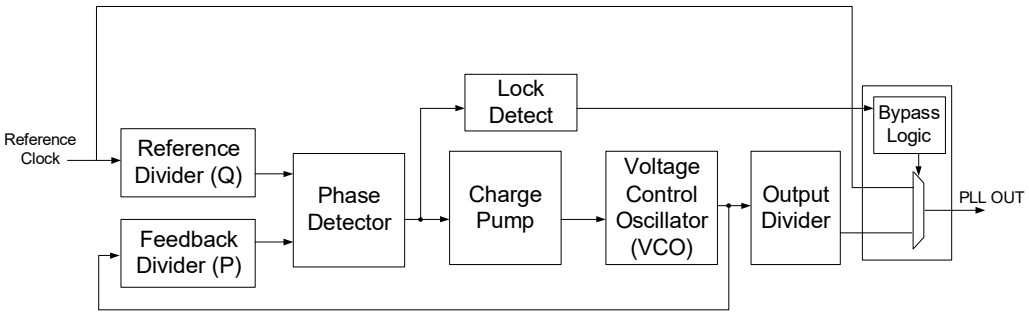
Nominal Frequency f (MHz)	RTRIM	FTRIM
f > 30 MHz	0x00	0x00
30 MHz ≥ f > 24 MHz	0x01	0x01
24 MHz ≥ f > 17 MHz	0x02	0x02
17 MHz ≥ f	0x03	0x03

8.2.5 PLL0 and PLL1

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

P and Q for PLL0 and PLL1 are configured by PLL0_CONFIG and PLL1_CONFIG, respectively. In this register, PLLx_CONFIG[7:0] is the P value and PLLx_CONFIG[13:8] is the Q value. By varying this integer value for the feedback counter (P has eight bits of resolution) and the reference counter (Q has six bits of resolution), the PLL can synthesize a large number of output frequencies from an input reference clock frequency (clk_ref).

Figure 8-2. PLL Block Diagram



The PLL has an integrated output divider, which allows for running the VCO at a higher frequency. This allows for improved jitter and duty cycle compared to an undivided output at the same frequency. The output divider has four post divide options,

configured by PLLx_CONFIG[15:14] as shown in Table 8-7. There may be a glitch in the PLL output when the divider values are changed. It is the users responsibility to change PLL destination routes before changing the divider values.

Table 8-7. Output Divider Options

PLL_CONFIG[15:14]	PLL Frequency
00	FVCO (Divider bypassed}
01	FVCO/2
10	FVCO/4
11	FVCO/8

The charge pump in the PLL has three configuration bits. These set the charge pump currents in the PLL. This configuration is preset based on frequency and is not suggested for user operation.

A lock detect block is used to indicate when the PLL achieves the "locked" condition. This lock condition can be observed in the PLL_STATUS register.

The PLL also has a bypass logic, which can be used to bypass the PLL block and have the reference clock stream out to the PLL output. This feature is configured in PLLx_CONFIG[21:20]. The bypass logic has three modes of operation:

- AUTO/AUTO1: When there is a lock detected, the PLL output is routed; otherwise, the PLL is bypassed.
- PLL_REF: The PLL is always bypassed and ignores the lock detect.
- PLL_OUT: Normal mode of operation. Uses the PLL to block output.

8.2.6 Watch Crystal Oscillator (WCO)

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

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

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

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

8.3 Clock Distribution

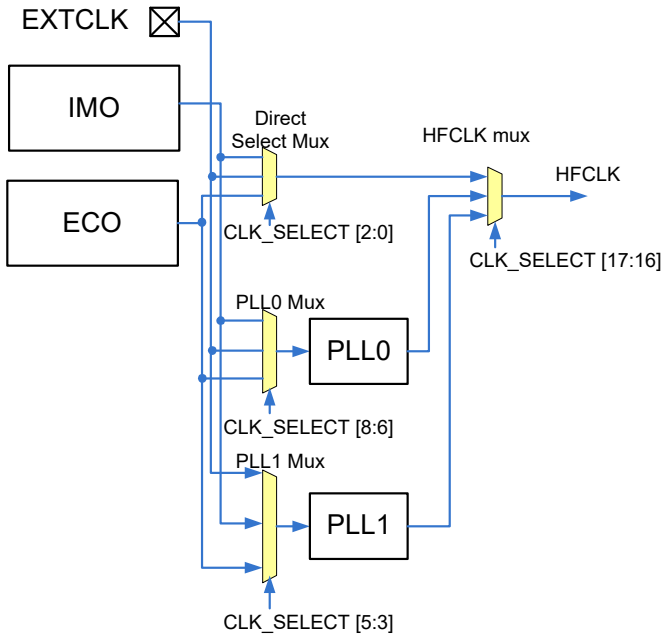
PSoC 4 clocks are developed and distributed throughout the device, as shown in Figure 8-1. The distribution configuration options are as follows:

- HFCLK input selection
- PLL0 input selection
- PLL1 input selection
- LFCLK input selection
- SYSCLK prescaler configuration
- Peripheral divider configuration

8.3.1 HFCLK and PLL Input Selection

Figure 8-3 shows the selection options for HFCLK and PLL.

Figure 8-3. HFCLK and PLL Selection Options



The PLL0 and PLL1 have multiplexers that select between EXTCLK, ECO, IMO, and four DSI output signals. The input selection for PLL0 is in the CLK_SELECT[8:6] register. The input selection for PLL1 is in the CLK_SELECT[5:3] register. Although the topology in Figure 8-1 makes it possible for DSI_Out signals to generate the HFCLK (through the PLL #0/1), this will not be allowed as a real use case. The DSI_Out signals is intended to feed the PLL#0/1 blocks to create asynchronous clocks for use in UDB User Modules.

For HFCLK, it is a two-stage selection. The three sources ECO, EXTCLK, and IMO are first multiplexed using a direct select multiplexer whose selection is configured in CLK_SELECT[2:0]. The second multiplexer called the HFCLK multiplexer selects the direct select output and the PLLs. This selection is configured in the CLK_SELECT[17:16] register.

8.3.2 HFCLK Input Selection

HFCLK in PSoC 4 has five input options: IMO, EXTCLK, PLL0, Pll1, and ECO. The IMO, EXTCLK, and ECO are first selected in a stage. The HFCLK input is selected using the CLK_SELECT register's DIRECT_SEL bits, as described in Table 8-8.

Table 8-8. 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

The HFCLK is selected in another level, between the muxed output of the DIRECT_SEL, PLL0, and PLL1. This selection is made in the CLK_SELECT register's HFCLK_SEL bits as described in Table 8-9.

Table 8-9. CLK_SELECT Selection

Name	Description
CLK_SELECT [17:16]	HFCLK input clock selection 0: HFCLK source is selected based on the CLK_SELECT[2:0] setting 1: PLL1 as the source of the HFCLK 2: PLL0 as the source of the HFCLK

8.3.3 PLL Input Selection

The PLLs in PSoC 4 has seven input options: IMO, ECO, EXTCLK, and four DSI sources. The DSI_Out signals are not a valid selection when the PLL is sourcing the HFCLK. THE PLL source is selected using the CLK_SELECT register, as shown in Table 8-10.

Table 8-10. PLL Input Selection Bits

Name	Description
CLL_SELECT[5:3]	PLL1 input clock selection 0: IMO. Uses the IMO as the PLL1 source 1: EXTCLK. Uses the external clock as the PLL1 source 2: ECO. Uses the external crystal oscillator as the PLL1 source 4: DSI_out[0] as the PLL1 source (Not to be used when PLL sources HFCLK) 5: DSI_out[1] as the PLL1 source (Not to be used when PLL sources HFCLK) 6: DSI_out[2] as the PLL1 source (Not to be used when PLL sources HFCLK) 7: DSI_out[3] as the PLL1 source (Not to be used when PLL sources HFCLK)
CLL_SELECT[8:6]	PLL0 input clock selection 0: IMO. Uses the IMO as the PLL0 source 1: EXTCLK. Uses the external clock as the PLL0 source 2: ECO. Uses the external crystal oscillator as the PLL0 source 4: DSI_out[0] as the PLL0 source (Not to be used when PLL sources HFCLK) 5: DSI_out[1] as the PLL0 source (Not to be used when PLL sources HFCLK) 6: DSI_out[2] as the PLL0 source (Not to be used when PLL sources HFCLK) 7: DSI_out[3] as the PLL0 source (Not to be used when PLL sources HFCLK)

8.3.4 LFCLK Input Selection

The LFCLK in PSoC 4 has two input options: ILO and WCO. The LFCLK is selected using the WDT_CONFIG register's LFCLK_SEL bits, as shown in Table 8-11. The LFCLK selection can glitch if the selection is changed near an LFCLK edge. The LFCLK clocks the watchdog timer (WDT); therefore, ensure that the LFCLK source switching does not affect the WDT. To safely change LFCLK_SEL, wait for WDT_CTRLOW/WDT_CTRHIGH to change; then, change the setting immediately.

Table 8-11. 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

8.3.5 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 bypowers 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 8-12. The prescaler is initially configured to divide by 1.

Table 8-12. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

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

8.3.6 Peripheral Clock Divider Configuration

PSoC 4 has 21 clock dividers, which include sixteen 16-bit clock dividers, four 16.5-bit fractional clock dividers, and one 24.5-bit fractional clock divider. Fractional clock dividers allow the clock divisor to include a fraction of 0..31/32. The formula for the output frequency of a fractional divider is Fout = Fin / (INT16_DIV + (FRAC5_DIV/32)). For example, a 16.5-divider with an integer divide value of 2 (INT16_DIV=3, FRAC5_DIV=0), produces signals to generate a 16-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 3 (INT16_DIV=3, FRAC5_DIV=0), produces signals to generate a 12-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 2 (INT16_DIV=3) and a fractional divider of 16 (FRAC5_DIV=16) produces signals to generate a 13.7-MHz clock from a 48-MHz HFCLK. Not all 13.7-MHz clock periods are equal in size; half of them will be 3 HFCLK cycles and half of them will be 2 HFCLK cycles.

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

The divide value for each of the 16 integer clock dividers are configured with the PERI_DIV_16_CTLx registers and the four 16.5-bit fractional clock dividers are configured with the PERI_DIV_16_5_CTLx registers. Table 8-13 and Table 8-14 describe the configurations for these registers. The 24.5-bit fractional dividers are configured using the PERI_DIV_24_5_CTL0 register. Table 8-15 describes the configuration for these registers.

Table 8-13. 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, 65536].

Table 8-14. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_16_5_CTLx

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

Table 8-15. Fractional Peripheral Clock Divider Configuration Register PERI_DIV_24_5_CTLx

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

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

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

The SEL_TYPE field specifies the type of divider being configured. This field is '1' for the 16-bit integer divider, '2' for the 16.5-bit fractional divider, and ‘3’ for the 24.5-bit fractional divider.

The SEL_DIV field specifies the number of the specific divider being configured... When SEL_TYPE = 63 and SEL_TYPE = 3, no divider is specified.

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

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

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

Table 8-16 and shows the mapping of the mux output to the corresponding peripheral blocks (shown in Figure 8-1). Any of the peripheral clock dividers can be mapped to a specific peripheral by using their respective PERI_PCLK_CTLx register, as described in Table 8-17.

Table 8-16. Peripheral Clock Multiplexer Output Mapping

Peripheral Clock #	Peripheral
0	IMO (Spread Spectrum)
1	CLOCK_PUMP
2	SCB0
3	SCB1
4	SCB2
5	SCB3
6	CSD0_ CLK0
7	CSD0_ CLK1
8	CSD1_CLK0
9	CSD 1_CLK1
10	SAR
11	TCPWM0
12	TCPWM1

Table 8-16. Peripheral Clock Multiplexer Output Mapping

Peripheral Clock #	Peripheral
13	TCPWM2
14	TCPWM3
15	TCPWM4
16	TCPWM5
17	TCPWM6
18	TCPWM7
19	UDB0
20	UDB1
21	UDB2
22	UDB3
23	LCD
24	PRGIO

Table 8-17. Programmable Clock Control Register - PERI_PCLK_CTLx

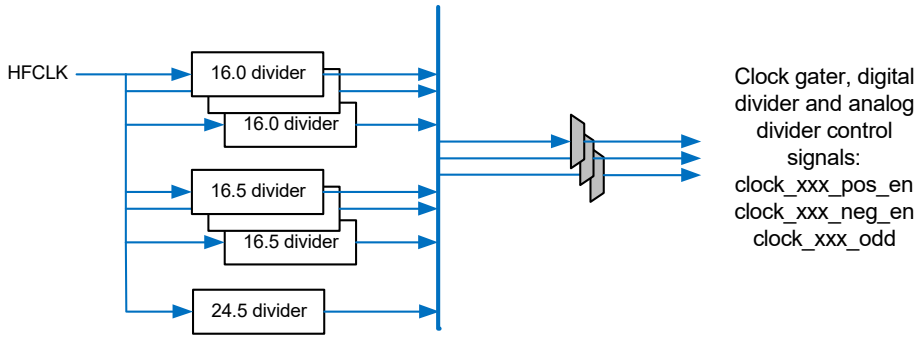
Bits	Name	Description
5:0	SEL_DIV	Specifies one of the dividers of the divider type specified by SEL_TYPE. If SEL_DIV is "4" and SEL_TYPE is "1", then the fifth (zero being first) 16-bit clock divider will be routed to the mux output for peripheral clock_x. Similarly, if SEL_DIV is "0" and SEL_TYPE is "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:

8.3.7 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 8-4 gives an overview of the peripheral clock divider.

Figure 8-4. Peripheral Clock Divider



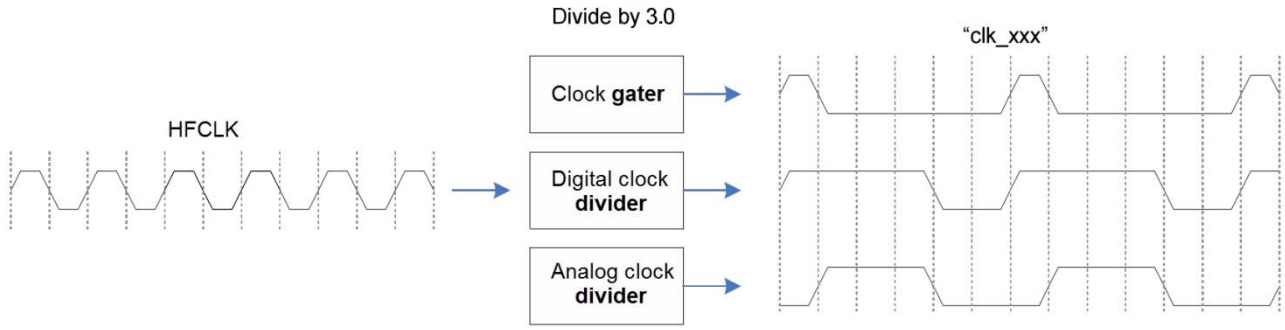
The peripheral clock divider consists of two layers:

- 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.

8.3.7.1 Clock Generation

Figure 8-5 gives an overview of the clock generation via the clock gater, digital clock divider, and analog clock divider.

Figure 8-5. 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

The high-frequency clocks including the IMO, EXTCLK, ECO, HFCLK, SYSCLK, PLLs, 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 8-18. 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.
CLK_ILO_CONFIG	ILO Configuration Register - This register controls the ILO configuration.
CLK_IMO_CONFIG	IMO Configuration Register - This register controls the IMO configuration.
CLK_SELECT	Clock Select - This register controls clock tree configuration, selecting different sources for the system clocks.
WCO_CONFIG	WCO Enable. This register enables or disables the external watch crystal oscillator.
CLK_IMO_SPREAD	Register controls IMO spread spectrum configuration.
ECO_CONFIG	Configuration register for the ECO
PLL0_CONFIG	Configuration register for PLL0.
PLL1_CONFIG	Configuration register for PLL1.

9. Power Supply and Monitoring

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

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

Note that the PSoC 4 devices support multiple supply rails depending on availability in the device package – V_{DDA}, V_{DDD}, and V_{DDIO}. These rails are independent and can be separately connected to desired and independent supplies. Even in direct supply mode, only the V_{DDD} needs to be limited to the 1.71 V to 1.89 V range and other rails such as V_{DDA}/V_{DDIO} can have the full 1.71 V to 5.50 V range. Refer to the device datasheet for details on various rails and their usage.

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

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

9.1 Block Diagram

Figure 9-1. Power System Block Diagram

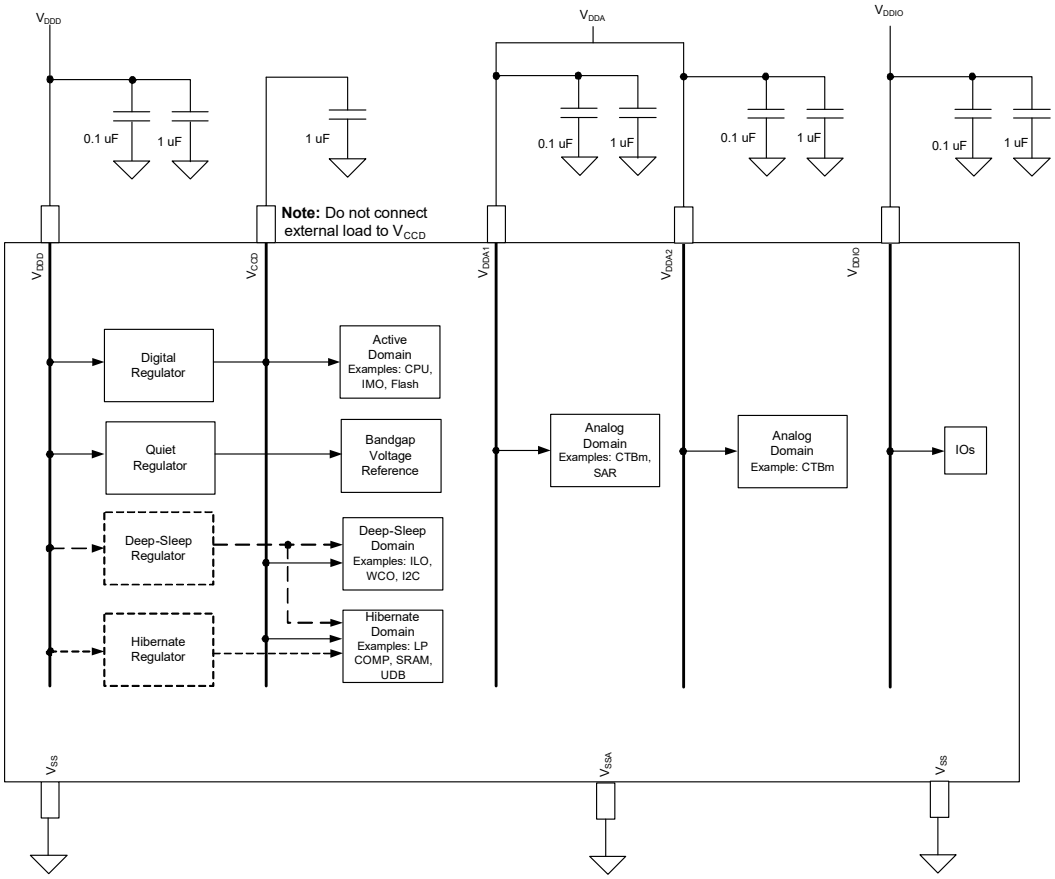


Figure 9-1 shows the power system diagrams and all the power supply pins as implemented for the PSoC 4. The system has one regulator in Active mode for the digital circuitry. There is no analog regulator; the analog circuits run directly from the V_{DDA} input. There are separate regulators for the Deep-Sleep and Hibernate (lowered power supply and retention) modes. There is a separate low-noise quiet regulator for the bandgap voltage references. The supply voltage range is 1.71 to 5.5 V with all functions and circuits operating over that range. The PSoC 4 allows two distinct modes of power supply operation: unregulated external supply and regulated external supply modes. See the "Power" Section in the device datasheet for details on power supply connections.

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

9.2.1 Regulator Summary

The Active digital regulator and Quiet regulator are enabled during the Active or Sleep power modes. They are turned off in the Deep-Sleep 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. 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

9.2.1.1 Active Digital Regulator

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

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

9.2.1.2 Quiet Regulator

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

9.2.1.3 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 Active digital regulator (V_{CCD}). This regulator also has a separate replica output that provides a stable voltage for the low-speed clock resources. This output is not connected to V_{CCD} in Active and Sleep modes.

9.2.1.4 Hibernate Regulator

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

9.3 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 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 9-2. Power Supply and Monitoring Register List

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

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

PSoC 4 operational modes are:

- Boot
- User
- Privileged
- Debug

10.1 Boot

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

10.2 User

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

10.3 Privileged

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

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see “Performing a System Call” on page 326. Exit from this mode returns the device to user mode.

10.4 Debug

Debug mode is an operational mode that allows observation of the PSoC 4 operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as PSoC Creator and Arm MDK to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the Program and Debug Interface chapter on page 318.

For more details on protection modes, see the Device Security chapter on page 104.

11. Power Modes

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

- Active
- Sleep
- Deep-Sleep
- 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 PSoC 4. 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 peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the PSoC 4

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

Figure 11-1. Power Mode Transitions State Diagram

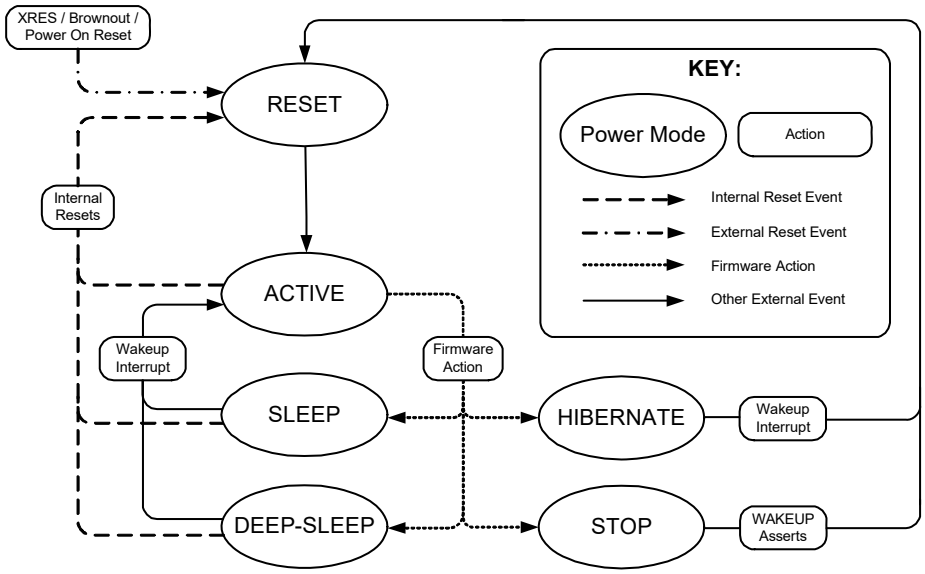


Table 11-1 illustrates the power modes offered by PSoC 4.

Table 11-1. PSoC 4 Power Modes

Power Mode	Description	Entry Condition	Wakeup Sources	Active Clocks	Wakeup Action	Available Regulators
Active	Primary mode of operation; all peripherals are available (programmable).	Wakeup from other power modes, internal and external resets, brownout, power on reset	Not applicable	All (programmable)	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	All (programmable)	Interrupt	All regulators are available. The Active digital regulator can be disabled if external regulation is used.
Deep-Sleep	All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency (32 kHz) clock is available. 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.

11.1 Active Mode

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

11.2 Sleep Mode

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

11.3 Deep-Sleep Mode

In Deep-Sleep mode, the CPU, SRAM, UDB, and high-speed logic are in retention. The high-frequency clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (32 kHz) oscillator 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, I^2C slave) continue to operate. Interrupts from low-speed, asynchronous or low-power analog peripherals can cause a wakeup from Deep-Sleep mode. CTBm can also operate in this mode with reduced power and bandwidth. For details on power consumption and CTBm bandwidth, refer to the device datasheet.

The available wakeup sources are listed in Table 11-2.

11.4 Hibernate Mode

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

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

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

Wakeup from Hibernate mode is possible from a pin interrupt or a low-power comparator only. Wakeup from hibernate 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 95.

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 95.

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

Table 11-2. 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 mode. The Cortex-M0 can delay the transition into a low-power mode until the lowest priority ISR is exited (if the SLEEPONEXIT bit in the CM0 System Control Register is set).

The transition to Sleep 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 PSoC 4 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 11-2), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode. See the Interrupts chapter on page 45 and the Reset System chapter on page 101 for details.

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 11-3. Power Mode Register List

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

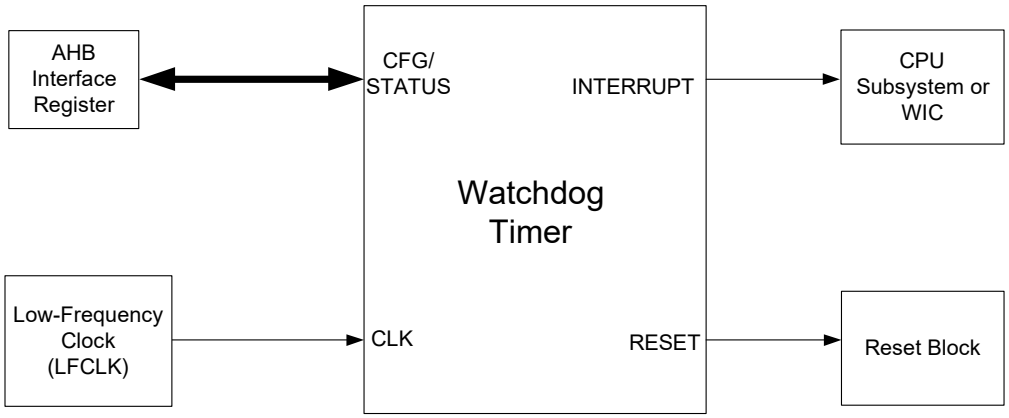
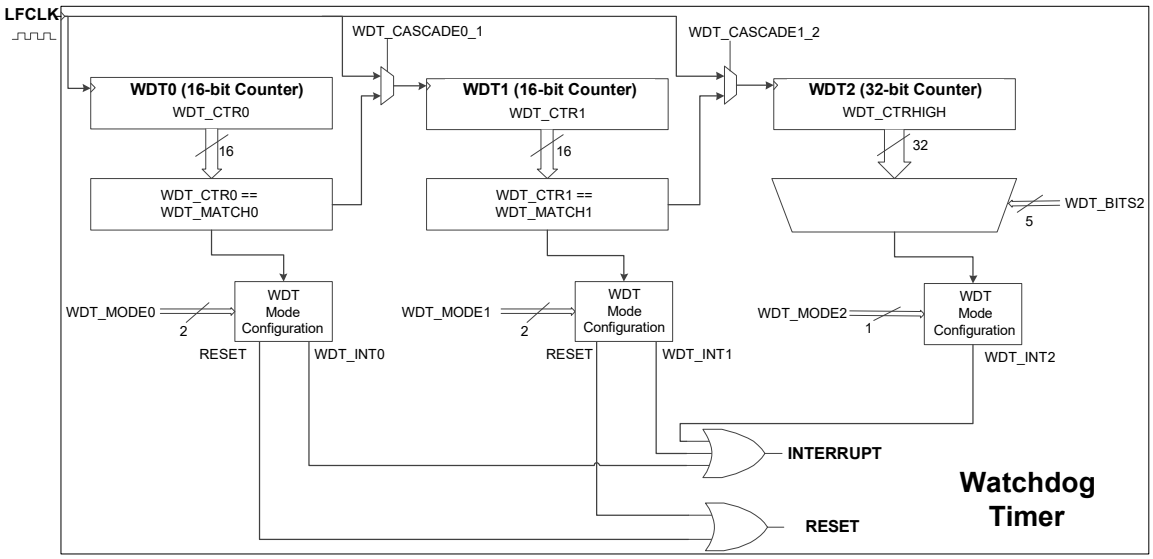


Figure 12-2. Watchdog Timer Internal Block Diagram



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 PSoC 4200L Registers 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 PSoC 4200L Registers 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 45 for details).
6. In the ISR, clear the WDT interrupt.

For more details on interrupts, see the Interrupts chapter on page 45.

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 PSoC 4200L Registers 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 PSoC 4200L Registers 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 on device power modes, see the Power Modes chapter on page 91.

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 101.

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

PSoC® 4 supports several types of resets that guarantee error-free operation during power up and allow the device to reset based on user-supplied external hardware or internal software reset signals. PSoC 4 also contains hardware to enable the detection of certain resets.

The reset system has these sources:

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

13.1.1 Power-on Reset

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

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

13.1.2 Brownout Reset

Brownout reset monitors the chip digital voltage supply V_{CCD} and generates a reset if V_{CCD} is below the minimum logic operating voltage specified in the device datasheet. BOD is available in all power modes 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 102.

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, XRES, 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 97.

13.1.4 Software Initiated Reset

Software initiated reset (SRES) is a mechanism that allows a software-driven reset. The Cortex-M0 application interrupt and reset control register (CM0_AIRCR) forces a device reset when a '1' is written into the SYSRESETREQ bit. CM0_AIRCR requires a value of A05F written to the top two bytes for writes. Therefore, write A05F0004 for the reset.

The RESET_SOFT status bit of the RES_CAUSE register is set when a software reset occurs. This bit remains set until cleared or until a POR, XRES, or 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 pin is ***active low*** – a high voltage on the pin has no effect and a low voltage causes a reset. The pin is pulled high inside the device. XRES is available as a dedicated pin in most of the devices. For detailed pinout, refer to the pinout section of the device datasheet.

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

XRES events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, 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 90.

The RESET_PROT_FAULT bit of the RES_CAUSE register is set when a protection fault occurs. This bit remains set until cleared or until a POR, XRES, or 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 93.

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 93.

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, external reset, or brownout detect.

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 the 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 its 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

PSoC® 4 offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and enabling flash protection provide a high level of security. In PSoC 4200L devices, 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 on flash row and chip protection, see the CY8C4xxxL Programming Specifications.

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

14.1 Features

The PSoC 4 device security system has the following features:

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

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

14.2 How It Works

14.2.1 Device Security

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

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

- **KILL mode**: The user may change the mode from OPEN to KILL. This mode removes all debug access to user code or memory, and the flash cannot be erased. Access to most registers is still available; debug access to registers to 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 PSoC 4 devices include a flexible flash-protection system that controls access to flash memory. This feature is designed to secure proprietary code, but it can also be used to protect against inadvertent writes to the bootloader portion of flash.

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

For more details, see the Nonvolatile Memory Programming chapter on page 325.

Table 14-1. Flash Protection Levels

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

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

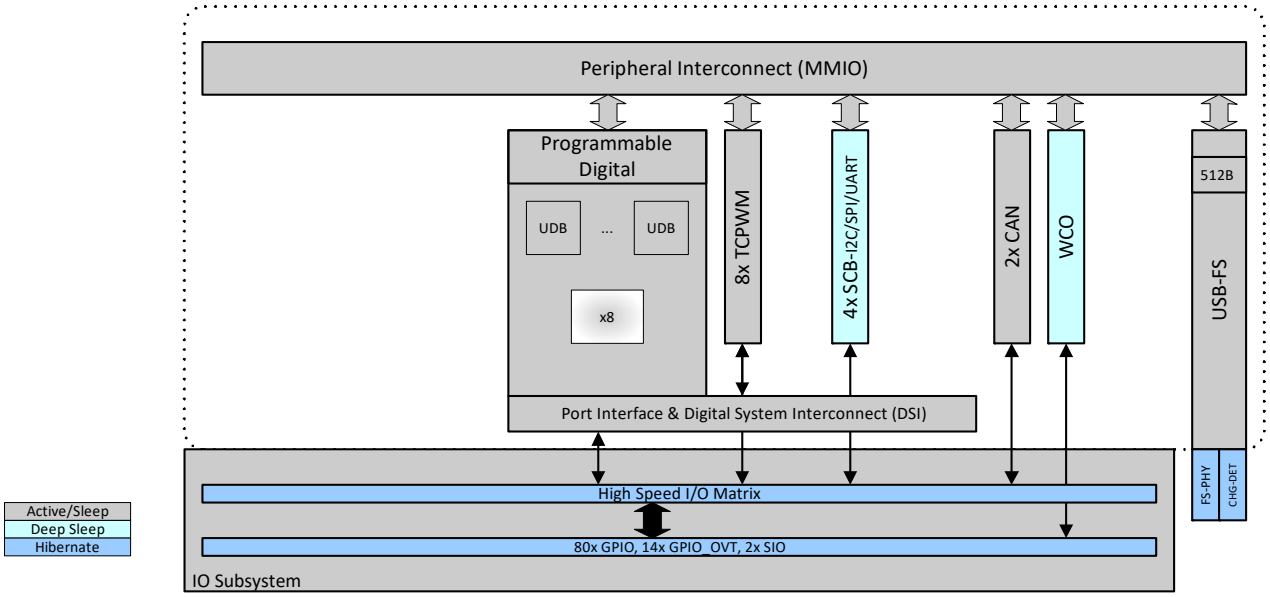
Section D: Digital System

This section encompasses the following chapters:

- Serial Communications Block (SCB) chapter on page 107
- Universal Digital Blocks (UDB) chapter on page 148
- Controller Area Network (CAN) chapter on page 190
- Universal Serial Bus (USB) chapter on page 207
- Timer, Counter, and PWM chapter on page 225

Top Level Architecture

Digital System Block Diagram



15. Serial Communications Block (SCB)

The Serial Communications Block (SCB) of PSoC® 4 supports three serial interface protocols: SPI, UART, and I²C. Only one of the protocols is supported by an SCB at any given time. PSoC 4 devices have four SCBs. Additional instances of the serial peripheral interface (SPI) and UART protocols can be implemented using the universal digital blocks (UDBs) in PSoC 4200L.

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
- Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
- 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 PSoC has to communicate with one or more SPI slave devices. The SPI slave mode can be used when the PSoC has to communicate with an SPI master device.

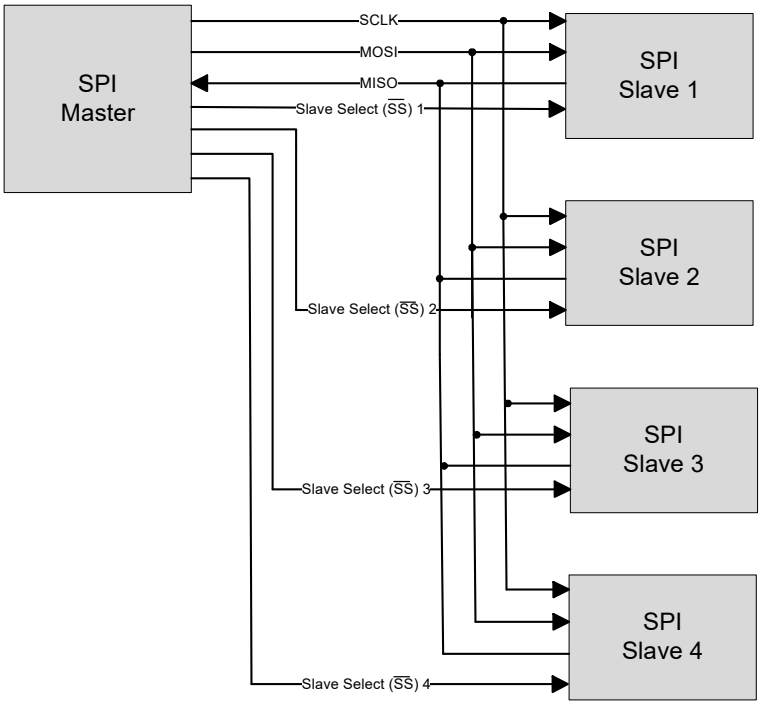
15.2.1 Features

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

15.2.2 General Description

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

Figure 15-1. SPI Example



A standard SPI interface consists of four signals as follows.

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

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

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

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

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

15.2.3 SPI Modes of Operation

15.2.3.1 Motorola SPI

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

Modes of Motorola SPI

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

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

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

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

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

Figure 15-2. SPI Motorola, 4 Modes

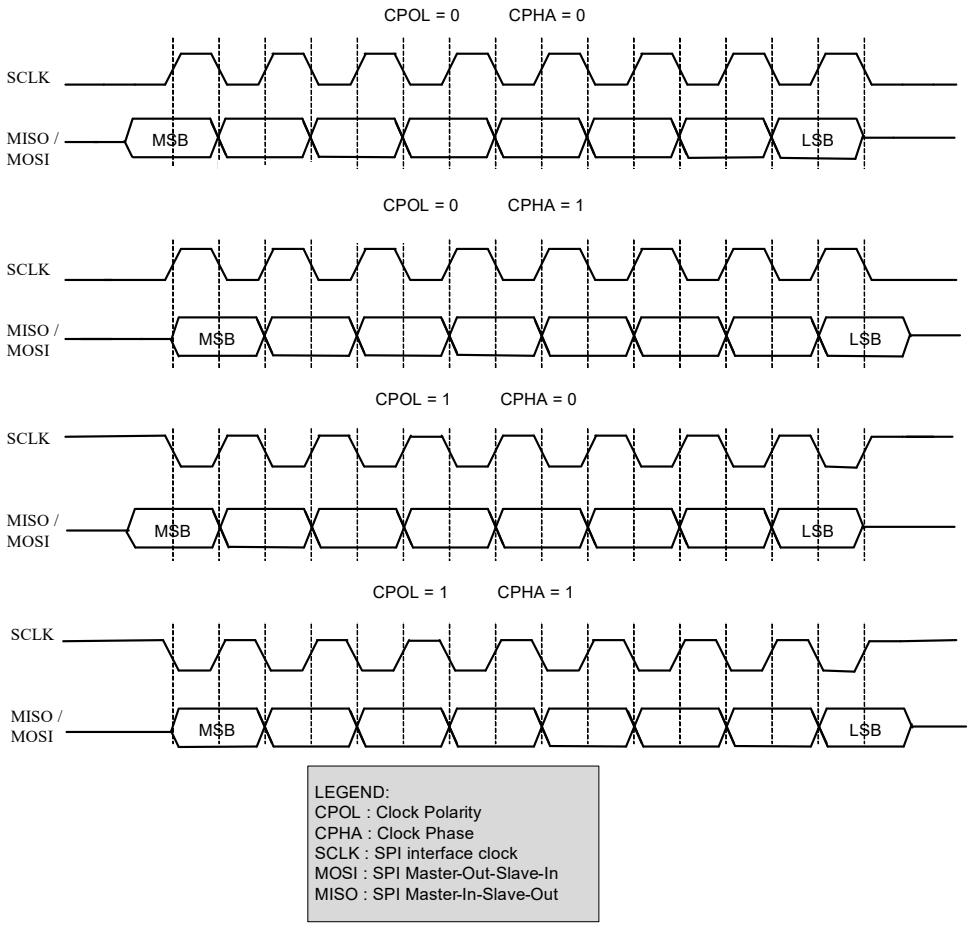
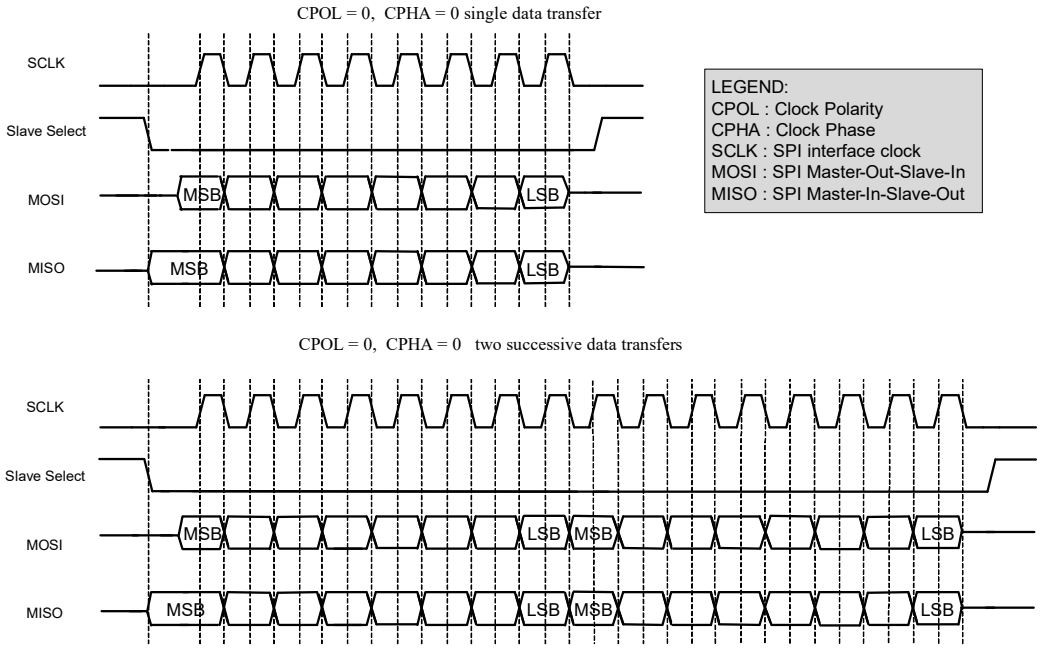


Figure 15-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 15-3. SPI Motorola Data Transfer Example



Configuring SCB for SPI Motorola Mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI Motorola mode by writing '00' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the CPHA and CPOL fields (bits 2 and 3 respectively) of the SCB_SPI_CTRL register.
4. Follow steps 2 to 4 mentioned in “Enabling and Initializing SPI” on page 116.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4200L Registers TRM.

15.2.3.2 Texas Instruments SPI

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

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

Figure 15-4. SPI TI Data Transfer Example

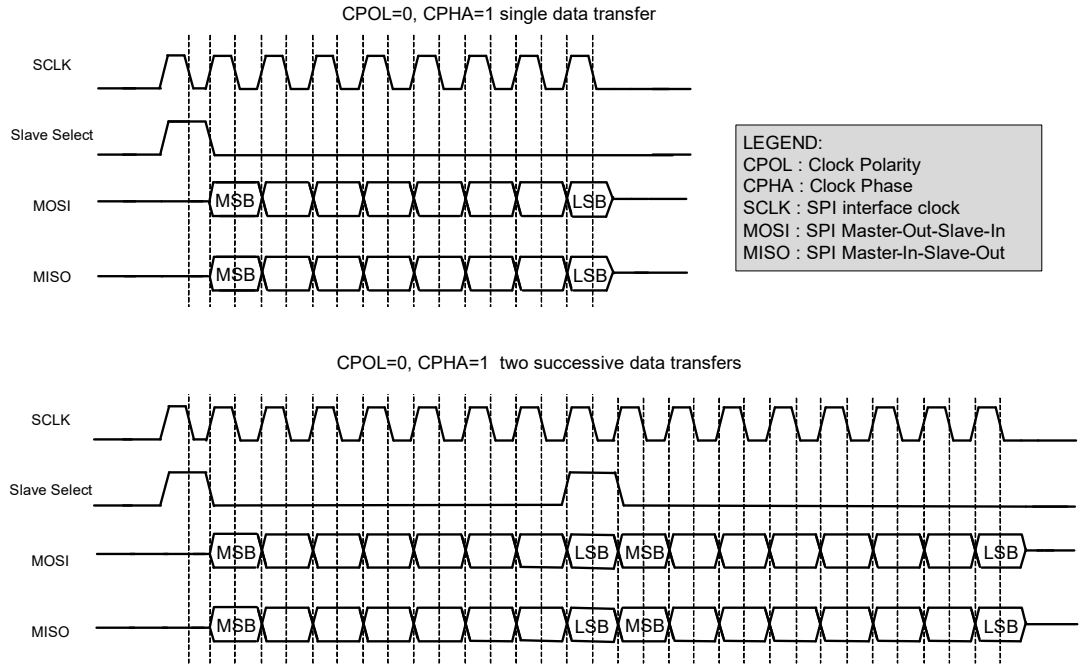
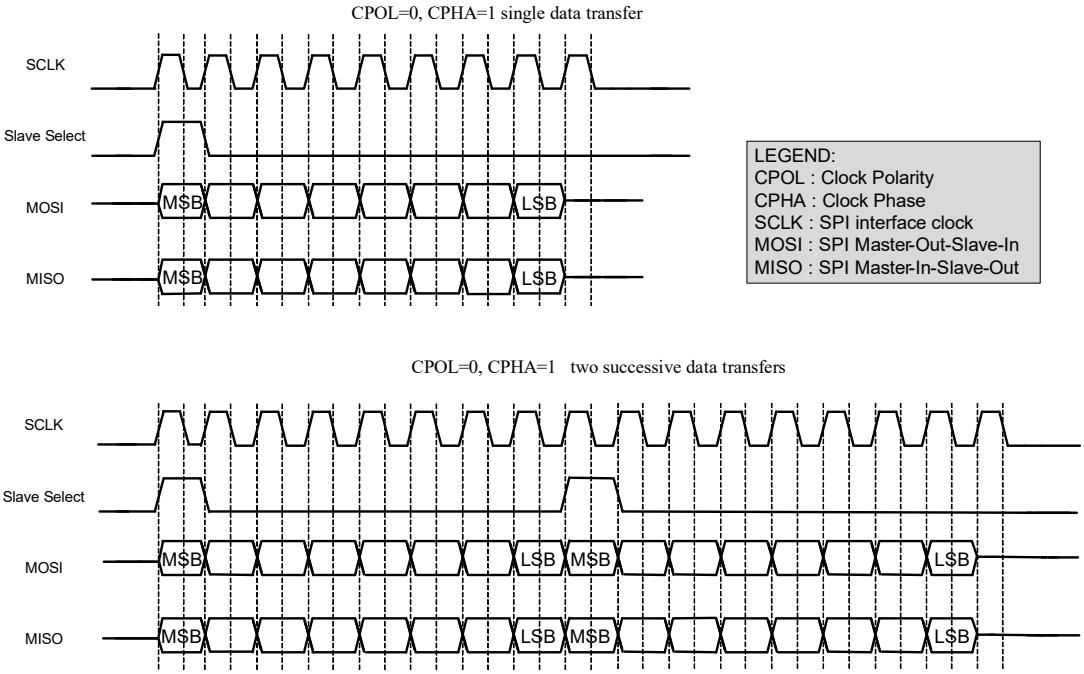


Figure 15-5 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

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 116.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4200L Registers TRM.

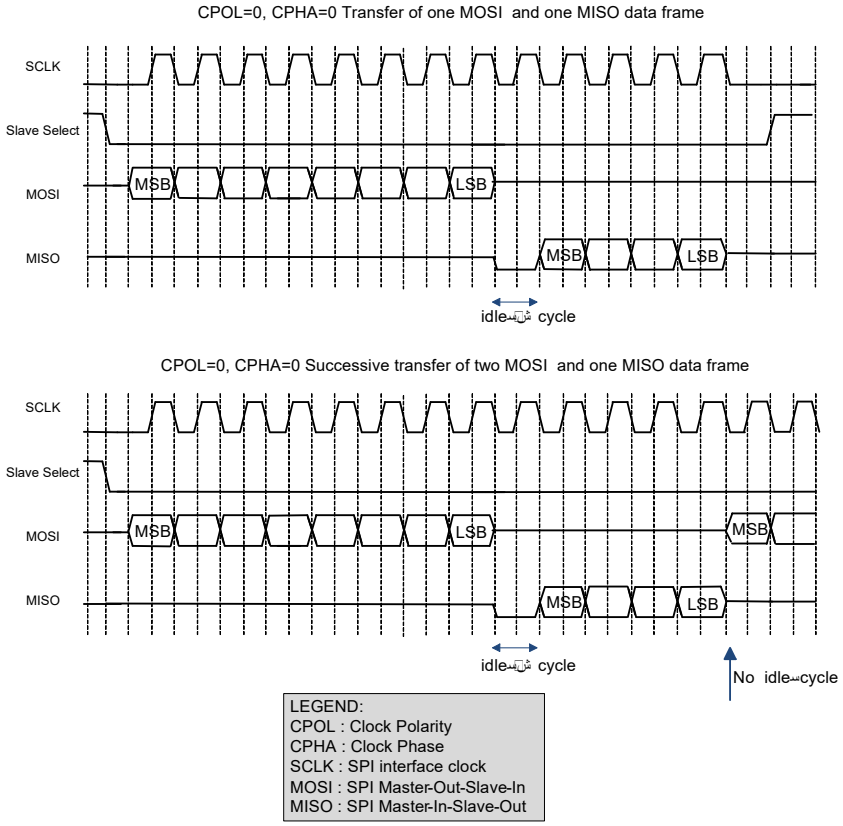
15.2.3.3 National Semiconductors SPI

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

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

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

Figure 15-6. SPI NS Data Transfer Example



Configuring SCB for SPI NS Mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in “Enabling and Initializing SPI” on page 116.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the PSoC 4200L Registers TRM.

15.2.4 Using SPI Master to Clock Slave

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

15.2.5 Easy SPI Protocol

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

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

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

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

15.2.5.1 EZ Address Write

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

15.2.5.2 Memory Array Write

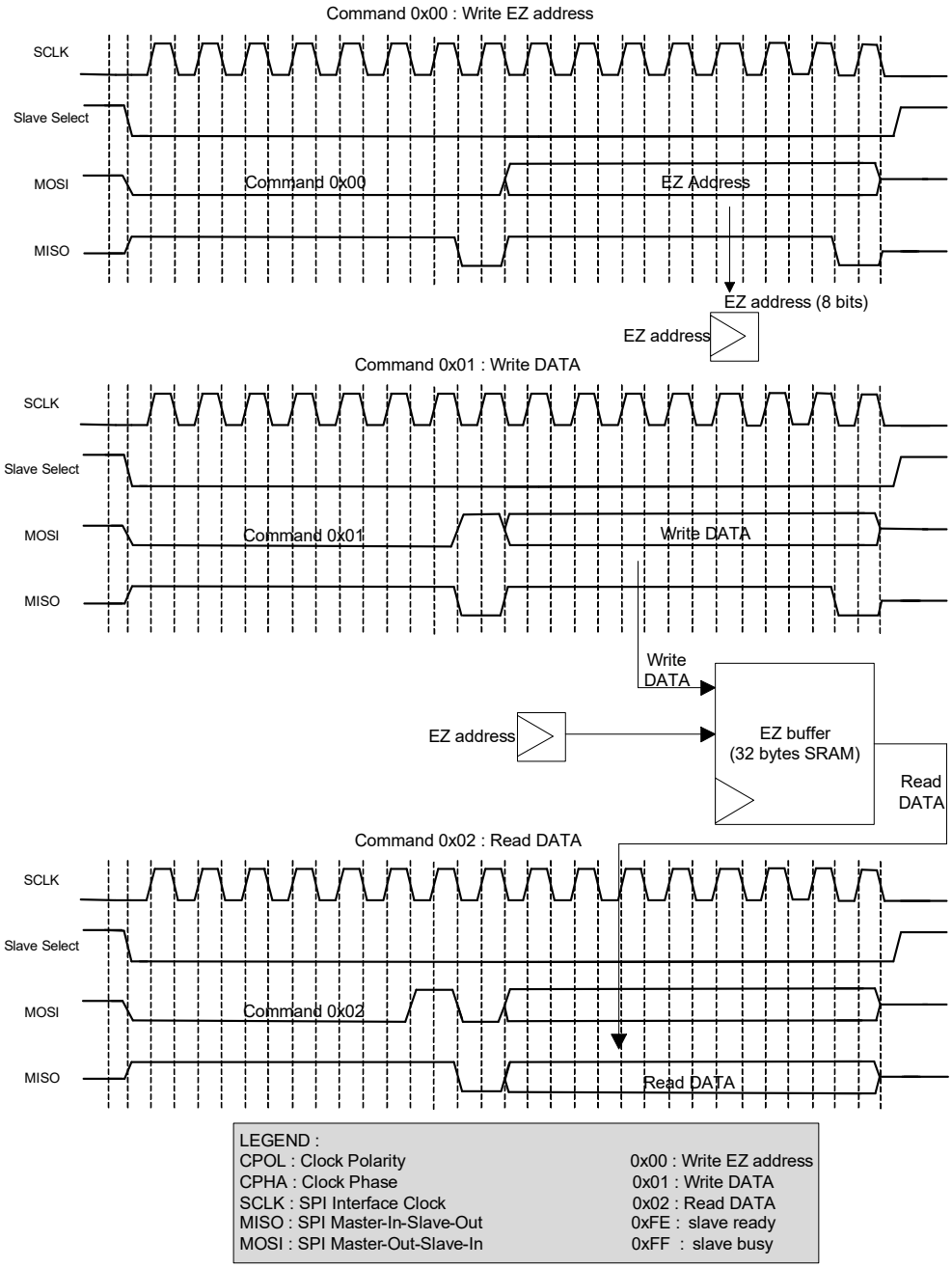
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional write data bytes on the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

15.2.5.3 Memory Array Read

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

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

Figure 15-7. EZSPI Example



15.2.5.4 Configuring SCB for EZSPI Mode

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

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCB_CTRL register.
2. Use continuous transmission mode for the transmitter by writing '1' to the CONTINUOUS bit of SCB_SPI_CTRL register.
3. Follow steps 2 to 5 mentioned in “Enabling and Initializing SPI” on page 116.

Note that PSoC Creator does all this automatically with the help of Component customizers. For more information on these registers, see the PSoC 4200L Registers TRM.

15.2.6 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 15-1. For more information on these registers, see the PSoC 4200L Registers TRM.

Table 15-1. SPI Registers

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

15.2.7 SPI Interrupts

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

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

The SPI supports interrupts on the following events:

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

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

15.2.8 Enabling and Initializing SPI

The SPI must be programmed in the following order:

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

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

Table 15-2. SCB_CTRL Register

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

Table 15-3. SCB_SPI_CTRL Register

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

Table 15-4. SCB_TX_CTRL/SCB_RX_CTRL Registers

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

Table 15-5. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

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

15.2.9 Internally and Externally Clocked SPI Operations

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

Internally clocked operation uses the high-frequency clock (HFCLK) of the system. For more information on system clocking, see the Clocking System chapter on page 72. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCB_CTRL register specify the oversampling.

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

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

Table 15-6. SPI Slave Maximum Data Rates

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

Externally clocked operation is limited to:

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

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

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

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

These two register fields determine the functional behavior of SPI. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. Table 15-7 and Table 15-8 describes 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)				
System Power Mode	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 0	EC_AM_MODE = 1
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using 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 91)			
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				
System Power Mode	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 0	EC_AM_MODE = 1
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using 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 91)			
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

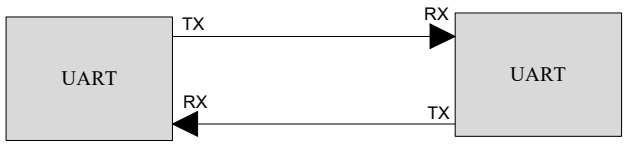
15.3.1 Features

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

15.3.2 General Description

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

Figure 15-8. UART Example



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

Three different serial interface protocols are supported:

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

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

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

15.3.3 UART Modes of Operation

15.3.3.1 Standard Protocol

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

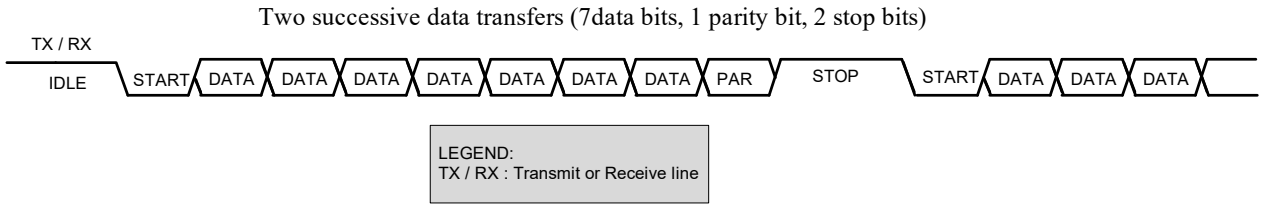
Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample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

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

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

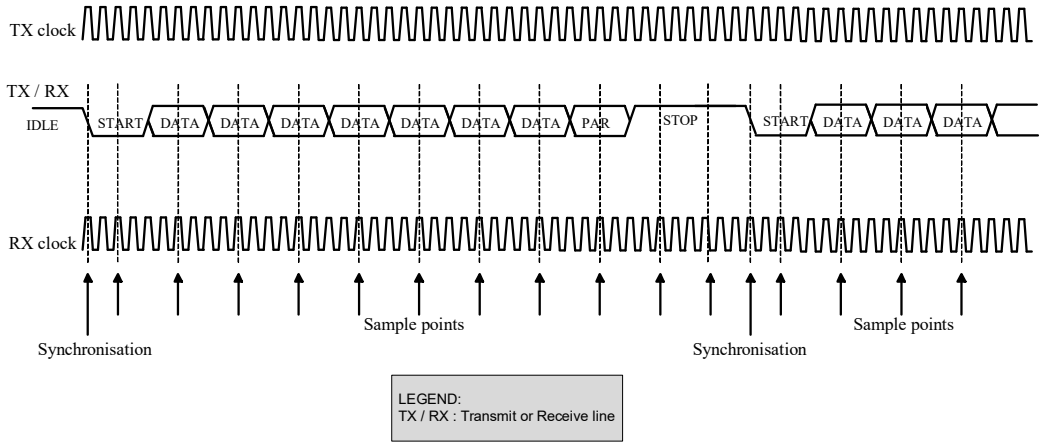
Figure 15-9 illustrates the UART protocol.

Figure 15-9. UART, Standard Protocol Example



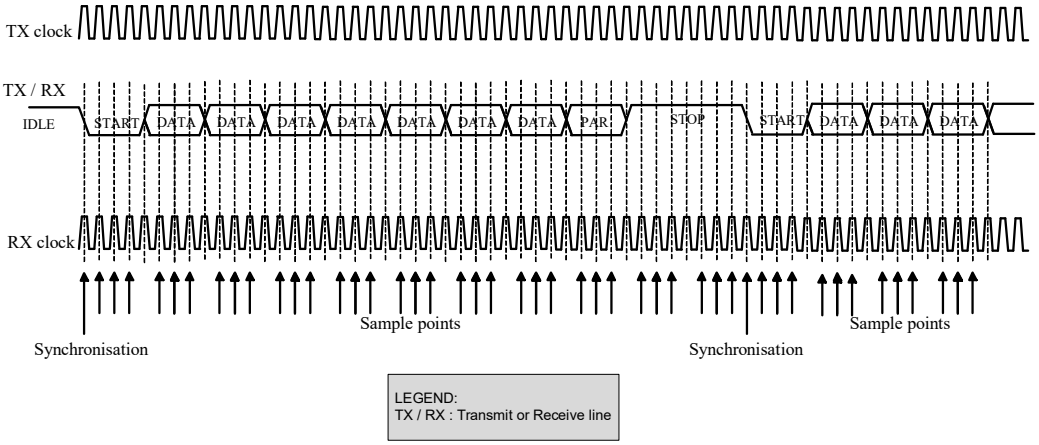
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. Figure 15-10 illustrates this.

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



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

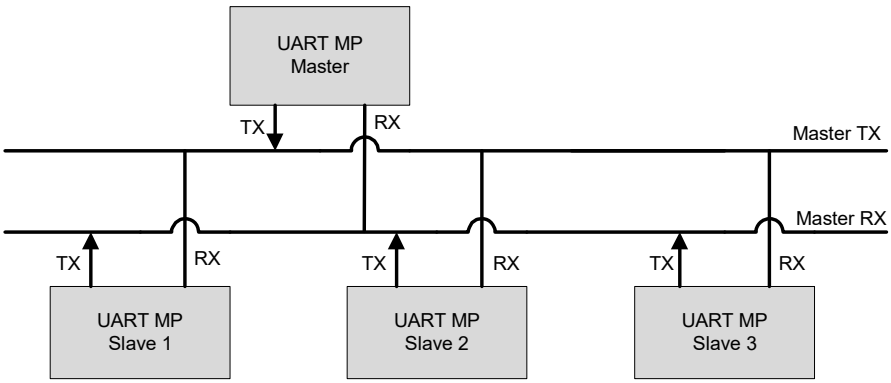
Figure 15-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor Mode

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

Figure 15-12. UART MP Mode Bus Connections



The main properties of UART_MP mode are:

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

Figure 15-13. UART MP Data Frame

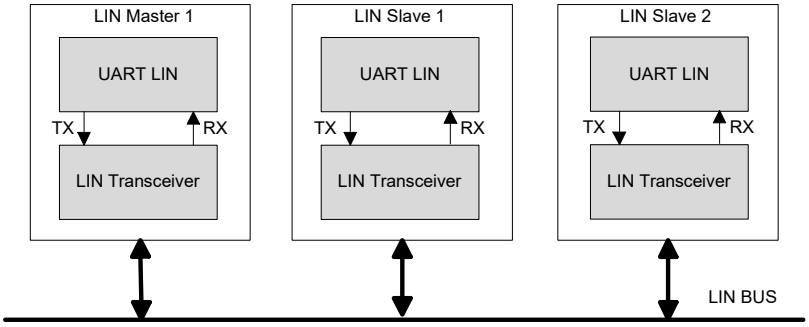


The SCB can be used as either master or slave device in UART_MP mode. Both SCB_TX_CTRL and SCB_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device,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

UART Local Interconnect Network (LIN) Mode

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

Figure 15-14. UART_LIN and LIN Transceiver

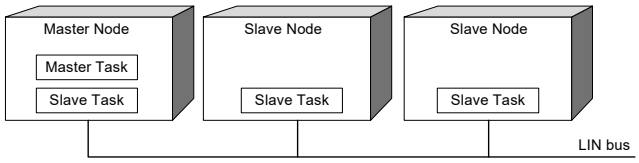


LIN protocol defines two tasks:

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

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

Figure 15-15. LIN Bus Nodes and Tasks

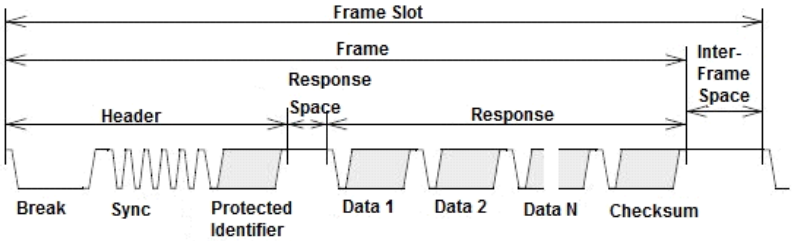


LIN Frame Structure

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

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

Figure 15-16. LIN Frame Structure

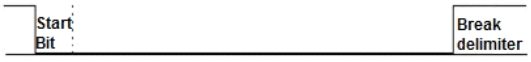


In LIN protocol communication, the least significant bit (LSB) of the data is sent first and the most significant bit (MSB) last. The start bit is encoded as zero and the stop bit is encoded as one. The following sections describe all the byte fields in the LIN frame.

Break Field

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

Figure 15-17. LIN Break Field



Sync Field

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

Figure 15-18. LIN Sync Field



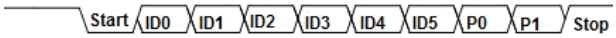
Protected identifier (PID) Field

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

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

Figure 15-19 shows the PID field structure.

Figure 15-19. PID Field



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

Checksum

The checksum is the last byte field in the LIN frame. It is calculated by inverting the 8-bit sum along with carryover of all data bytes only or the 8-bit sum with the carryover of all data bytes and the PID field. The two types of checksums in LIN frames 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 129.

Note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4200L Registers TRM.

15.3.3.2 SmartCard (ISO7816)

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

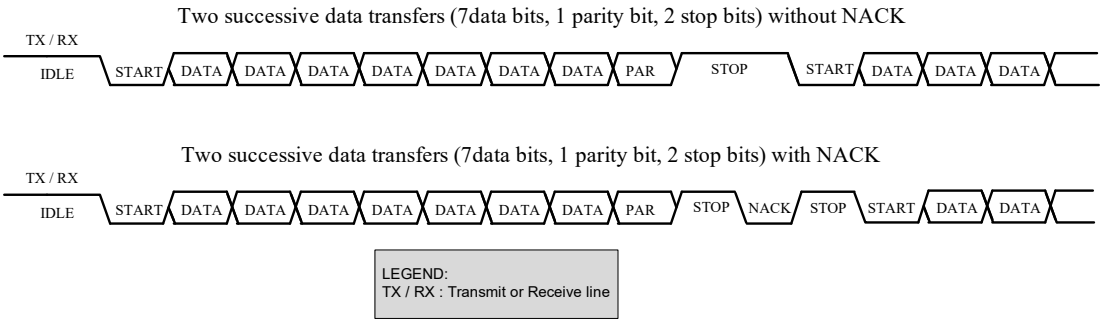
The SmartCard transfer is similar to a UART transfer, with the addition of a negative acknowledgement (NACK) that may be sent from the receiver to the transmitter. A NACK is always '0'. Both master and slave may drive the same line, although never at the same time.

A SmartCard transfer has the transmitter drive the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus. Releasing results in the line being '1' (the value of a stop bit). After one bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of '0') for one bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by one bit transfer period. For this protocol to work, the stop period should be longer than one bit transfer period. Note that a data transfer with a NACK takes one bit transfer period

longer, than a data transfer without a NACK. Typically, implementations use a tristate driver with a pull-up resistor, such that when the line is not transmitting data or transmitting the Stop bit, its value is '1'.

Figure 15-20 illustrates the SmartCard protocol.

Figure 15-20. SmartCard Example



The communication Baud rate for ISO7816 is given as:

Baud rate= f_{7816} × (D/F)

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

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

Configuring SCB as UART SmartCard Interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4200L Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as a SmartCard protocol by writing '01' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Follow steps 2 to 5 described in “Enabling and Initializing UART” on page 129.

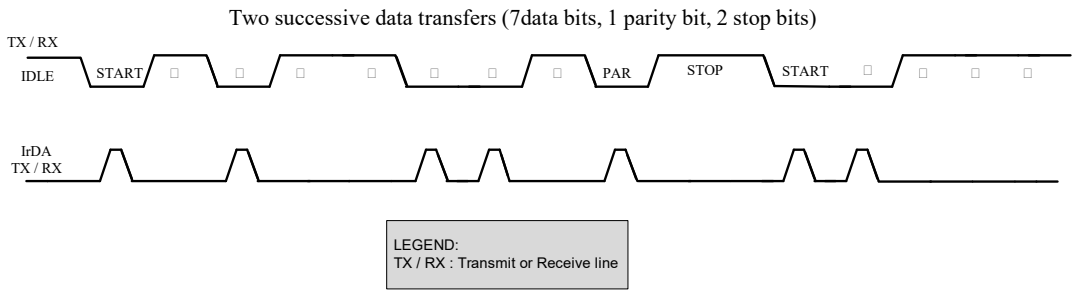
15.3.3.3 IrDA

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

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

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

Figure 15-21. IrDA Example



Configuring the SCB as UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order; note that PSoC Creator does all this automatically with the help of GUIs. For more information on these registers, see the PSoC 4200L Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCB_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the MODE (bits [25:24]) of the SCB_UART_CTRL register.
3. Enable the Median filter on the input interface line by writing ‘1’ to MEDIAN (bit 9) of the SCB_RX_CTRL register.
4. Configure the SCB as described in “Enabling and Initializing UART” on page 129.

15.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 15-9. For more information on these registers, see the PSoC 4200L Registers TRM.

Table 15-9. UART Registers

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

15.3.5 UART Interrupts

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

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

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

15.3.6 Enabling and Initializing UART

The UART must be programmed in the following order:

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

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

Table 15-10. SCB_UART_CTRL Register

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

Table 15-11. SCB_TX_CTRL/SCB_RX_CTRL Registers

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

Table 15-12. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL Registers

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

Table 15-13. SCB_CTRL Register

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

15.4 Inter Integrated Circuit (I^2C)

This section explains the I^2C implementation in PSoC 4. For more information on the I^2C protocol specification, refer to the UM10204, I^2C-bus specification and user manual.

15.4.1 Features

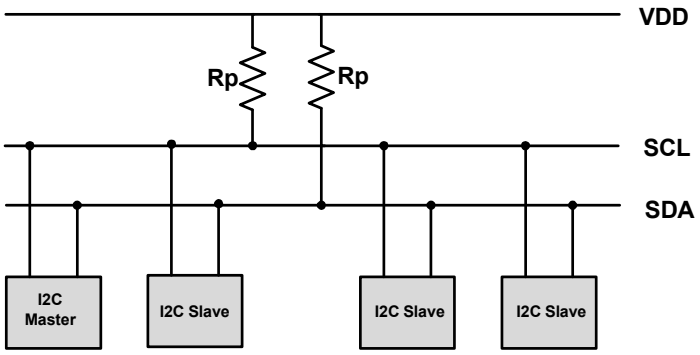
This block supports the following features:

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

15.4.2 General Description

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

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



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

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

I^2C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address. PSoC 4 also supports 10-bit address matching for I^2C with firmware support.

15.4.3 Terms and Definitions

Table 15-14 explains the commonly used terms in an 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
Slave	The device addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

15.4.3.1 Clock Stretching

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

15.4.3.2 Bus Arbitration

The 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 I²C bus bit format during data transfer.

Table 15-16. I²C Bus Events Terminology

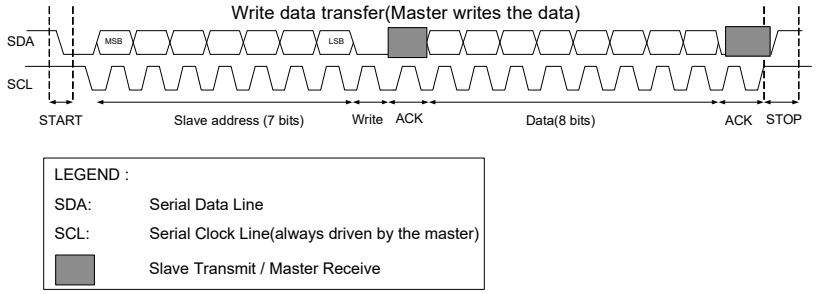
Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH
ACK	The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse, after the transmitter transmits each byte. This 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 140, "Slave Mode Transfer Examples" on page 142, and "Multi-Master Mode Transfer Example" on page 146.

15.4.4.1 Write Transfer

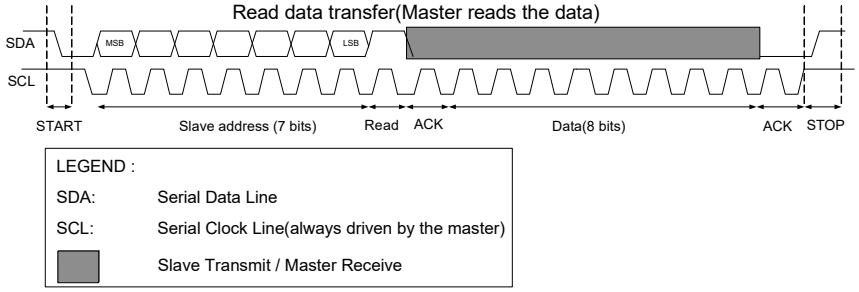
Figure 15-23. Master Write Data Transfer



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

15.4.4.2 Read Transfer

Figure 15-24. Master Read Data Transfer



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

15.4.5 Easy I2C (EZI2C) Protocol

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

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

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

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

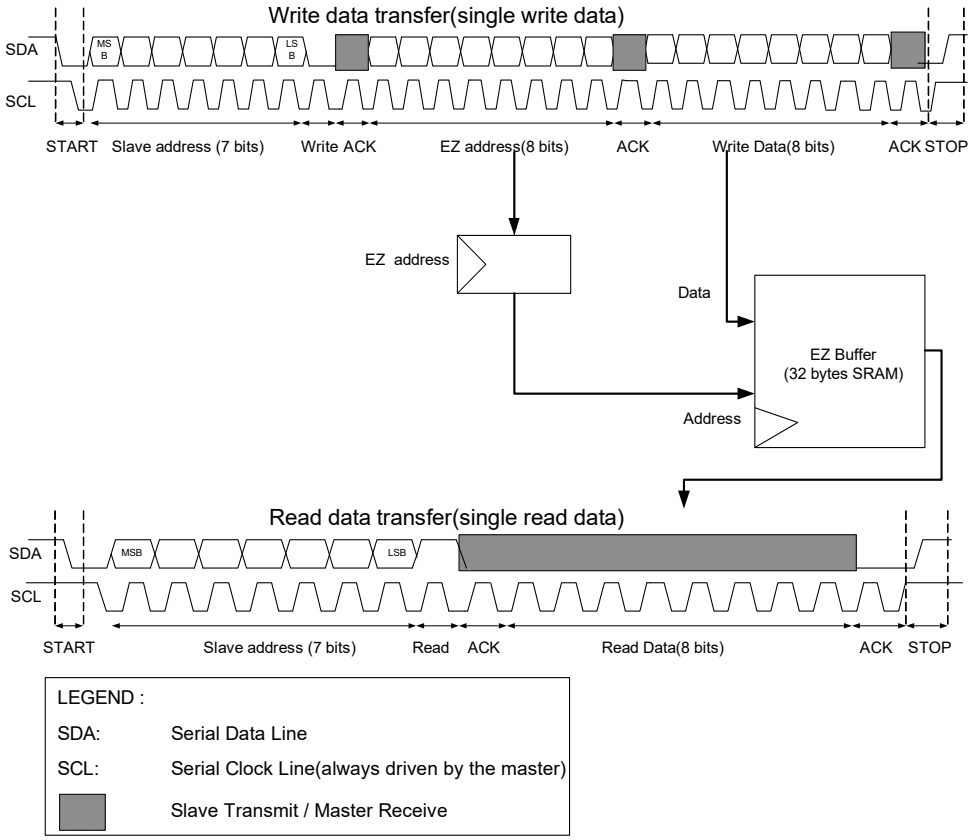
15.4.5.1 Memory Array Write

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

15.4.5.2 Memory Array Read

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

Figure 15-25. 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 SCB 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.
SCB_RX_CTRL	Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.
SCB_RX_FIFO_CTRL	Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.
SCB_RX_FIFO_STATUS	Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.
SCB_RX_FIFO_RD	Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO; behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCB_RX_FIFO_RD_SILENT	Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCB_RX_MATCH	Stores slave device address and is also used as slave device address MASK.
SCB_EZ_DATA	Holds the data in an EZ memory location.

Note Detailed descriptions of the I²C register bits are available in the PSoC 4200L Registers TRM.

15.4.7 I2C Interrupts

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

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

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

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the PSoC 4200L Registers TRM.

15.4.8 Enabling and Initializing the I2C

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

15.4.8.1 I2C Standard (Non-EZ) Mode Configuration

The I2C interface must be programmed in the following order.

1. Program protocol specific information using the SCB_I2C_CTRL register according to Table 15-18. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB_TX_CTRL and SCB_RX_CTRL registers, as shown in Table 15-19.
   a. Specify the data frame width.
   b. Specify that MSB is the first bit to be transmitted/received.
3. Program transmitter and receiver FIFO using the SCB_TX_FIFO_CTRL and SCB_RX_FIFO_CTRL registers, respectively, as shown in Table 15-20.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCB_CTRL register to enable the I2C block and select the I2C mode. These register bits are shown in Table 15-21. For a complete description of the I2C registers, see the PSoC 4200L Registers TRM.

Table 15-18. SCB_I2C_CTRL Register

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

Table 15-19. SCB_TX_CTRL/SCB_RX_CTRL Register

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

Table 15-20. SCB_TX_FIFO_CTRL/SCB_RX_FIFO_CTRL

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

Table 15-21. SCB_CTRL Registers

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

15.4.8.2 EZI2C Mode Configuration

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

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

15.4.9 Internal and External Clock Operation in I2C

The I2C block supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the PSoC system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the Clocking System chapter on page 72.

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.

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

15.4.9.1 I2C Non-EZ Mode of Operation

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

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

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

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

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

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

<table>
<tr><th colspan="5">I2C (Non-EZ) Mode</th></tr>
<tr><th rowspan="2">System Power Mode</th><th colspan="2">EC_OP_MODE = 0</th><th colspan="2">EC_OP_MODE = 1</th></tr>
<tr><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th><th>EC_AM_MODE = 0</th><th>EC_AM_MODE = 1</th></tr>
<tr><td>Active and Sleep</td><td>Address match using internal clock.
Operation using internal clock.</td><td>Address match using external clock.
Operation using internal clock.</td><td colspan="2" rowspan="2">Not supported</td></tr>
<tr><td>Deep-Sleep</td><td>Not supported</td><td>Address match using external clock.
Operation using internal clock.</td></tr>
<tr><td>Hibernate</td><td colspan="4" rowspan="2">The SCB is not available in these modes (see the Power Modes chapter on page 91).</td></tr>
<tr><td>Stop</td></tr>
</table>

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

15.4.9.2 I2C EZ Operation Mode

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

nally clocked logic (rest of the operation). The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

Table 15-23. 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.

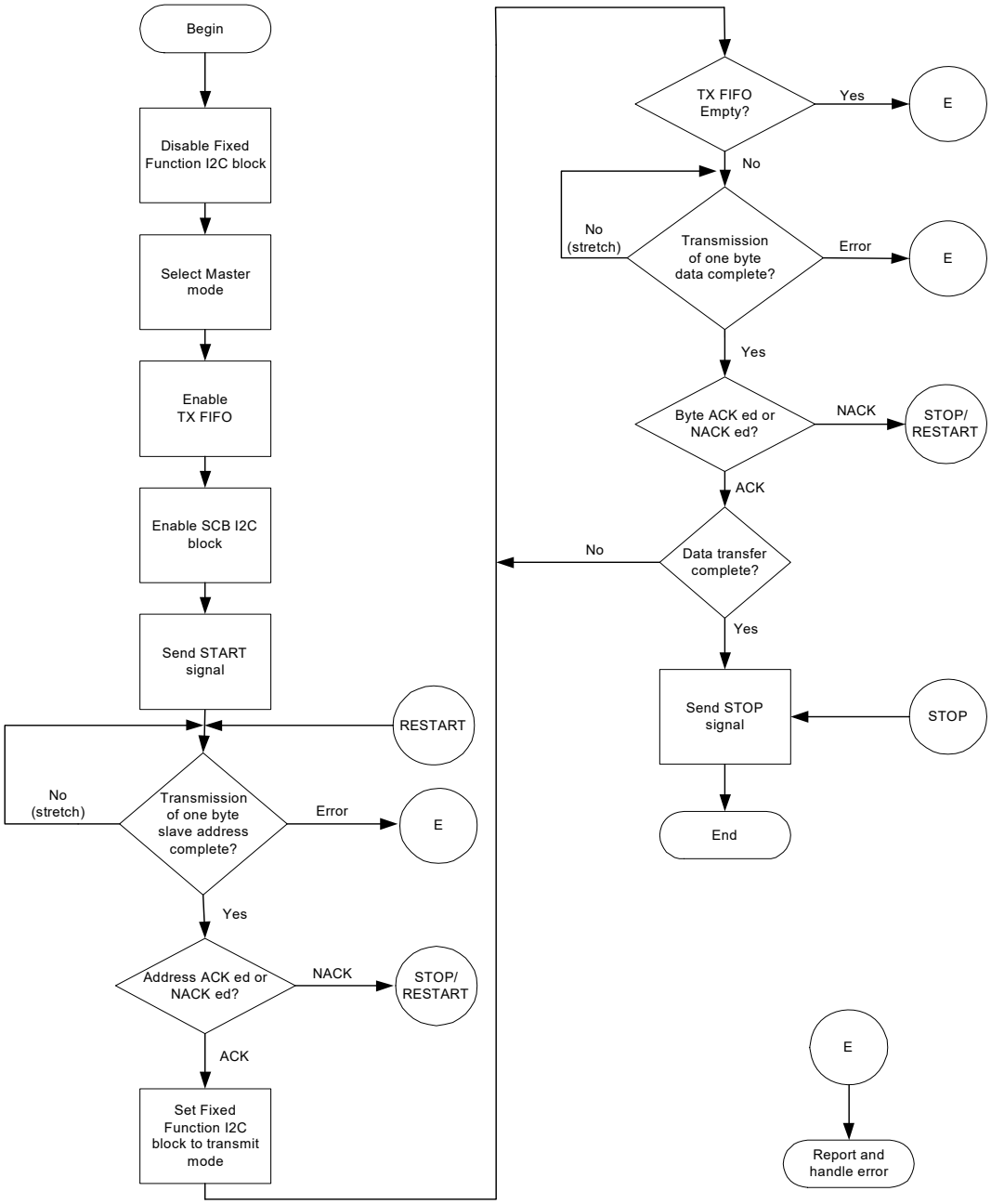
Note The interrupt bit WAKE_UP (bit 0) of the SCB_INTR_I2C_EC register must be enabled for the I2C to wake up the device on slave address match while switching to the Sleep mode.

15.4.11 Master Mode Transfer Examples

Master mode transmits or receives data.

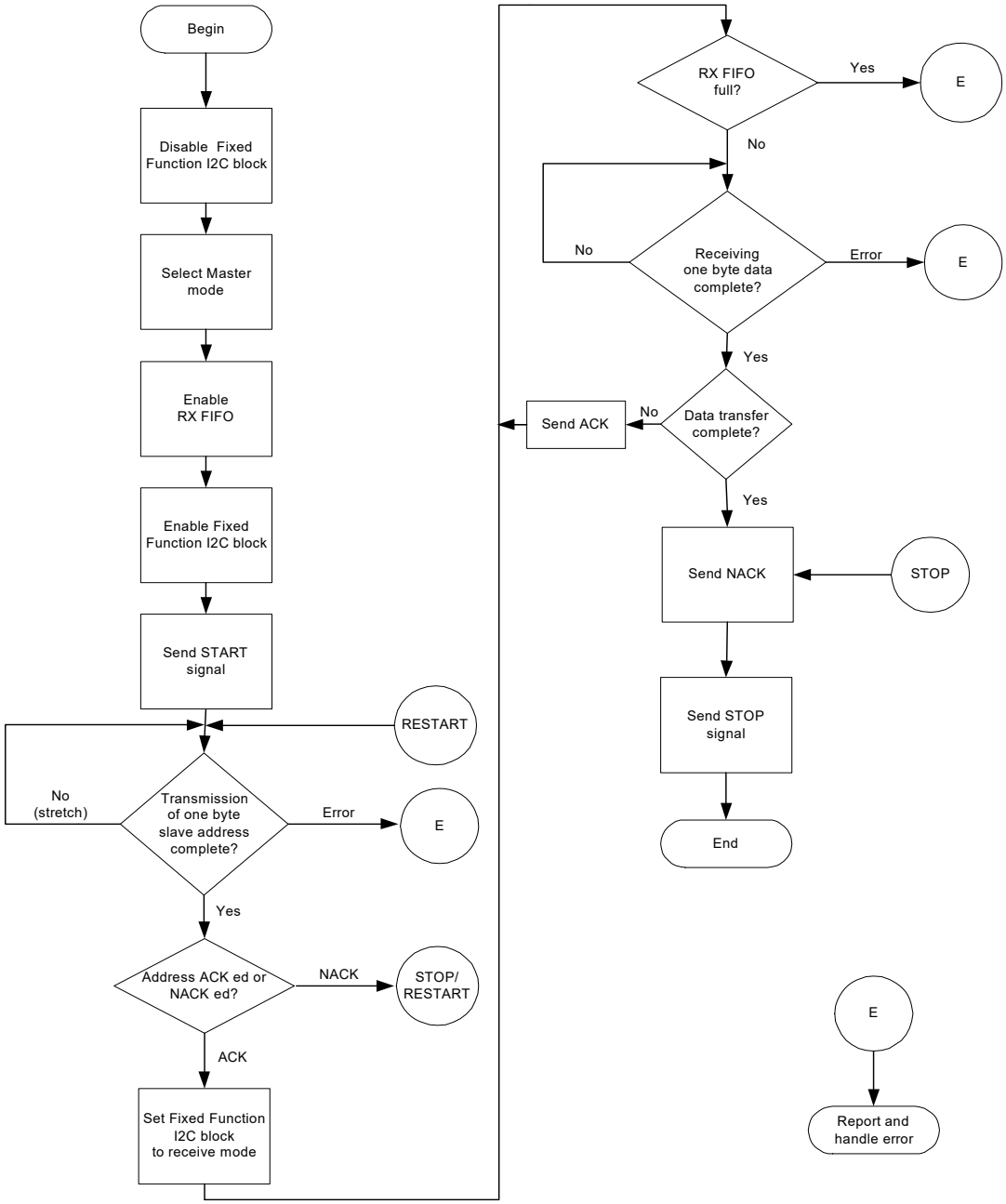
15.4.11.1 Master Transmit

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

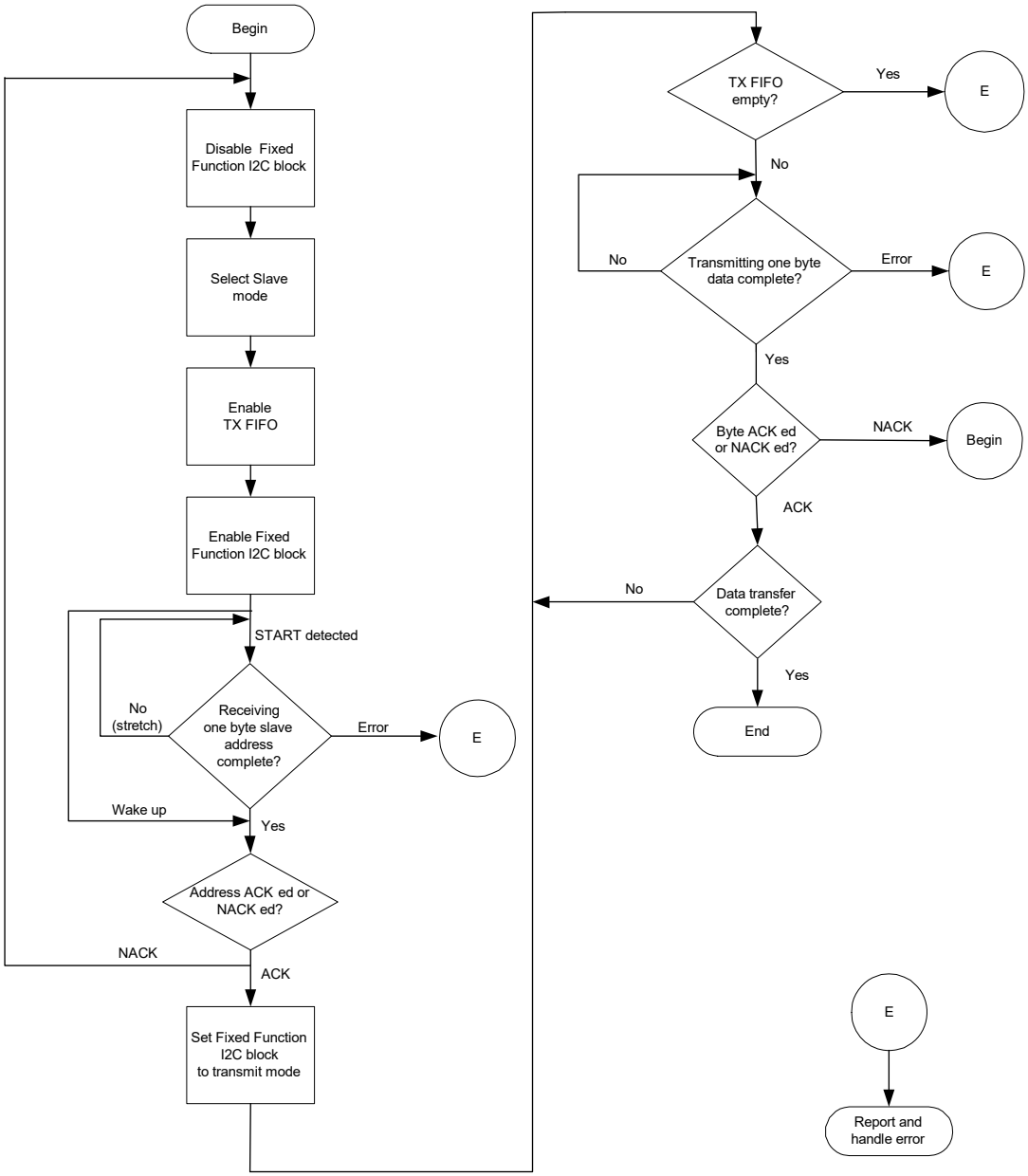


15.4.12 Slave Mode Transfer Examples

Slave mode transmits or receives data.

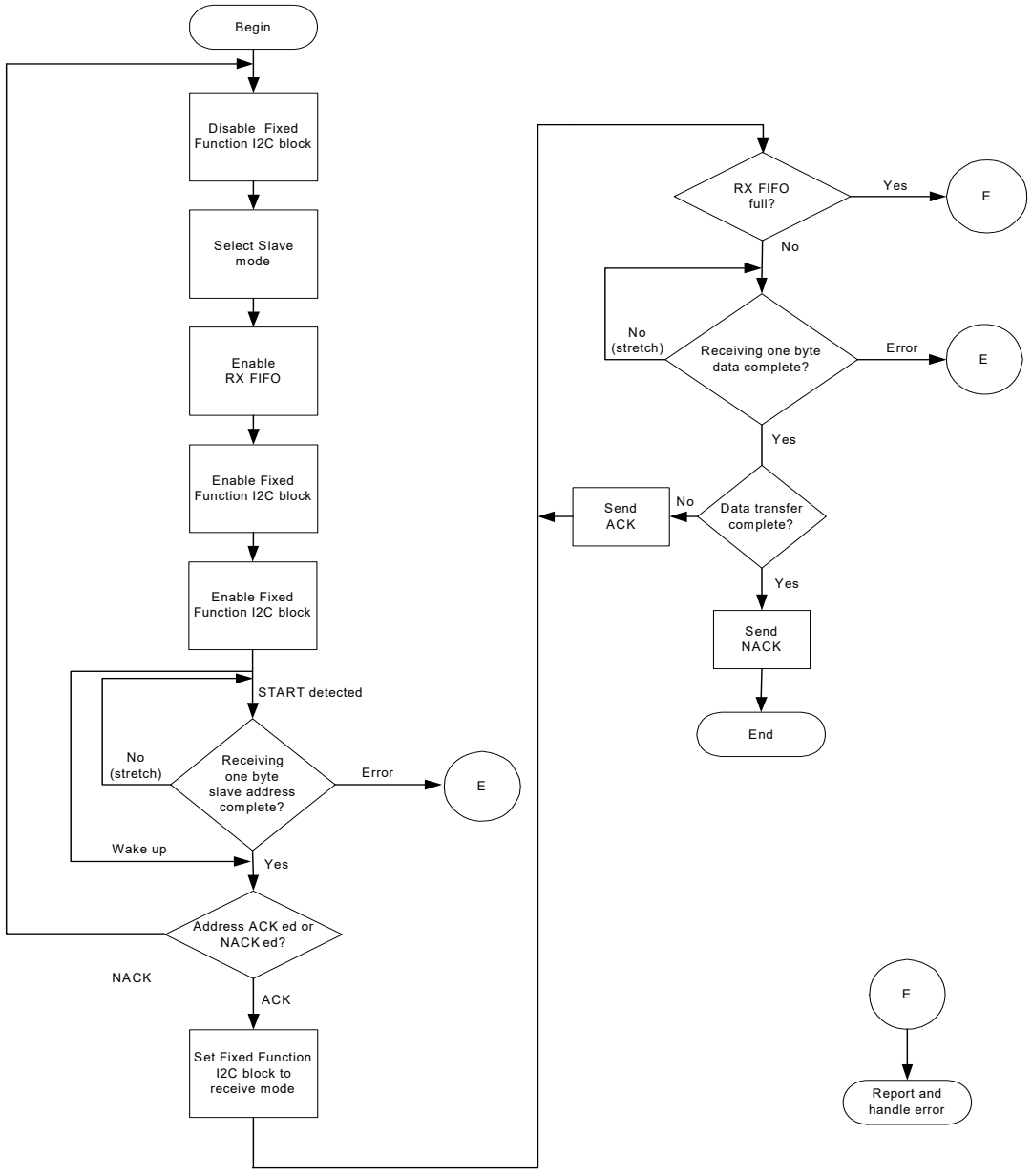
15.4.12.1 Slave Transmit

Figure 15-28. Slave Mode Write Operation Flow Chart



15.4.12.2 Slave Receive

Figure 15-29. Slave Mode Read Operation Flow Chart

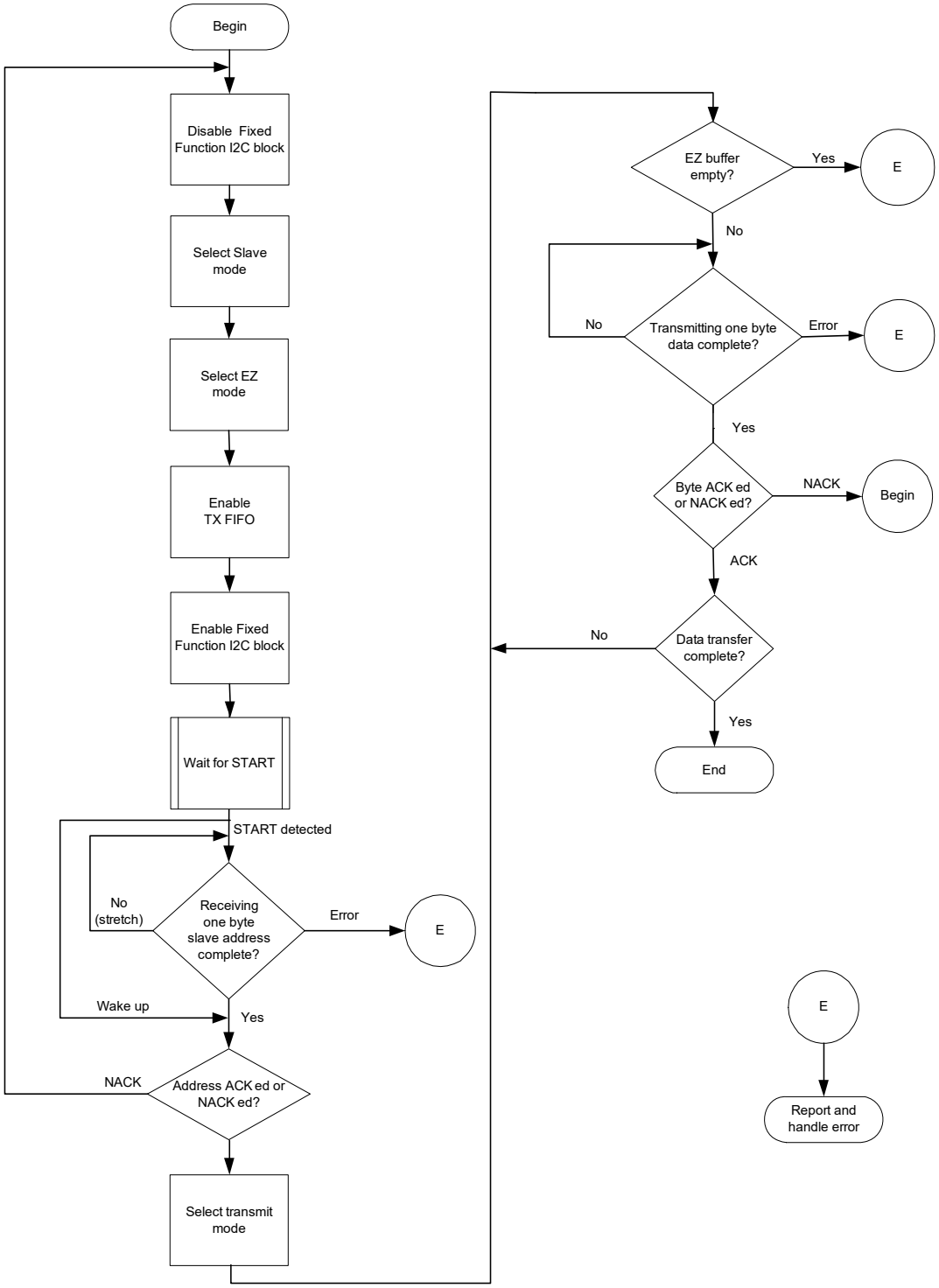


15.4.13 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

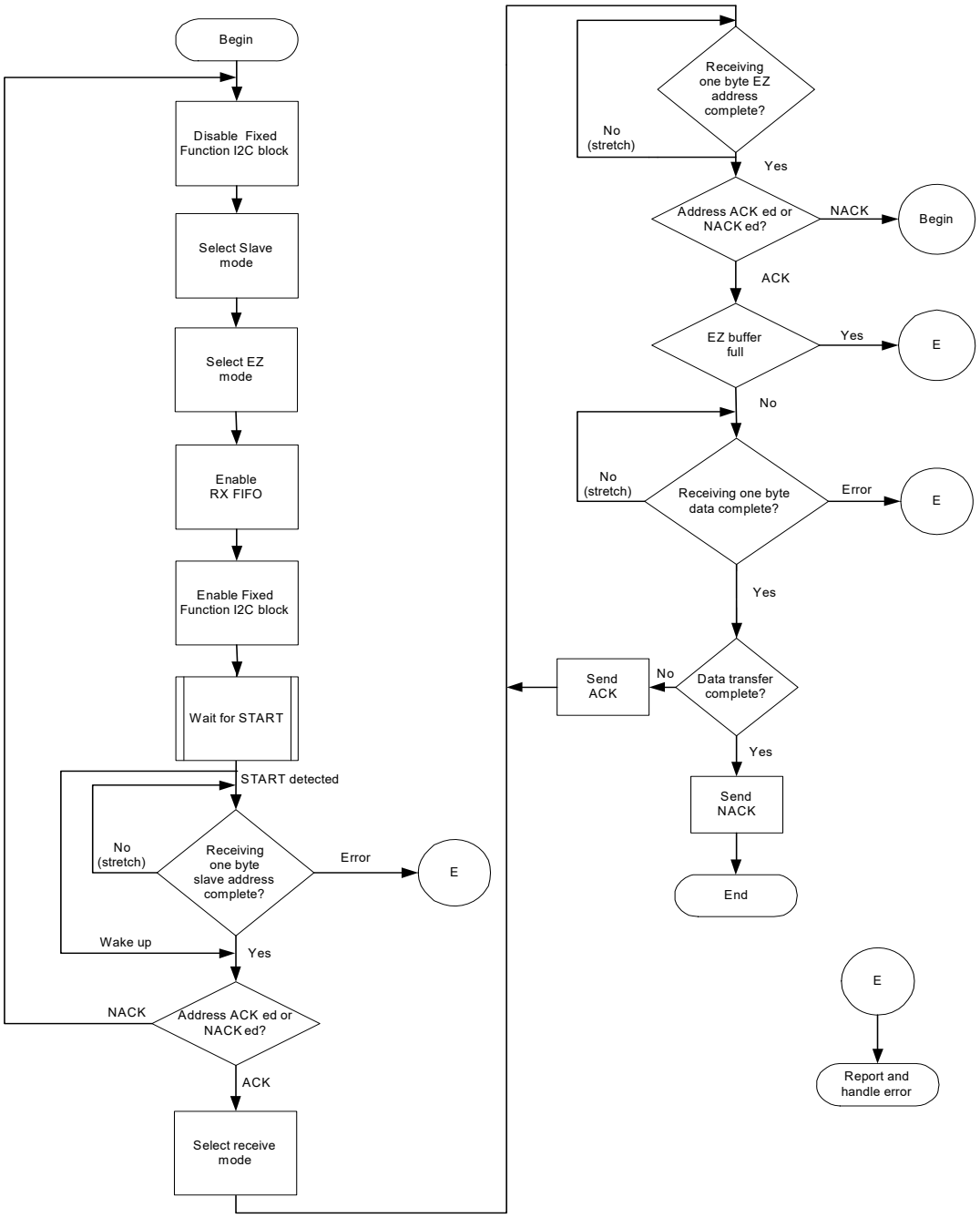
15.4.13.1 EZ Slave Transmit

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

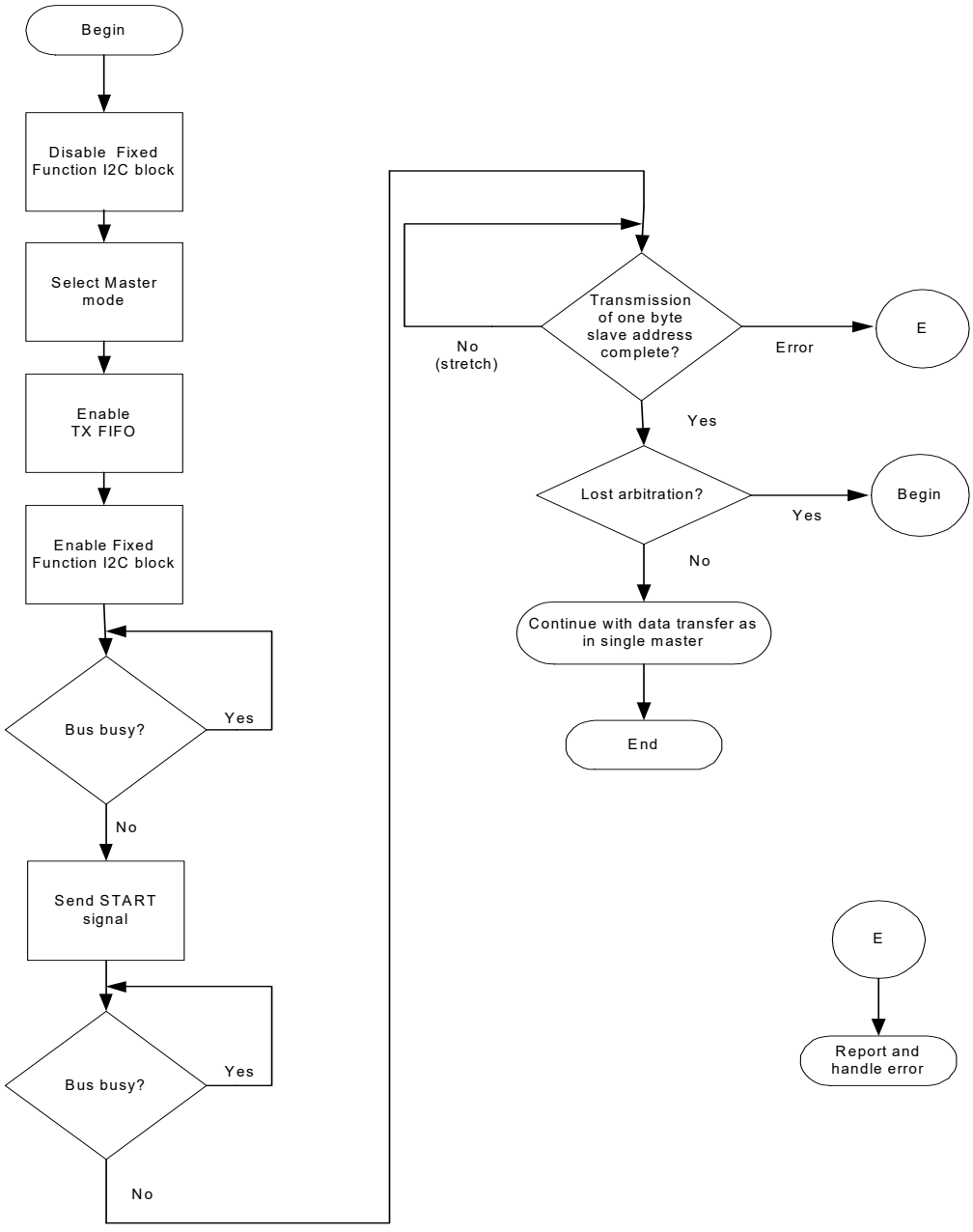


15.4.14 Multi-Master Mode Transfer Example

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

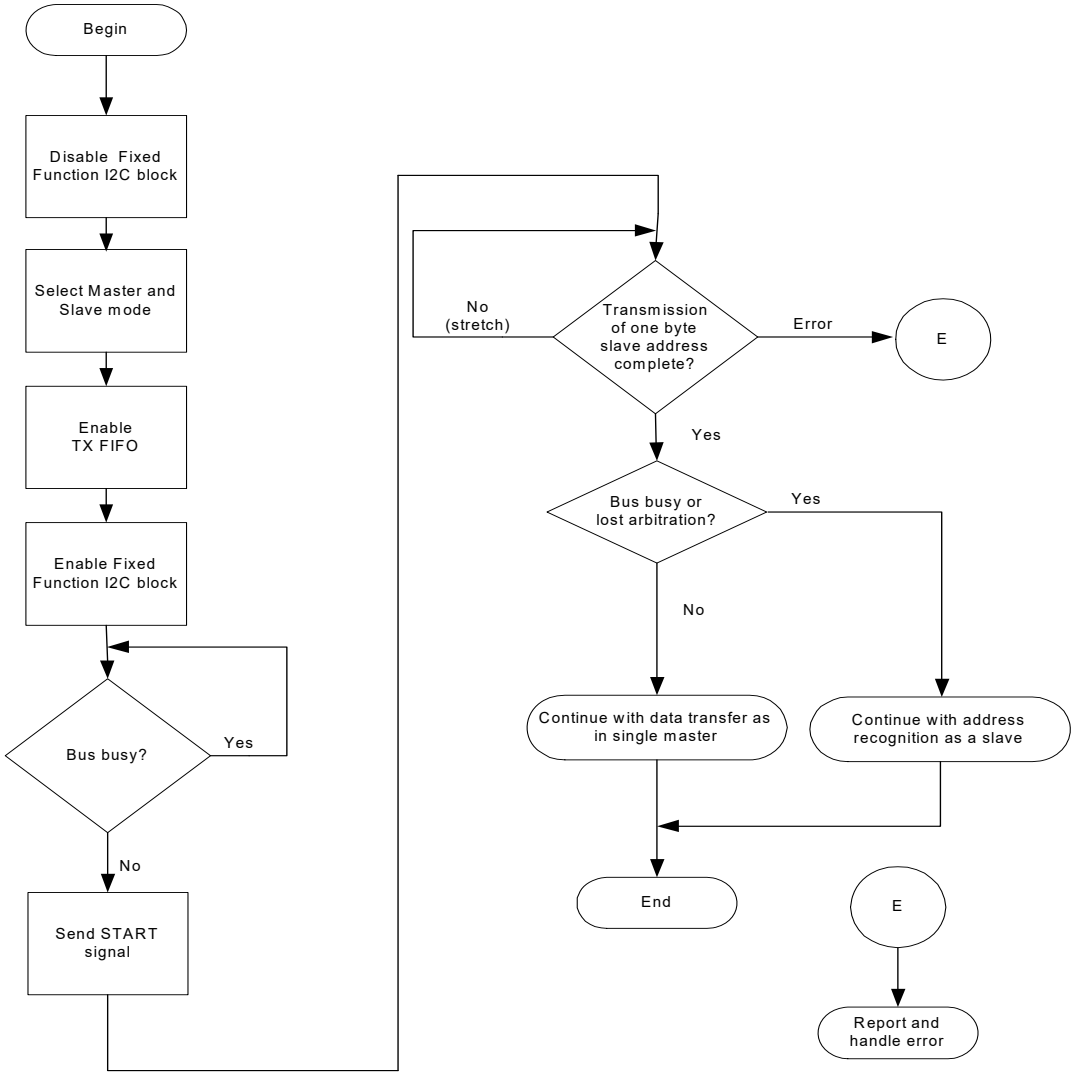
15.4.14.1 Multi-Master - Slave Not Enabled

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. Universal Digital Blocks (UDB)

This chapter shows the design details of the PSoC® 4 universal digital blocks (UDBs). The UDB architecture implements a balanced approach between configuration granularity and efficiency; UDBs have a combination of programmable logic devices (PLDs), structured logic (datapaths), and a flexible routing scheme.

16.1 Features

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

Figure 16-1 shows the components of a single UDB: two PLDs, a datapath, and control, status, clock and reset functions.

Figure 16-1. Single UDB Block Diagram

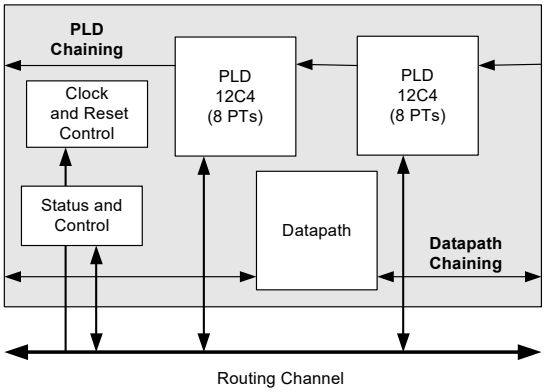
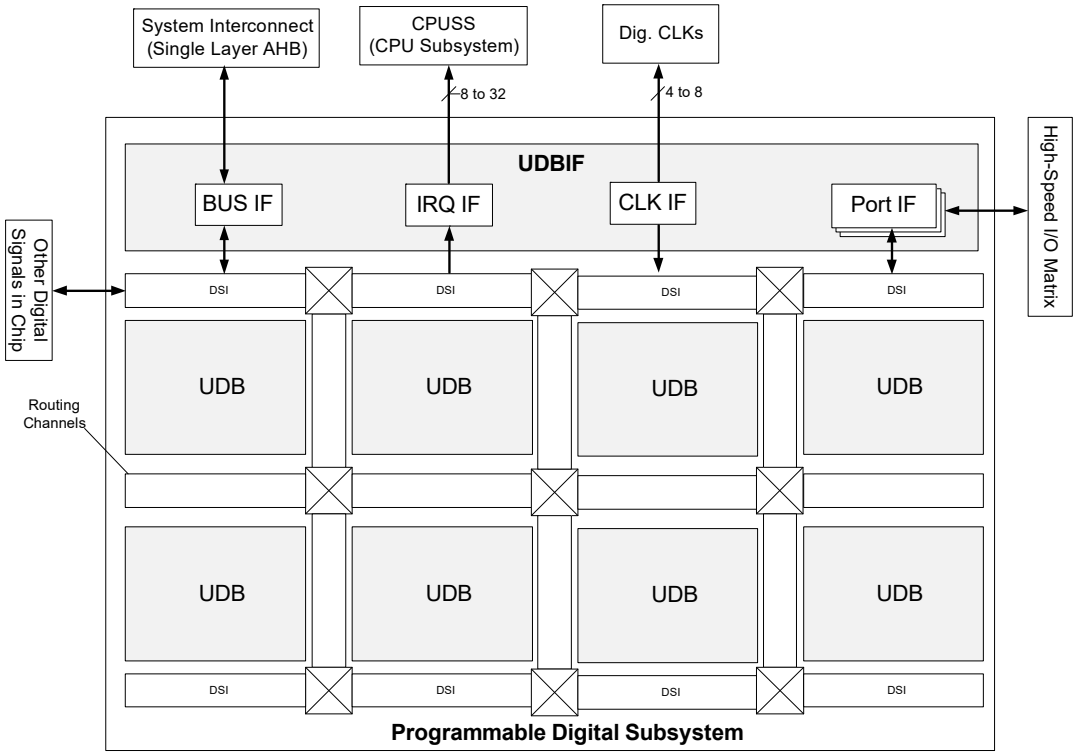


Figure 16-2 shows how the array of four UDBs interfaces with the rest of the PSoC 4.

Figure 16-2. UDBs Array in PSoC 4



16.2 How It Works

The major components of a UDB are:

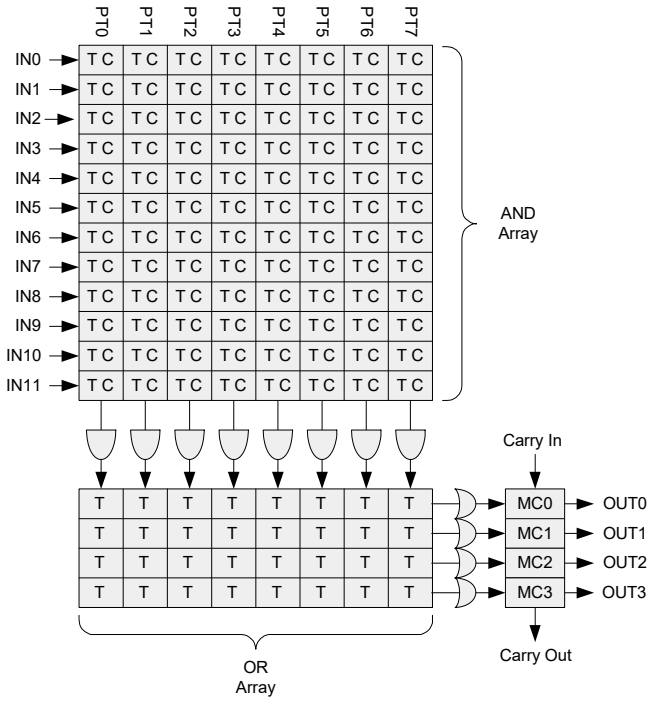
- **PLDs (2)** – These blocks take inputs from the routing channel and form registered or combinational sum-of-products logic to implement state machines, control for datapath operations, conditioning inputs, and driving outputs.
- **Datapath** – This block contains a dynamically programmable ALU, four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

16.2.1 PLDs

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

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

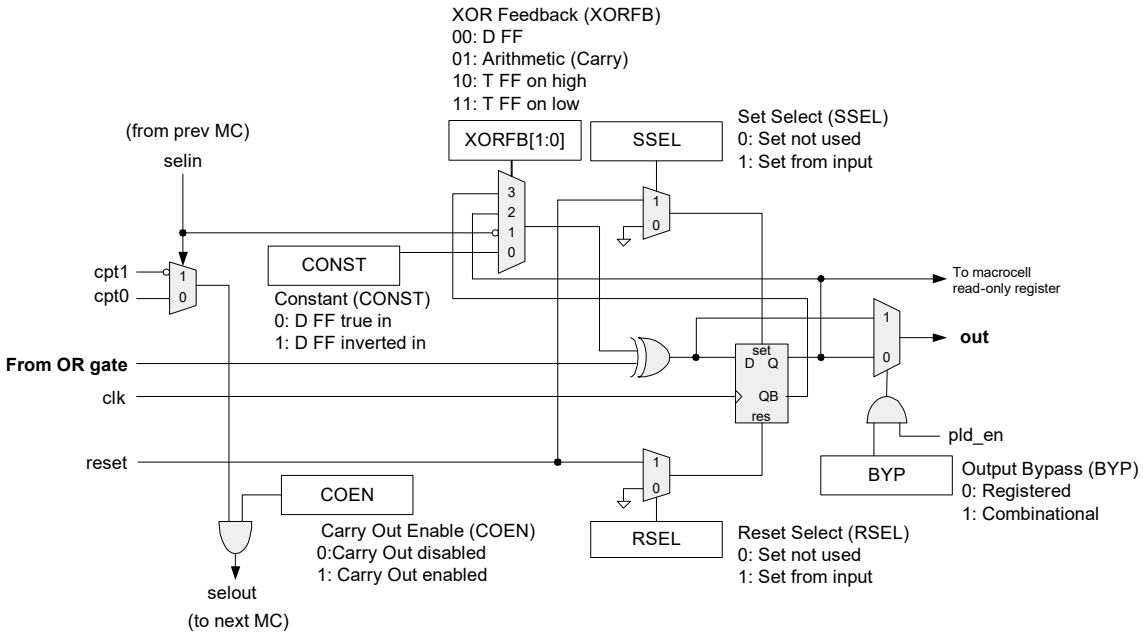
Figure 16-3. PLD 12C4 Structure



16.2.1.1 PLD Macrocells

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

Figure 16-4. PLD Macrocell Architecture



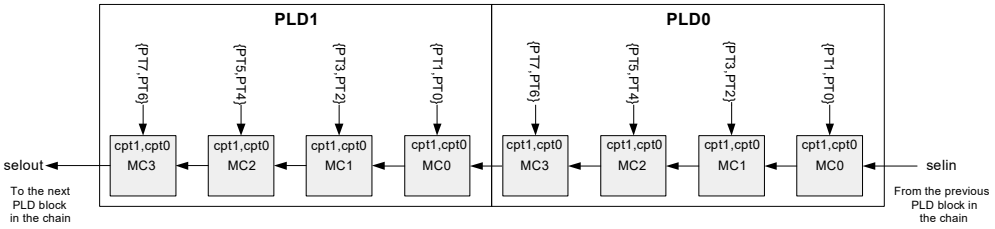
PLD Macrocell Read-Only Registers

The outputs of the eight macrocells in the two PLDs can be accessed by the CPU as an 8-bit read-only register. Macrocells across multiple UDBs can be accessed as 16- or 32-bit read-only registers. See “UDB Addressing” on page 185.

16.2.1.2 PLD Carry Chain

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

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



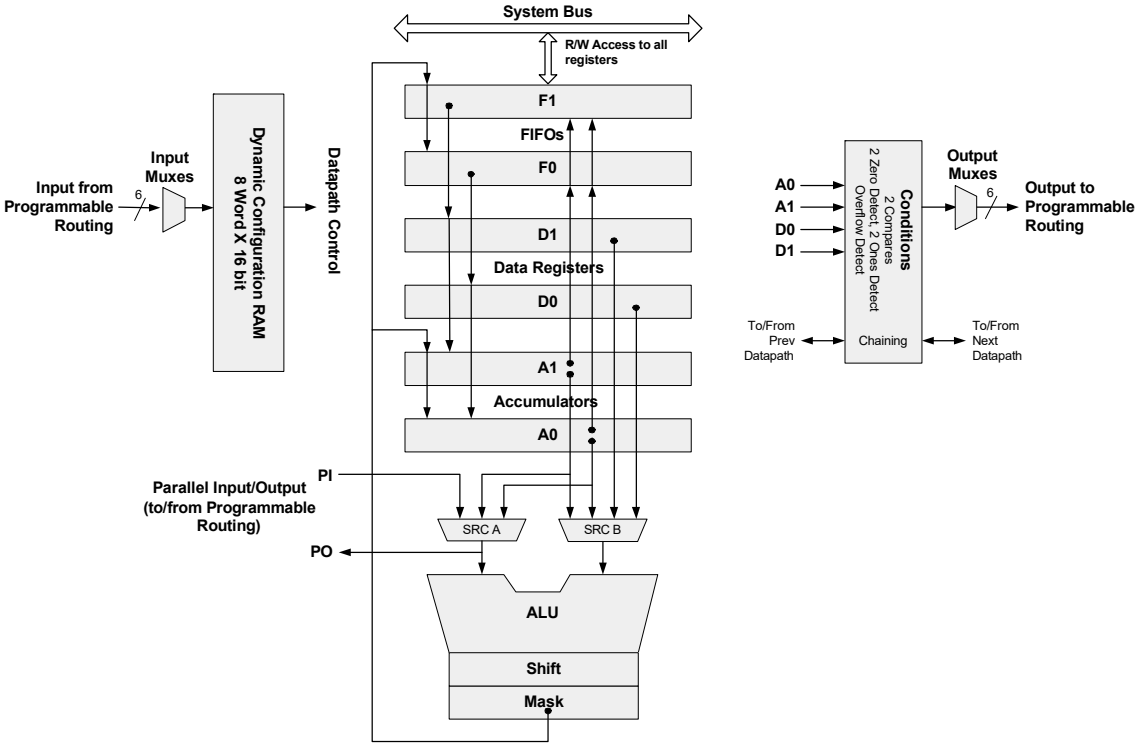
16.2.1.3 PLD Configuration

The PLDs can be configured by accessing a set of 16- or 32-bit registers; see “UDB Addressing” on page 185.

16.2.2 Datapath

The datapath, shown in Figure 16-6, contains an 8-bit single-cycle ALU, with associated compare and condition generation circuits. A datapath may be chained with datapaths in neighboring UDBs to achieve higher precision functions. The datapath includes a small dynamic configuration RAM, which can dynamically select the operation to perform in a given cycle. The datapath is optimized to implement typical embedded functions such as timers, counters, PWMs, PRS, CRC, shifters, and dead band generators. The add and subtract functions allow support for digital delta-sigma operations.

Figure 16-6. Datapath Top Level



16.2.2.1 Overview

The following are key datapath features.

Dynamic Configuration

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

ALU

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

Conditionals

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

Built-in CRC/PRS

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

Variable MSB

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

Input/Output FIFOs

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

Chaining

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

Time Multiplexing

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

Datapath Inputs

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

Datapath Outputs

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

Datapath Working Registers

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

Table 16-1. Datapath Working Registers

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

16.2.2.2 Datapath FIFOs

FIFO Modes and Configurations

Each FIFO has a variety of operation modes and configurations.

Table 16-2. FIFO Modes and Configurations

Mode	Description
Input/Output	In input mode, the CPU writes to the FIFO and the data is read and consumed by the datapath internals. In output mode, the FIFO is written to by the datapath internals and is read and consumed by the CPU.
Single Buffer	The FIFO operates as a single-byte buffer with no status. Data written to the FIFO is immediately available for reading, and can be overwritten at anytime.
Level/Edge	The control to load the FIFO from the datapath internals can be either level or edge triggered.

Table 16-2. FIFO Modes and Configurations *(continued)*

Mode	Description
Normal/Fast	The control to load the FIFO from the datapath source is sampled on the currently selected datapath clock (normal) or the HFCLK (fast). This allows captures to occur at the highest rate in the system (HFCLK), independent of the datapath clock.
Software Capture	When this mode is enabled and the FIFO is in output mode, a read by the CPU of the associated accumulator (A0 for F0, A1 for F1) initiates a synchronous transfer of the accumulator value into the FIFO. The captured value may then be immediately read from the FIFO. If chaining is enabled, the operation follows the chain to the MS block for atomic reads by datapaths of multi-byte values.
Asynch	When the datapath is being clocked asynchronously to the HFCLK, the FIFO status signals can be routed to the rest of the datapath either directly, single sampled to the datapath clock, or double sampled in the case of an asynchronous datapath clock
Independent Clock Polarity	Each FIFO has a control bit to invert polarity of the FIFO clock with respect to the datapath clock.

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

Figure 16-7. FIFO Configurations

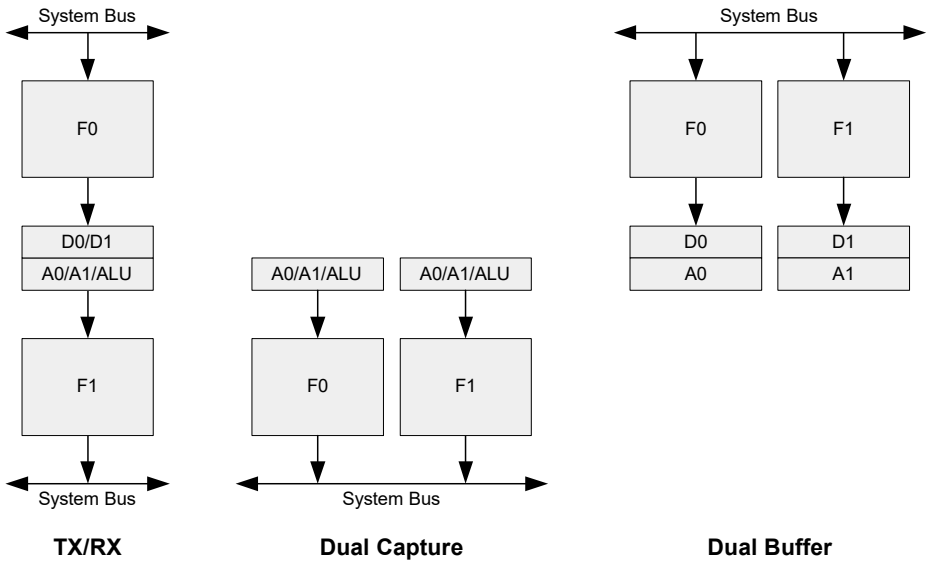
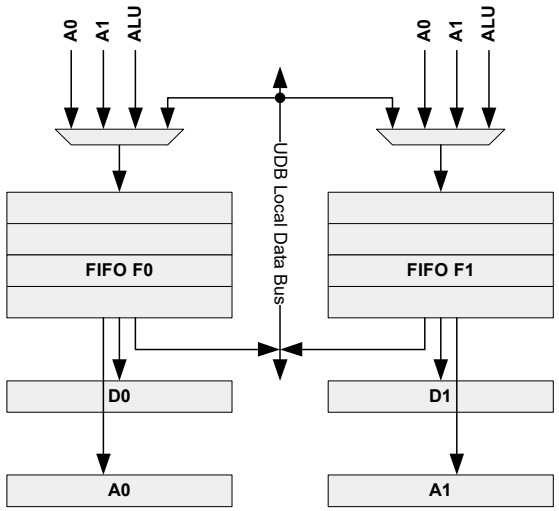


Figure 16-8 shows a detailed view of FIFO sources and sinks.

Figure 16-8. FIFO Sources and Sinks



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

Table 16-3. FIFO Multiplexer Set in UDB CFG15 Register

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

FIFO Status

Each FIFO generates two status signals, “bus” and “block,” which are sent to the UDB routing through the datapath output multiplexer. The “bus” status can be used to assert an interrupt request to read/write the FIFO. The “block” status is primarily intended to provide the FIFO state to the UDB internals. The meanings of the status bits depend on the configured direction (Fx_INSEL[1:0] in the UDB CFG15 register) and the FIFO level bits. The FIFO level bits (Fx_LVL) are set in the Auxiliary Control Working register (ACTL) in working register space. Table 16-4 shows the options.

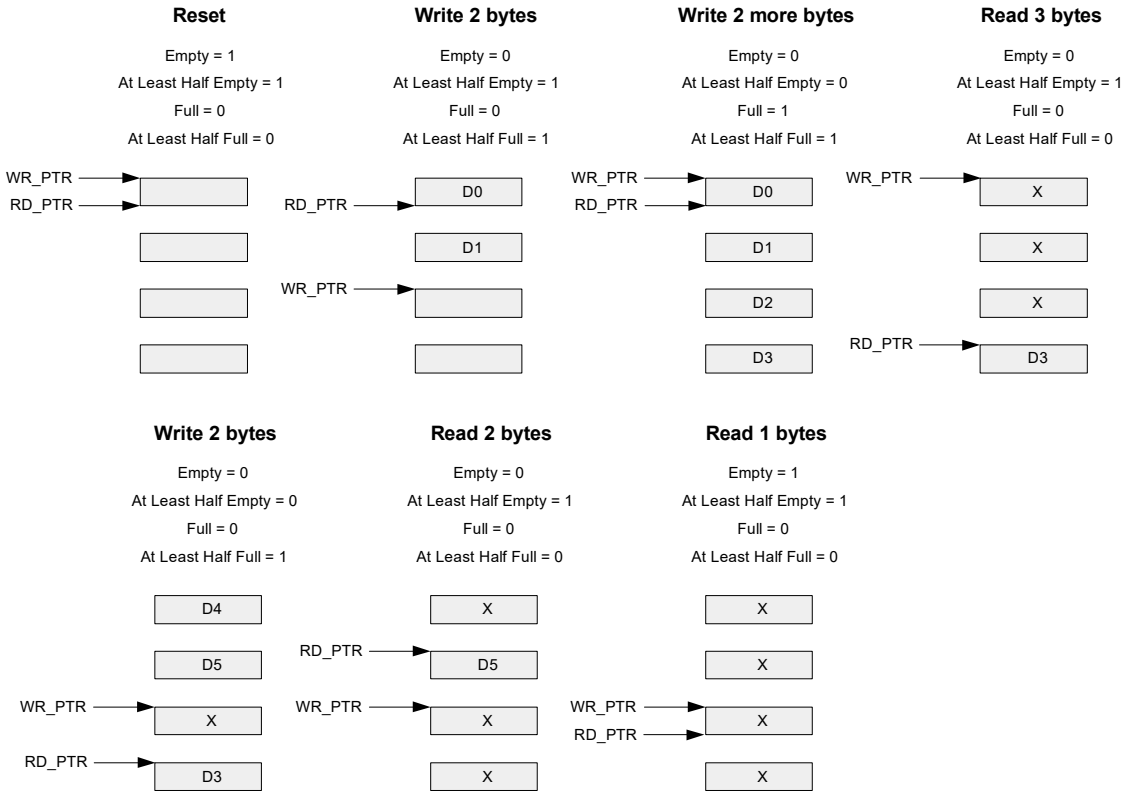
Table 16-4. FIFO Status Options

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

FIFO Operation

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

Figure 16-9. Detailed FIFO Operation Sinks

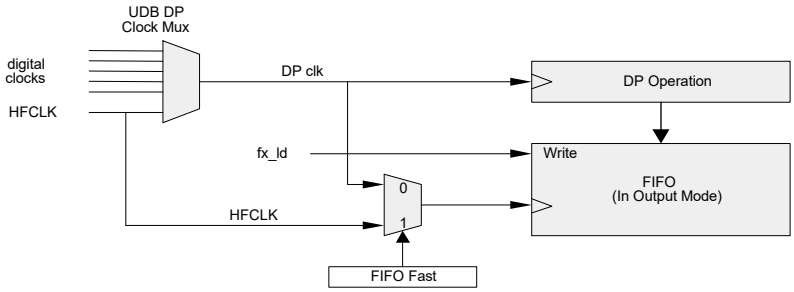


FIFO Fast Mode (FIFO FAST)

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

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

Figure 16-10. FIFO Fast Configuration Sinks



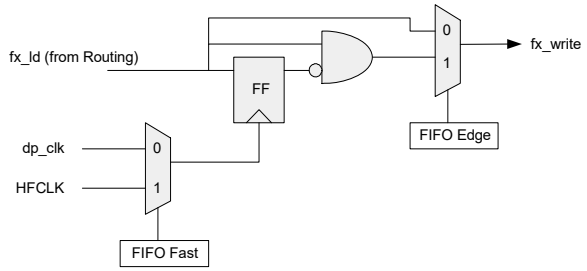
FIFO Level/Edge Write Mode

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

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

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

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



FIFO Software Capture Mode

A common and important requirement is to allow the CPU the ability to reliably read the contents of an accumulator during normal operation. This is done with software capture and is enabled by setting the FIFO Cap configuration bit (FIFO_CAP bit in the UDB CFG16 register). This bit applies to both FIFOs in a UDB, but is only operational when a FIFO is in output mode. When using software capture, F0 should be set to load from A0 and F1 from A1.

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

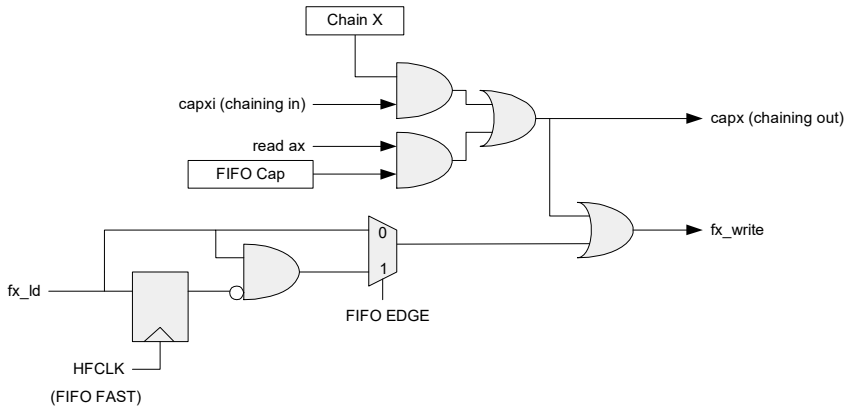
The fx_ld signal, which generates a FIFO load, is ORed with the software capture signal; the results can be unpredictable when both hardware and software capture are used at the same time. As a general rule, these functions should be mutually exclusive; however, hardware and software capture can be used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

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

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

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

Figure 16-12. Software Capture Configuration



FIFO Control Bits

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

The FIFO0 CLR and FIFO1 CLR bits are used to reset or flush the FIFO. When a '1' is written to one of these bits, the associated FIFO is reset. The bit must be written back to '0' for FIFO operation to continue. If the bit is left asserted, the given FIFO is disabled and operates as a one byte buffer without status. Data can be written to the FIFO; the data is immediately available for reading and can be overwritten at anytime. Data direction using the Fx INSEL[1:0] (UDB CFG15 register) configuration bits is still valid.

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

Table 16-5. FIFO Level Control Bits in UDB ACTL Register

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

FIFO Asynchronous Operation

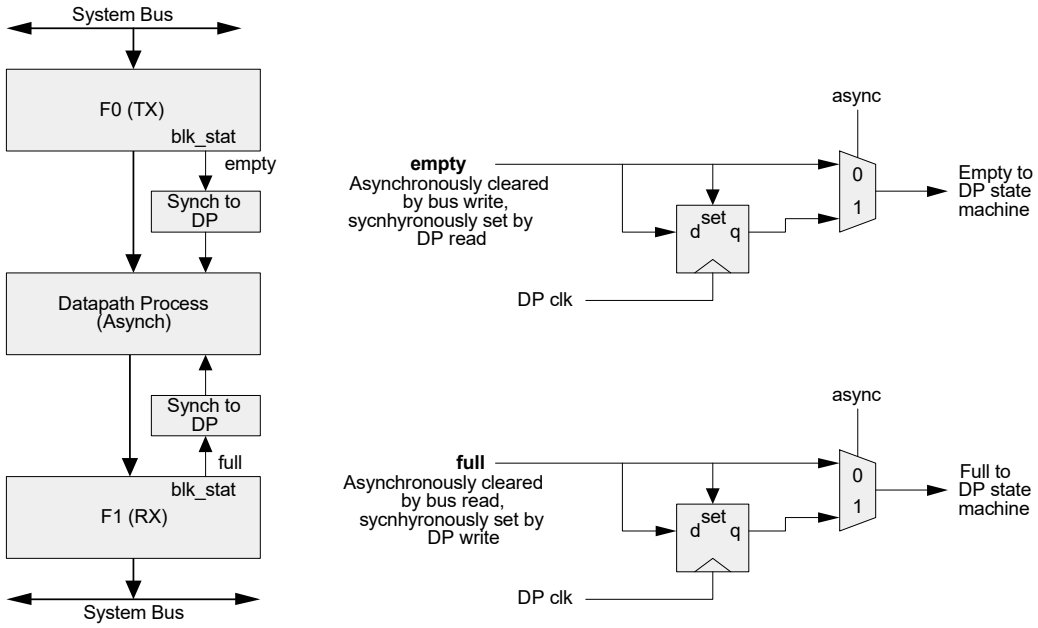
Figure 16-13 illustrates the concept of asynchronous FIFO operation. As an example, assume F0 is set for input mode and F1 is set for output mode, which is a typical configuration for TX and RX registers.

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

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

A single FIFO ASYNCH bit of the UDB CFG16 register is used to enable this synchronization method; when set it applies to both FIFOs. It is only applied to the block status, as it is assumed that bus status is naturally synchronized by the interrupt process.

Figure 16-13. FIFO Asynchronous Operation



FIFO Overflow Operation

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

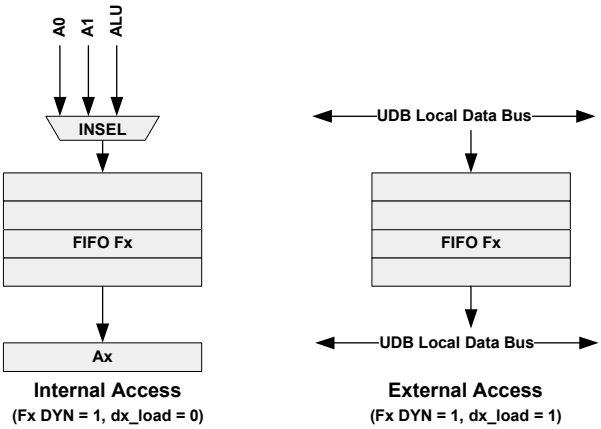
FIFO Clock Inversion Option

Each FIFO has a control bit called Fx CK INV in the UDB CFG16 register that controls the polarity of the FIFO clock, with respect to the polarity of the DP clock. By default, the FIFO operates at the same polarity as the DP clock. When this bit is set, the FIFO operates at the opposite polarity as the DP clock. This provides support for “both clock edge” communication protocols, such as SPI.

FIFO Dynamic Control

Normally, the FIFOs are configured statically in either input or output mode. As an alternative, each FIFO can be configured into a mode where the direction is controlled dynamically, that is, by routed signals. One configuration bit per FIFO (Fx DYN bit in the UDB CFG17 register) enables the mode. Figure 16-14 shows the configurations available in dynamic FIFO mode.

Figure 16-14. FIFO Dynamic Mode



In internal access mode, the datapath can read and write the FIFO. In this configuration, the Fx INSEL bits must be configured to select the source for the FIFO writes. Fx INSEL = 00 (CPU bus source) is invalid in this mode; they can only be 01, 10, or 11 (A0, A1, or ALU). Note that the only read access is to the associated accumulator; the data register destination is not available in this mode.

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

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

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

Table 16-6. FIFO Status

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

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

16.2.2.3 FIFO Status

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

16.2.2.4 Datapath ALU

The ALU core consists of three independent 8-bit programmable functions, which include an arithmetic/logic unit, a shifter unit, and a mask unit. See the UDB datapath architecture block diagram (Figure 16-6) for more details.

Arithmetic and Logic Operation

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

Table 16-7. ALU Functions in UDB DCFG Register

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

srca = ‘A’ input source to the ALU, srcb = ‘B’ input source to the ALU. See Figure 16-6.

Carry In

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

Table 16-8. Carry In Functions

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

In addition to this default arithmetic mode for carry operation, there are three additional carry options. The CI SELA and CI SELB configuration bits in the CFG13 register determine the carry in for a given cycle. Dynamic configuration RAM selects either the A or B configuration on a cycle-by-cycle basis. The options are defined in Table 16-9.

Table 16-9. Additional Carry In Functions in UDB CFG13

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

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

Table 16-10. Routed Carry In Functions

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

Carry Out

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

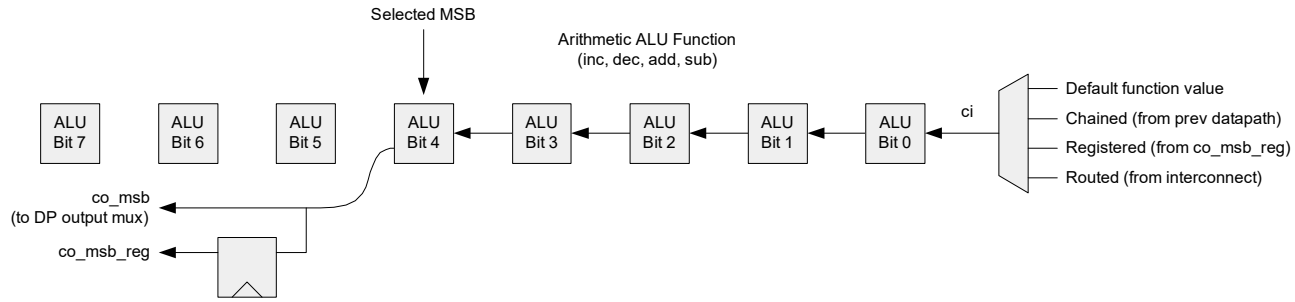
Table 16-11. Carry Out Functions

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

Carry Structure

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

Figure 16-15. Carry Operation



Shift Operation

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

Table 16-12. Shift Operation Functions in UDB DCFG Register

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

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

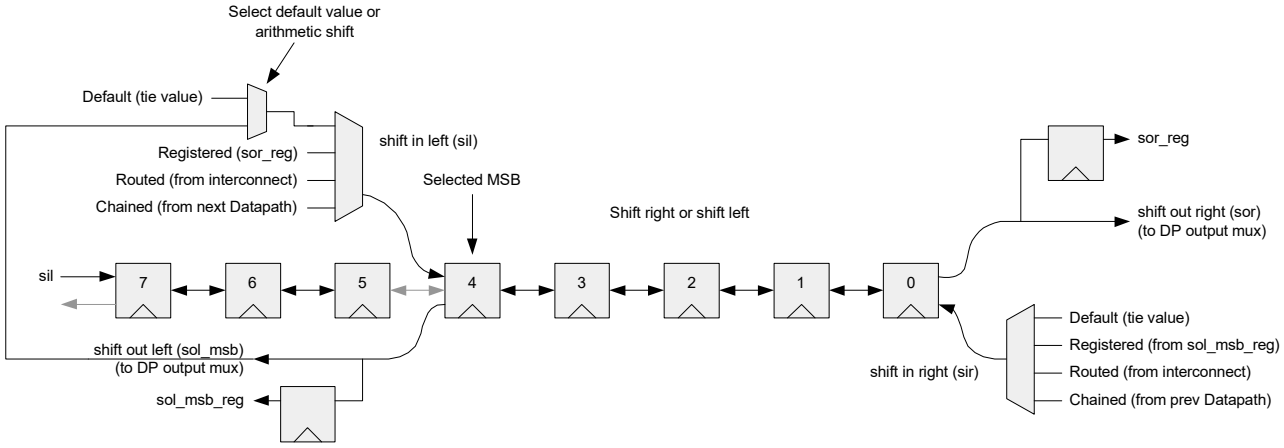
The SI SELA and SI SELB configuration bits determine the shift in data for a given operation. Dynamic configuration RAM selects the A or B configuration on a cycle-by-cycle basis. Shift in data is only valid for left and right shift; it is not used for pass and nibble swap. Table 16-13 shows the selections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

Table 16-13. Shift In Functions in UDB CFG15 Register

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

The shift out left data comes from the currently defined MSB position (MSB_EN and MSB_SEL bits in the CFG14 register), and the data that is shifted in from the left (in a shift right operation) goes into the currently defined MSB position. Both shift out data (left or right) are registered and can be used in a subsequent cycle. This feature can be used to implement a higher precision shift in multiple cycles.

Figure 16-16. Shift Operation



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

ALU Masking Operation

An 8-bit mask register (AMASK) in the UDB static configuration register space (CFG9) defines the masking operation. In this operation, the output of the ALU is masked (ANDed) with the value in the mask register. A typical use for the ALU mask function is to implement free-running timers and counters in power of two resolutions.

16.2.2.5 Datapath Inputs and Multiplexing

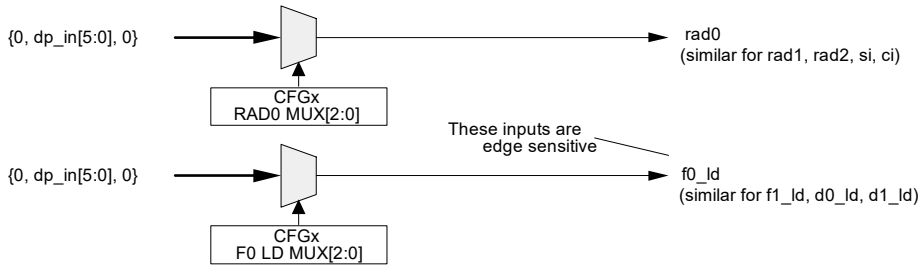
The datapath has a total of nine inputs, as shown in Table 16-14, including six inputs from the channel routing. These consist of the configuration RAM address, FIFO and data register load control signals, and the data inputs shift in and carry in.

Table 16-14. Datapath Inputs

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

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

Figure 16-17. Datapath Input Selection



16.2.2.6 CRC/PRS Support

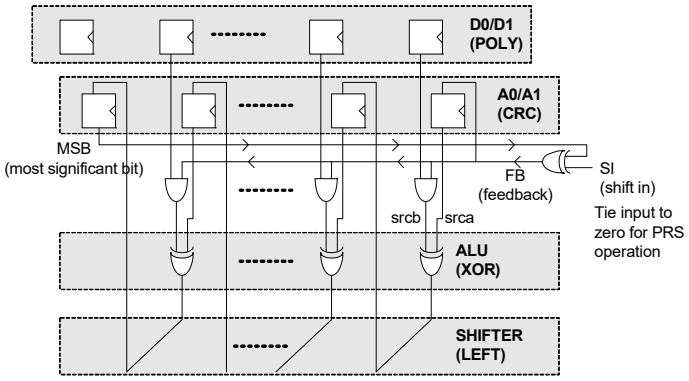
The datapath can support cyclic redundancy checking (CRC) and pseudo random sequence (PRS) generation. Chaining signals are routed between datapath blocks to support CRC/PRS bit lengths of longer than eight bits.

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

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

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

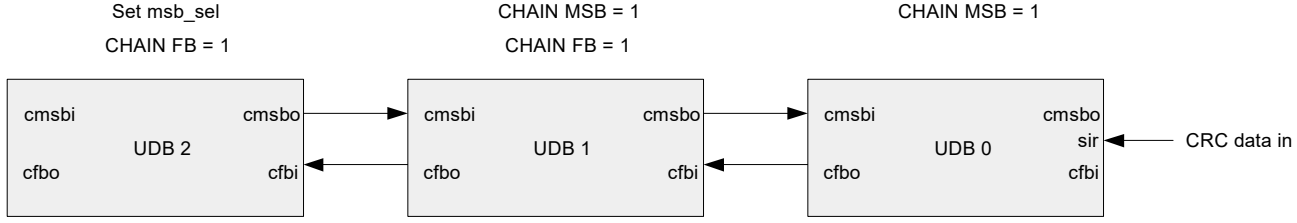
Figure 16-18. CRC Functional Structure



CRC/PRS Chaining

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

Figure 16-19. CRC/PRS Chaining Configuration



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

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

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

- If a given block is the most significant block, the MSB bit (according to the polynomial selected) is configured using the MSB_SEL configuration bits in the UDB CFG14 register.
- If a given block is not the most significant block, the CHAIN CMSB configuration bit in the UDB CFG14 register must be set and the MSB signal is chained from the next block in the chain.

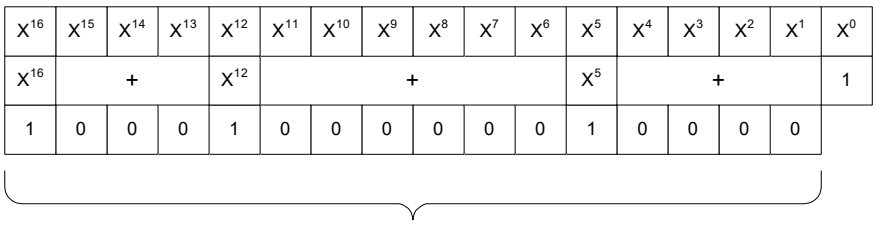
CRC/PRS Polynomial Specification

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

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

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

Figure 16-20. CCITT CRC16 Polynomial Format



Example CRC/PRS Configuration

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

1. Select a suitable polynomial and write it into D0.
2. Select a suitable seed value (for example, all zeros for CRC, all ones for PRS) and write it into A0.
3. Configure chaining if necessary.
4. Select the MSB position as defined in the polynomial from the MSB_SEL static configuration register bits and set the MSB_EN register bit in the UDB CFG14 register.
5. Configure the dynamic configuration RAM word fields:
   a. Select D0 as the ALU "SRCB" (ALU B Input Source)
   b. Select A0 as the ALU "SRCA" (ALU A Input Source)
   c. Select "XOR" for the ALU function
   d. Select "SHIFT LEFT" for the SHIFT function
   e. Select "CFB_EN" to enable the support for CRC/PRS
   f. Select ALU as the A0 write source

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

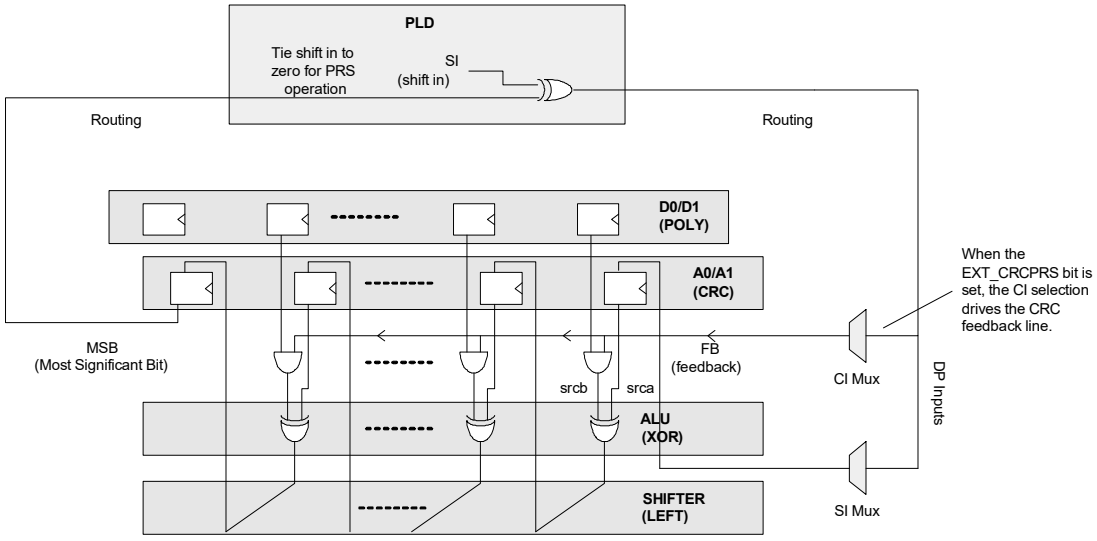
Clocking the UDB with this configuration generates the required CRC or outputs the MSB, which may be output to the routing for the PRS sequence.

External CRC/PRS Mode

A static configuration bit may be set (EXT CRCPRS in the UDB CFG16 register) to enable support for external computation of a CRC or PRS. As shown in Figure 16-21, computation of the CRC feedback is done in a PLD block. When the bit is set, the CRC feedback signal is driven directly from the CI (Carry In) datapath input selection mux, bypassing the internal computation. The figure shows a simple configuration that supports up to an 8-bit CRC or PRS. Normally the built-in circuitry is used, but this feature gives the capability for more elaborate configurations, such as up to a 16-bit CRC/PRS function in one UDB using time division multiplexing.

In this mode, the dynamic configuration RAM bit CFB_EN in the UDB DCFG0 register still controls whether the CRC feedback signal is ANDed with the SRCB ALU input. Therefore, as with the built-in CRC/PRS operation, the function can be interleaved with other functions if required.

Figure 16-21. External CRC/PRS Mode



16.2.2.7 Datapath Outputs and Multiplexing

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

Table 16-15. Datapath Condition Generation

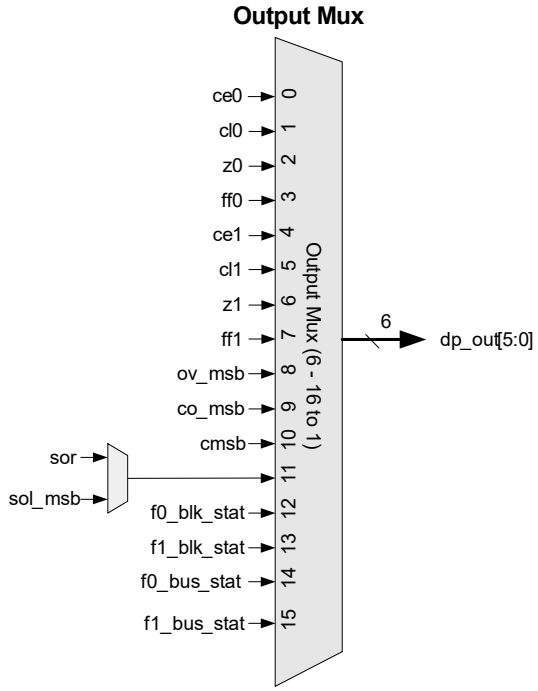
Name	Condition	Chain	Description
ce0	Compare Equal	Y	A0 == D0
cl0	Compare Less Than	Y	A0 < D0
z0	Zero Detect	Y	A0 == 00h
ff0	Ones Detect	Y	A0 == FFh
ce1	Compare Equal	Y	A1 or A0 == D1 or A0 (dynamic selection)
cl1	Compare Less Than	Y	A1 or A0 < D1 or A0 (dynamic selection)
z1	Zero Detect	Y	A1 == 00h
ff1	Ones Detect	Y	A1 == FFh
ov_msb	Overflow	N	Carry(msb) ^ Carry(msb–1)
co_msb	Carry Out	Y	Carry out of MSB defined bit

Table 16-15. Datapath Condition Generation

Name	Condition	Chain	Description
cmsb	CRC MSB	Y	MSB of CRC/PRS function
So	Shift Out	Y	Selection of shift output
f0_blk_stat	FIFO0 Block Status	N	Definition depends on FIFO configuration
f1_blk_stat	FIFO1 Block Status	N	Definition depends on FIFO configuration
f0_bus_stat	FIFO0 Bus Status	N	Definition depends on FIFO configuration
f1_bus_stat	FIFO1 Bus Status	N	Definition depends on FIFO configuration

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

Figure 16-22. Output Mux Connections



Compares

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

Comparator 1 inputs are dynamically configurable. As shown in Table 16-16, there are four options for Comparator 1, which applies to both the "less than" and the "equal" conditions. The CMP SELA and CMP SELB configuration bits in the UDB CFG12 register determine the possible compare configurations. A dynamic configuration RAM bit CMP SEL in the UDB DCFG0 register selects one of the A or B configurations on a cycle-by-cycle basis.

Table 16-16. Compare Configuration

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

Compare 0 and Compare 1 are independently chainable to the conditions generated in the previous datapath (in addressing order). The decision to chain compares is statically specified by CHAIN0 and CHAIN1 bits of the UDB CFG14 registers. Figure 16-23 illustrates compare equal chaining, which is just an ANDing of the compare equal in this block with the chained input from the previous block.

Figure 16-23. Compare Equal Chaining

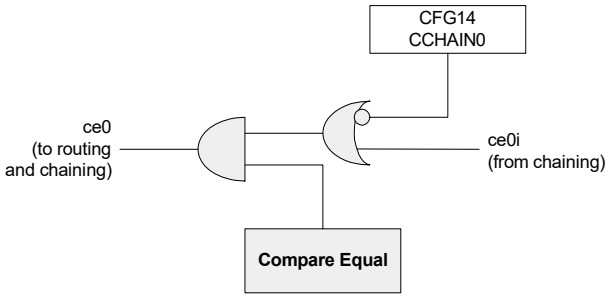
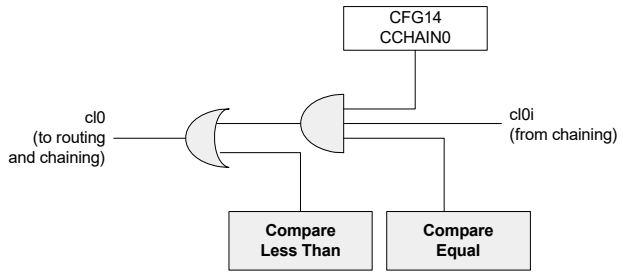


Figure 16-24 illustrates compare less than chaining. In this case, the “less than” is formed by the compare less than output in this block, which is unconditional. This is ORed with the condition where this block is equal, and the chained input from the previous block is asserted as less than.

Figure 16-24. Compare Less Than Chaining



All Zeros and All Ones Detect

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

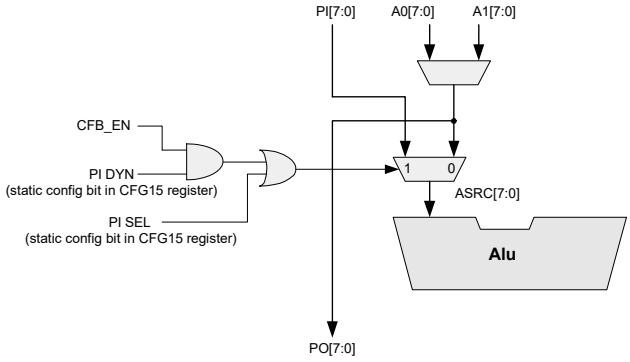
Overflow

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

16.2.2.8 Datapath Parallel Inputs and Outputs

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

Figure 16-25. Datapath Parallel In/Out

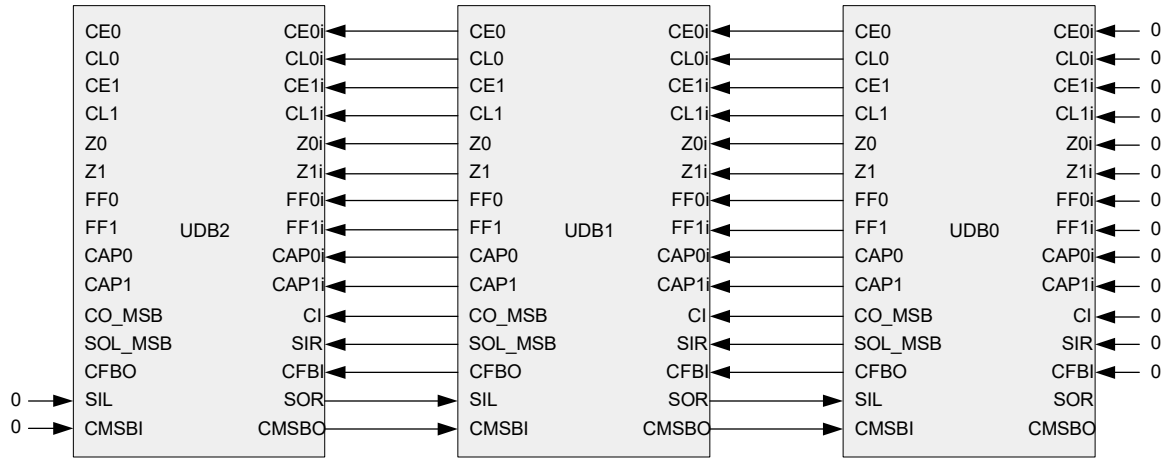


Parallel In needs to be selected for input to the ALU. The two options available are static operation or dynamic operation. For static operation, the PI SEL bit of the UDB CFG15 register forces the ALU asrc to be PI. The PI DYN bit of the UDB CFG15 register is used to enable the PI dynamic operation. When it is enabled, and assuming the PI SEL is 0, the PI multiplexer may then be controlled by the CFB_EN (UDB DCFG0 register) dynamic control bit. The primary function of CFB_EN is to enable PRS/CRC functionality.

16.2.2.9 Datapath Chaining

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

Figure 16-26. Datapath Chaining Flow

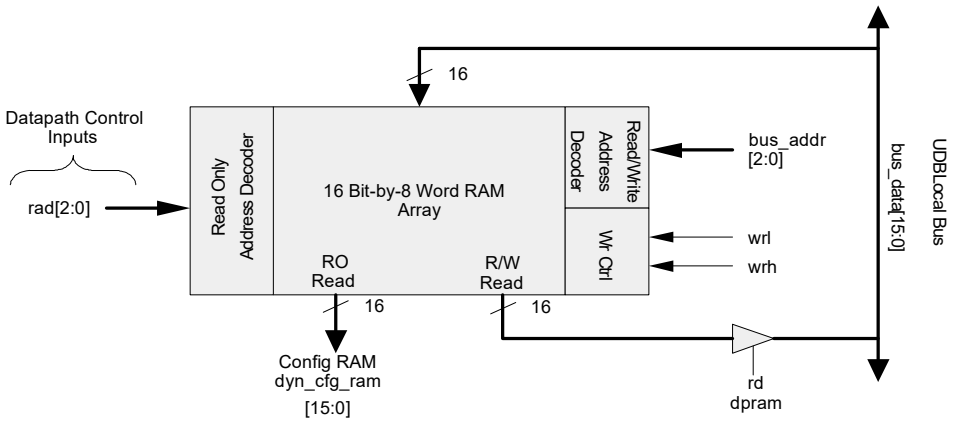


16.2.2.10 Dynamic Configuration RAM

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

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

Figure 16-27. Configuration RAM I/O



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

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

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

Table 16-17. Dynamic Configuration Quick Reference

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

a. For CI, SI, and CMP, the RAM fields select between two predefined static settings; see Table 16-9, Table 16-13, and Table 16-16 respectively.

16.2.3 Status and Control Module

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

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

Figure 16-28. Status and Control Registers

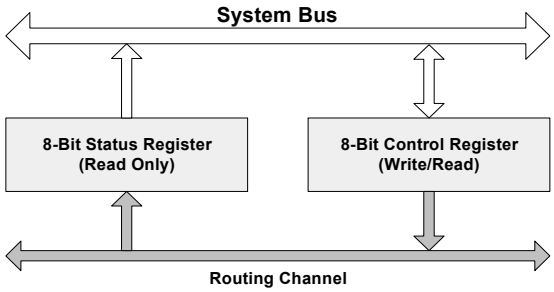
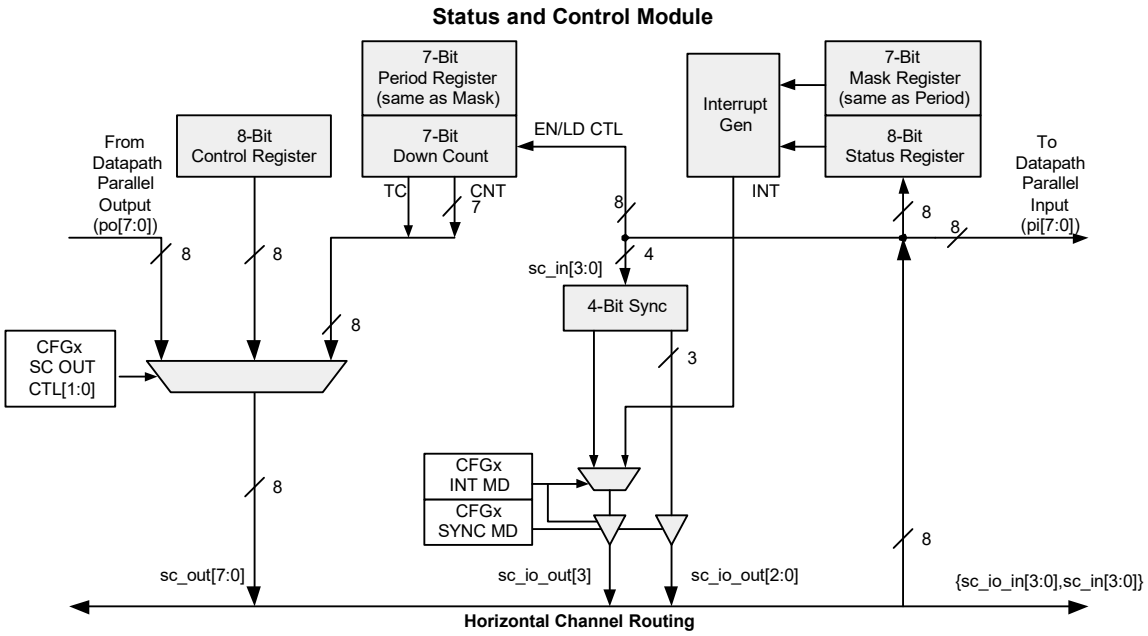


Figure 16-29. Status and Control Module



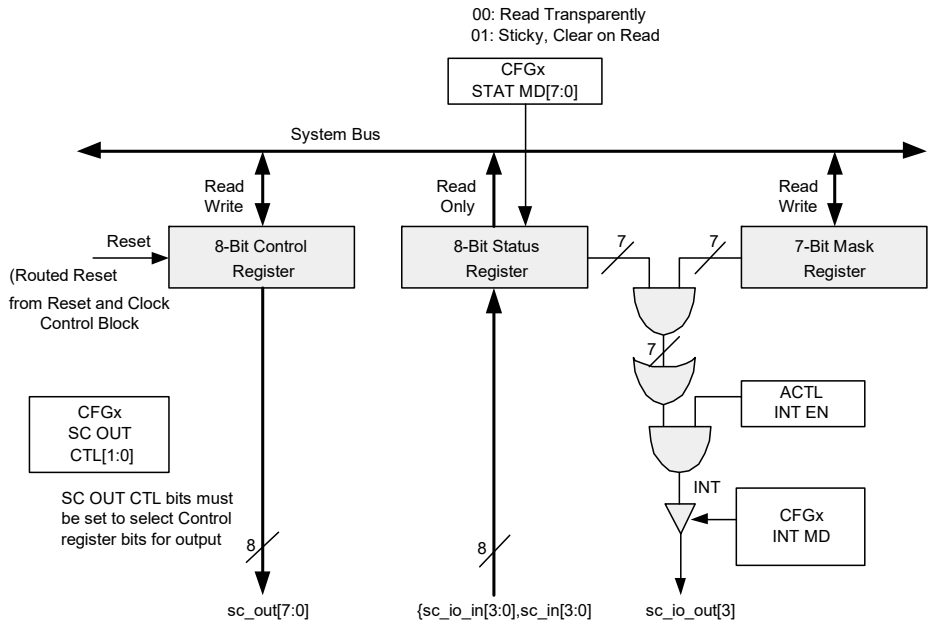
Modes of operation include:

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

16.2.3.1 Status and Control Mode

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

Figure 16-30. Status and Control Operation



Status Register Operation

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

Table 16-18. Status Register Mode Selection in UDB CFG20 Register

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

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

Transparent Status Read

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

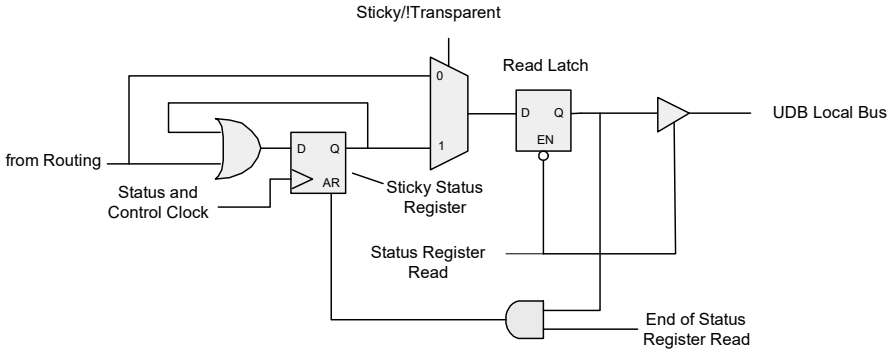
Sticky Status, with Clear on Read

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

Status Latching During Read

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

Figure 16-31. Status Read Logic



Interrupt Generation

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

16.2.3.2 Control Register Operation

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

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

Control Register Operating Modes

Three modes are available that may be configured on a bit-by-bit basis. The configuration is controlled by the concatenation of the bits of the two 8-bit registers CTL_MD1[7:0] and CTL_MD0[7:0] of the UDB CFG18 and CFG19 registers. For example, {CTL_MD1[0],CTL_MD0[0]} controls the mode for Control Register bit 0, as shown in Table 16-19.

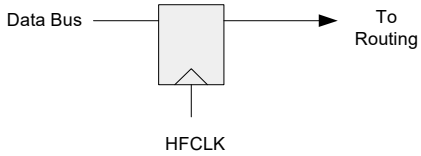
Table 16-19. Mode for Control Register Bit 0 in the UDB CFG18 and CFG19 Registers

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

Control Register Direct Mode

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

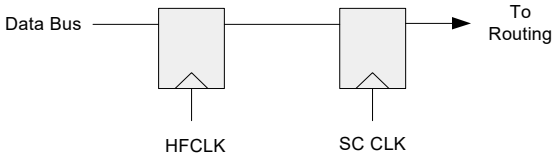
Figure 16-32. Control Register Direct Mode



Control Register Sync Mode

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

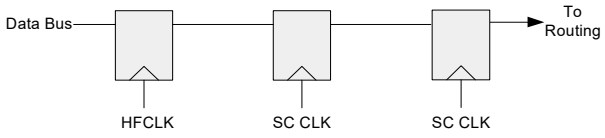
Figure 16-33. Control Register Sync Mode



Control Register Double Sync Mode

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

Figure 16-34. Control Register Double Sync Mode



Control Register Pulse Mode

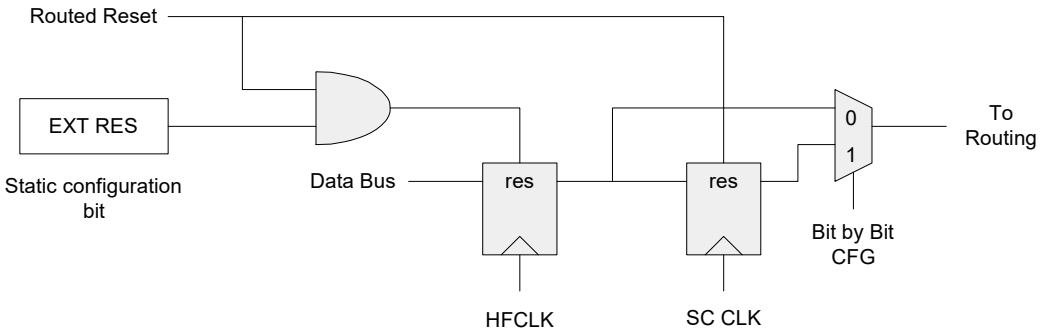
Pulse mode is similar to Sync mode in that the control bit is re-sampled by the SC clock; the pulse starts on the first SC clock cycle following the bus write cycle. The output of the control bit is asserted for one full SC clock cycle. At the end of this clock cycle, the control bit is automatically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

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

Control Register Reset

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

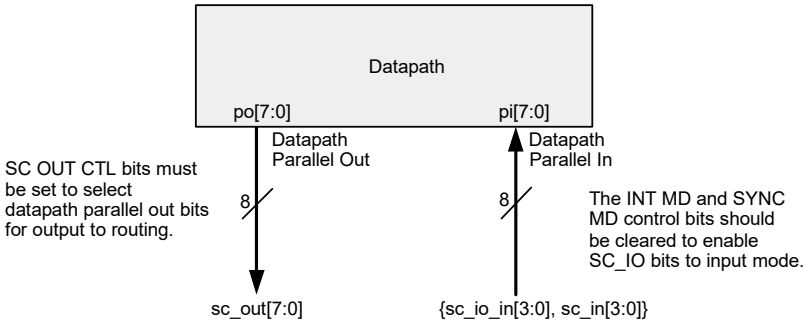
Figure 16-35. Control Register Reset



16.2.3.3 Parallel Input/Output Mode

In this mode, as Figure 16-36 shows, the status and control routing is connected to the datapath parallel in and parallel out signals. To enable this mode, the SC OUT configuration bits in the UDB CFG22 registers are set to select datapath parallel out. The parallel input connection is always available, but these routing connections are shared with the status register inputs, counter control inputs, and the interrupt output.

Figure 16-36. Parallel Input/Output Mode



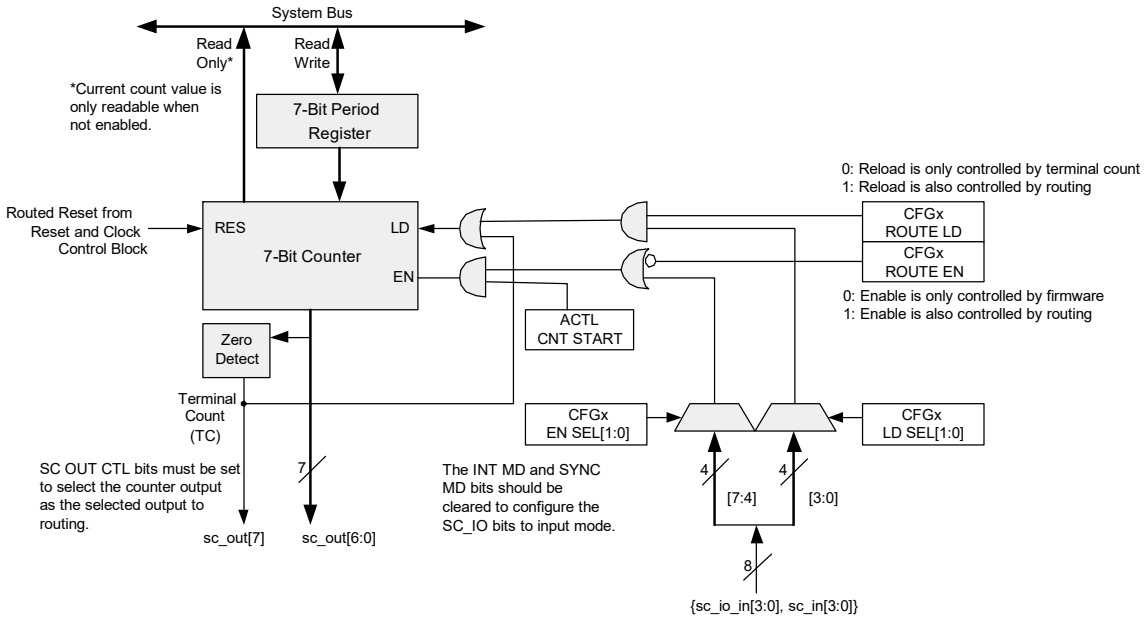
16.2.3.4 Counter Mode

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

- A 7-bit read/write period register.
- A 7-bit read/write count register. It can be accessed only when the counter is disabled.
- Automatic reload of the period to the count register on terminal count (0).
- A firmware control bit in the Auxiliary Control Working register (ACTL0) called CNT START, to start and stop the counter. (This is an overriding enable and must be set for optional routed enable to be operational.)
- Selectable bits from the routing for optional dynamic control of the counter enable and load functions:
  - EN, routed enable to start or stop counting.
  - LD, routed load signal to force the reload of period. When this signal is asserted, it overrides a pending terminal count. It is level sensitive and continues to load the period while asserted.
- The 7-bit count may be driven to the routing fabric as sc_out[6:0].
- The terminal count may be driven to the routing fabric as sc_out[7].
- In default mode, the terminal count is registered. In alternate mode the terminal count is combinational.
- In default mode, the routed enable, if used, must be asserted for routed load to operate. In alternate mode the routed enable and routed load signals operate independently.

To enable the counter mode, the SC_OUT_CTL[1:0] bits of the UDB CFG22 register must be set to counter output. In this mode the normal operation of the control register is not available. The status register can still be used for read operations, but should not be used to generate an interrupt because the mask register is reused as the counter period register. The Period register is implemented as a retention register and maintains its state across sleep intervals. For a period of N clocks, the period value of N–1 should be loaded. N = 1 (period of 0) is not supported as a clock divide value, and results in the terminal count output of a constant 1.The use of SYNC mode depends on whether the dynamic control inputs (LD/EN) are used. If they are not used, SYNC mode is unaffected. If they are used, SYNC mode is unavailable.

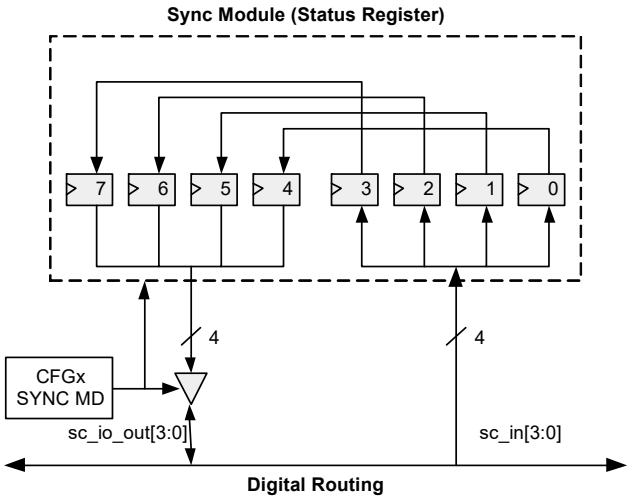
Figure 16-37. Counter Mode



16.2.3.5 Sync Mode

As shown in Figure 16-38, the status register can operate as a 4-bit double synchronizer, clocked by the current SC_CLK, when the SYNC MD bit in the UDB CFG22 register is set. This mode may be used to implement local synchronization of asynchronous signals, such as GPIO inputs. When enabled, the signals to be synchronized are selected from SC_IN[3:0], the outputs are driven to the SC_IO_OUT[3:0] pins, and SYNC MD automatically puts the SC_IO pins into output mode. When in this mode, the normal operation of the status register is not available, and the status sticky bit mode is forced off, regardless of the control settings for this mode. The control register is not affected by the mode. The counter can still be used with limitations. No dynamic inputs (LD/EN) to the counter can be enabled in this mode.

Figure 16-38. Sync Mode



16.2.3.6 Status and Control Clocking

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

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

The clock for this is allocated in the reset and clock control module. See “Reset and Clock Control Module” on page 177.

16.2.3.7 Auxiliary Control Register

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

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

FIFO0 Clear, FIFO1 Clear

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

FIFO0 Level, FIFO1 Level

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

Table 16-20. FIFO Level Control Bits

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

Interrupt Enable

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

Count Start

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

16.2.3.8 Status and Control Register Summary

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

Table 16-21. Status, Control Register Function Summary

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

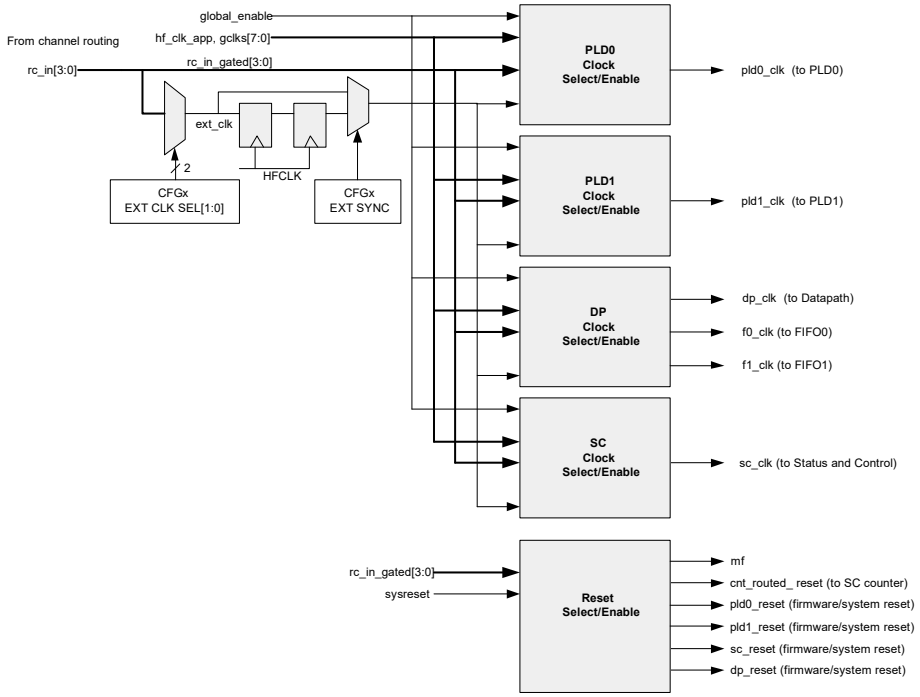
a. In counter mode, the mask register is operating as a period register and cannot function as a mask register. Therefore, interrupt output is not available when counter mode is enabled.
b. In SYNC mode, the status register function is not available, and therefore, the mask register is unusable. However,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

16.2.4 Reset and Clock Control Module

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

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

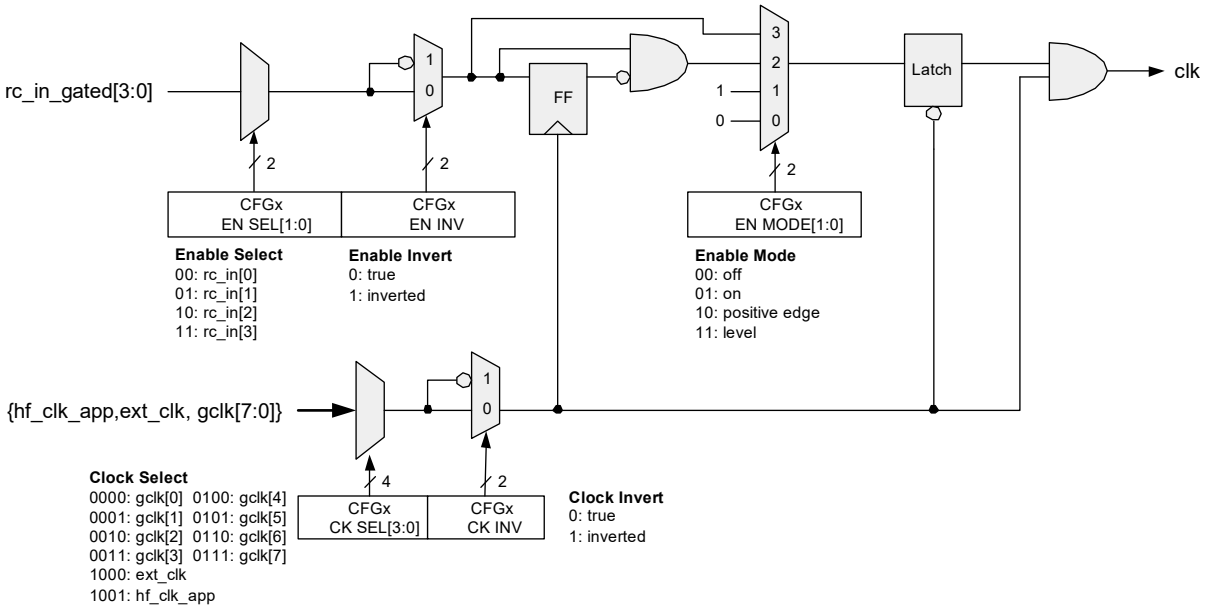
Figure 16-39. Reset and Clock Control



16.2.4.1 Clock Control

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

Figure 16-40. Clock Select/Enable Control



Clock Selection

Four UDB peripheral clocks (see Clocking System chapter on page 72), gclk[0] to gclk[3], are routed to all UDBs; the remaining four clock configurations, gclk[4] to gclk[7], are not supported in the PSoC 4 family of devices. Any of these clocks may be selected. UDB peripheral clocks are the output of user-selectable clock dividers. Another selection is HFCLK, which is the highest frequency in the system. Called “hf_clk_app,” this signal is routed separately from the system HFCLK. In addition, an external routing signal can be selected as a clock input to support direct-clocked functions such as SPI. Because application functions are mapped to arbitrary boundaries across UDBs, individual clock selection for each UDB subcomponent block supports a fine granularity of programming.

Clock Inversion

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

Clock Enable Selection

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

Clock Enable Inversion

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

Clock Enable Mode

By default, the clock enable is OFF. After configuring the target block operation, software can set the mode to one of the following using the EN MODE[1:0] bits of the UDB CFG24 register shown in Figure 16-40.

Table 16-22. Clock Enable Mode in UDB CFG24 Register

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

Clock Enable Usage

The two general usage scenarios for the clock enable are:

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

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

Special FIFO Clocking

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

- Each FIFO clock can be inverted with respect to the selected datapath clock polarity.
- When FIFO FAST mode is set in the UDB CFG16 register, the HFCLK overrides the datapath clock selection normally in use by the FIFO.

16.2.4.2 Reset Control

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

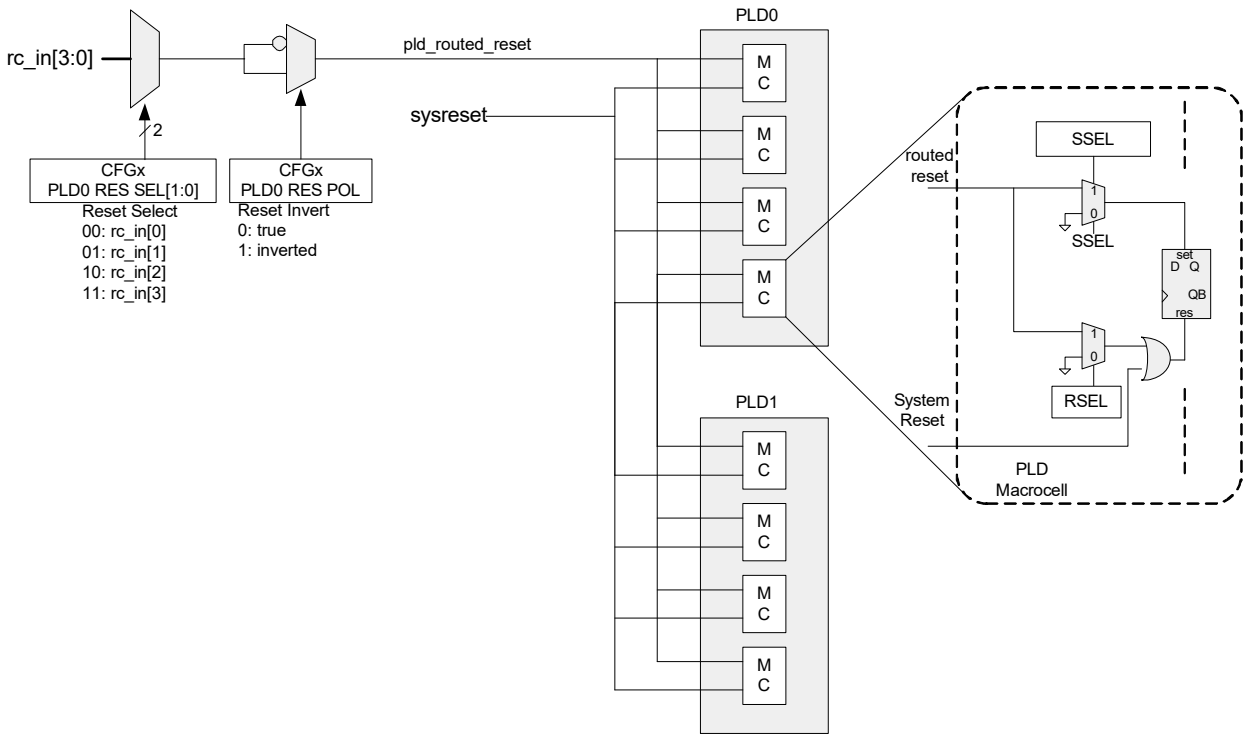
Compatible Reset Scheme

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

Compatible PLD Reset Control

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

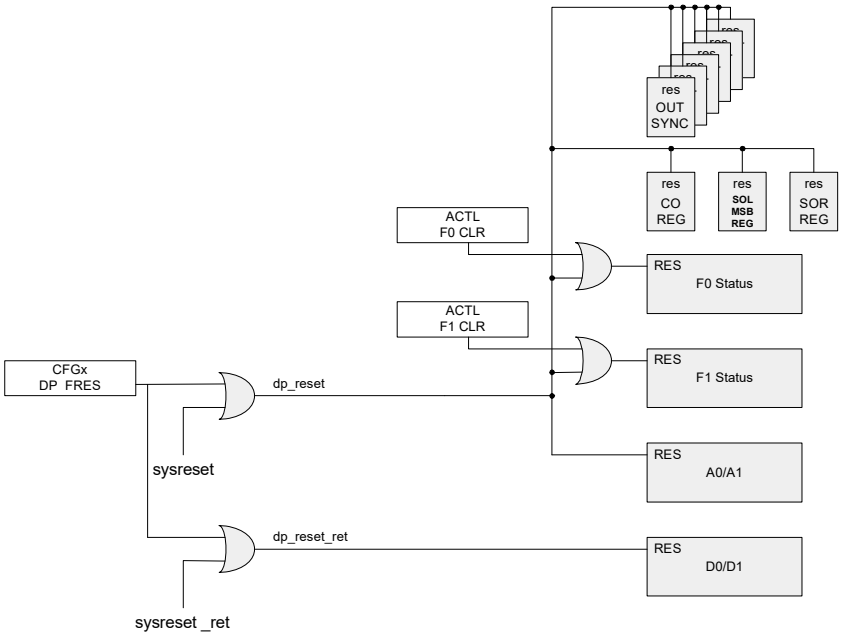
Figure 16-41. Compatible PLD Reset Structure



Compatible Datapath Reset Control

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

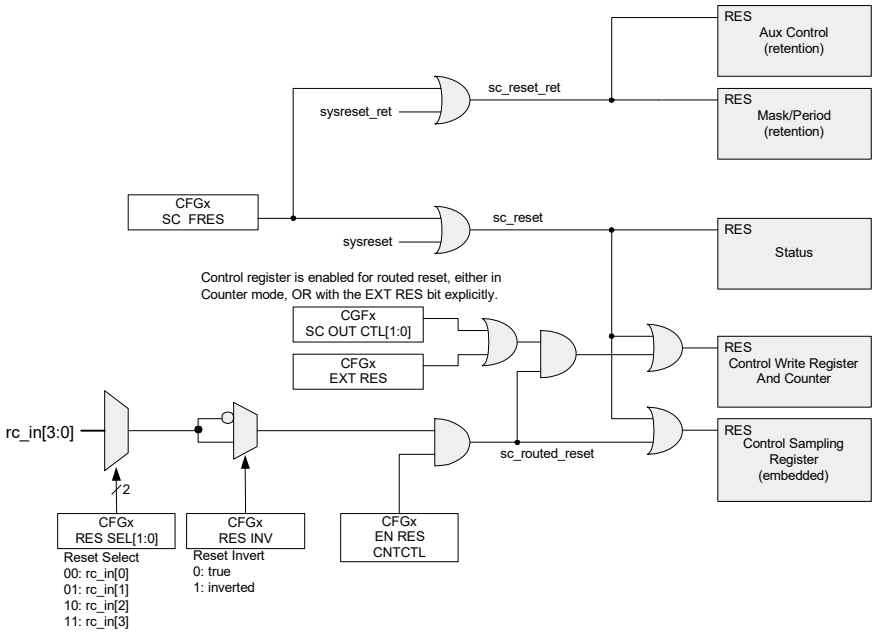
Figure 16-42. Compatible Datapath Reset Structure



Compatible Status and Control Reset Control

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

Figure 16-43. Compatible Status and Control Reset Control



Alternate Reset Scheme

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

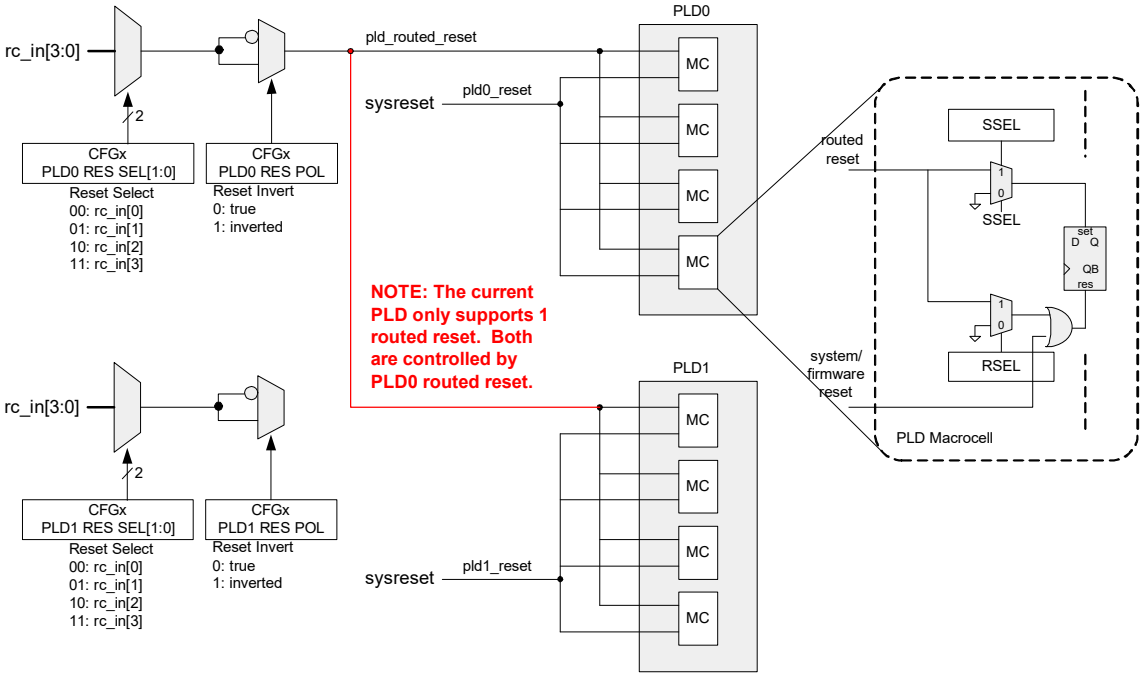
Table 16-23. Reset Schemes

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

Alternate PLD Reset Control

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

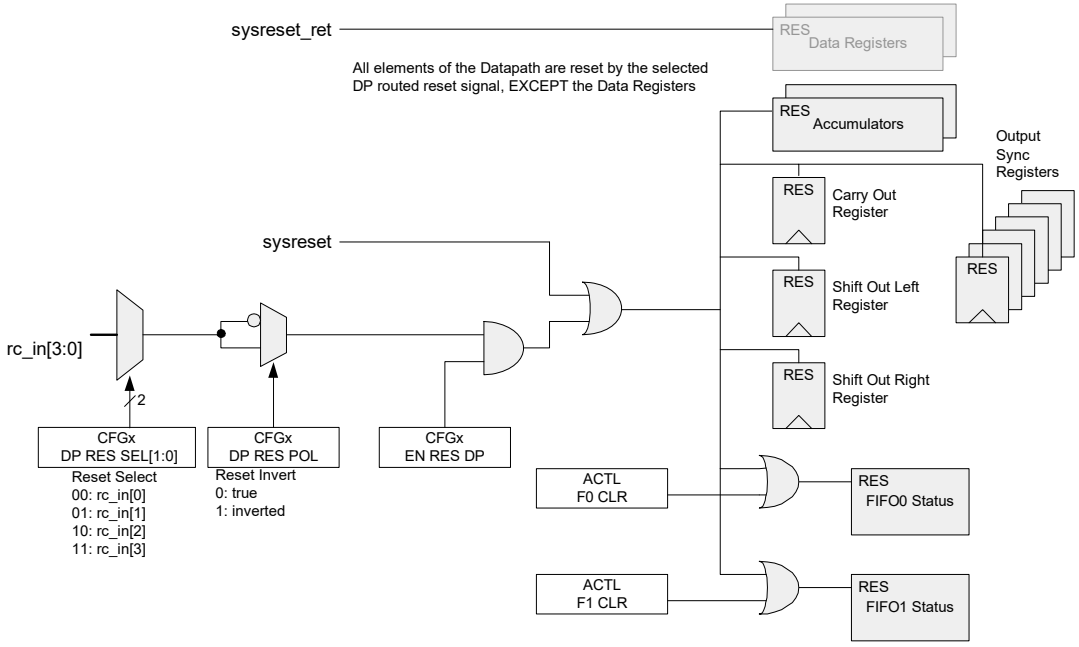
Figure 16-44. Alternate PLD Reset Structure



Alternate Datapath Reset Control

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

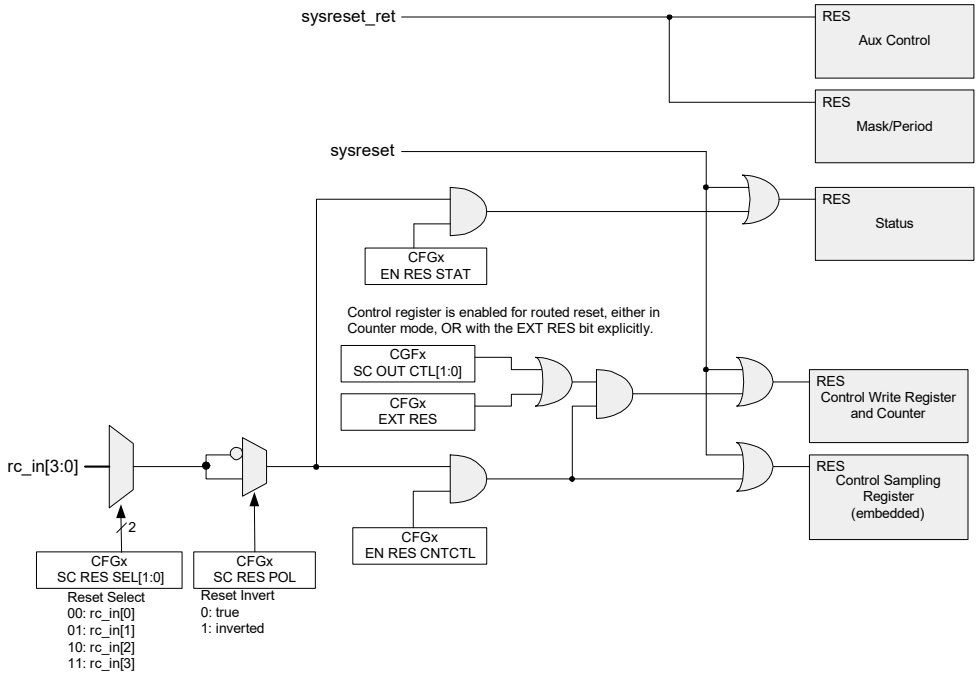
Figure 16-45. Alternate Datapath Reset Structure



Alternate Status and Control Reset Control

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

Figure 16-46. Alternate Status and Control Reset Control



16.2.4.3 UDB POR Initialization

Register and State Initialization

Table 16-24. UDB POR State Initialization

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

Routing Initialization

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

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

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

16.2.5 UDB Addressing

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

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

16.2.6 System Bus Access Coherency

UDB registers have dual access modes:

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

16.2.6.1 Simultaneous System Bus Access

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

Table 16-25. Simultaneous System Bus Access

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

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

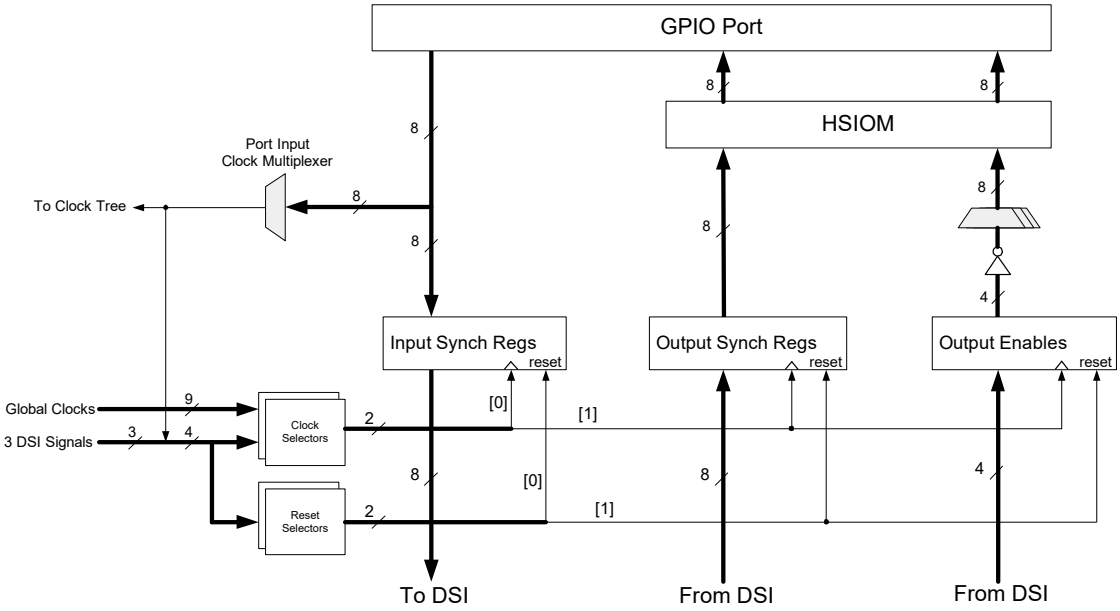
16.2.6.2 Coherent Accumulator Access (Atomic Reads and Writes)

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

16.3 Port Adapter Block

The Port Adaptor block extends the UDBs to provide an interface to the GPIOs through the High-Speed I/O Matrix (HSIOM), described in “High-Speed I/O Matrix” on page 66. The HSIOM places registers for faster routing of DSI signals to GPIO outputs and output enables. The HSIOM also allows GPIOs to be shared amongst multiple blocks, for example port data registers and peripherals such as I2C. Figure 16-47 shows a high-level view.

Figure 16-47. Port Adapter Block Diagram



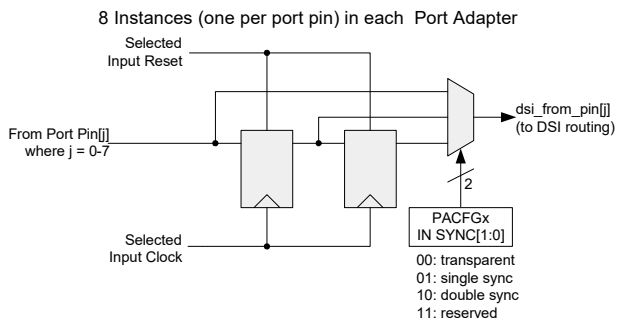
Each 8-bit GPIO port has one port adaptor (PA). There are eight inputs from the GPIO data in, eight outputs to the GPIO data out, and eight output enable (OE) connections. The registers in the PA are used for synchronizing inputs, outputs, and output enables. Another feature is the port input clock multiplexer. This multiplexer selects one of the port inputs to be used as a clock. The clock can be used locally in the PA and routed to the global clocks (see Clocking System chapter on page 72).

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

16.3.1 PA Data Input Logic

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

Figure 16-48. Detail of GPIO Input Logic



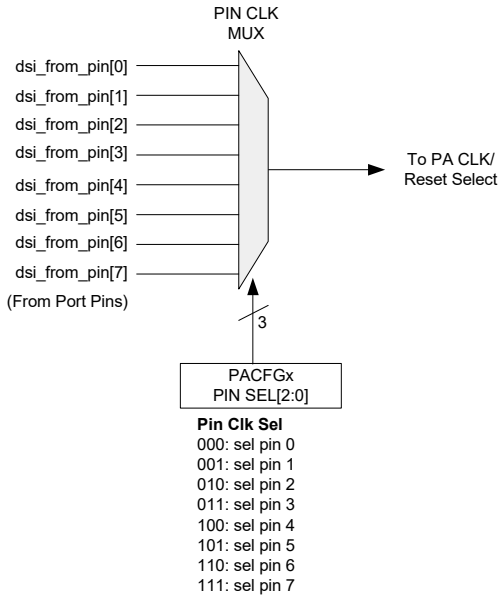
16.3.2 PA Port Pin Clock Multiplexer Logic

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

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

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

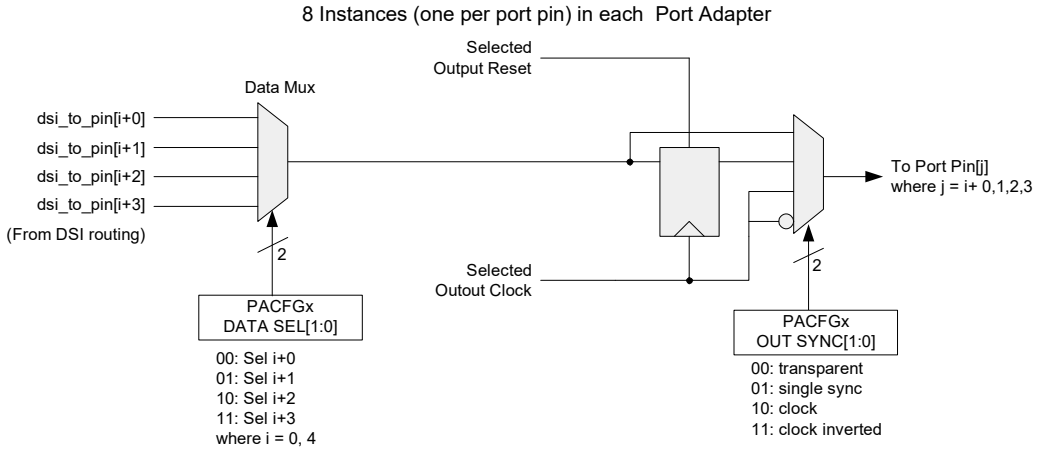
Figure 16-49. Detail of GPIO Pin Selection



16.3.3 PA Data Output Logic

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

Figure 16-50. Detail of GPIO Output Data Logic



16.3.4 PA Output Enable Logic

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

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

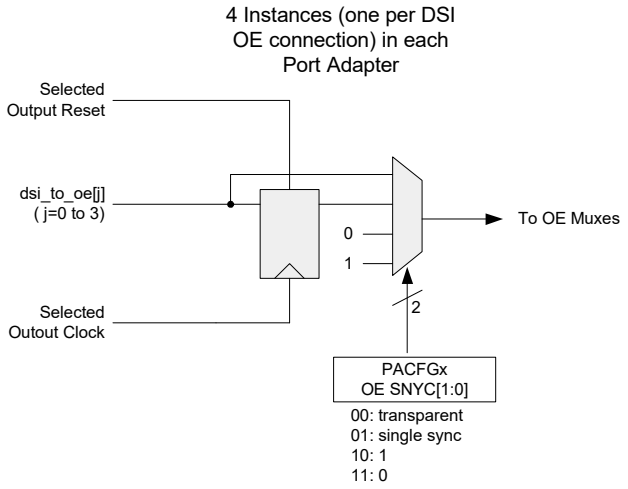
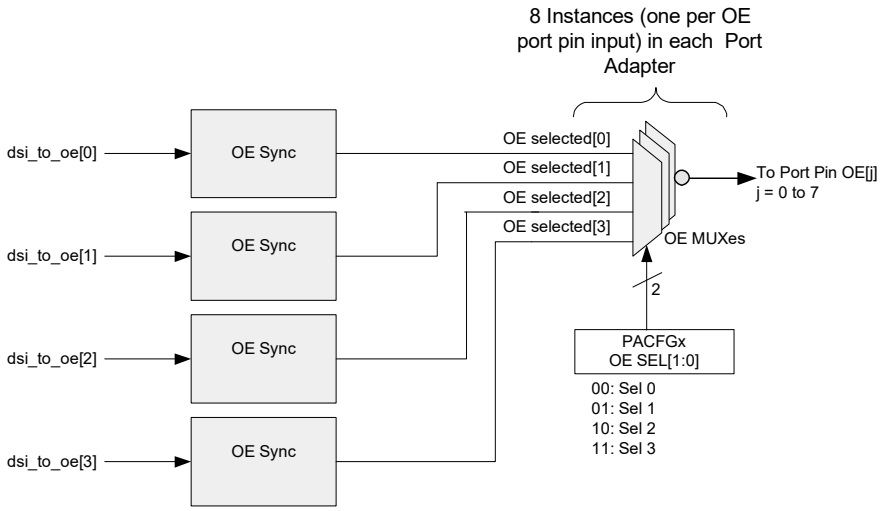


Figure 16-52. GPIO Output Enable (OE) Multiplexers

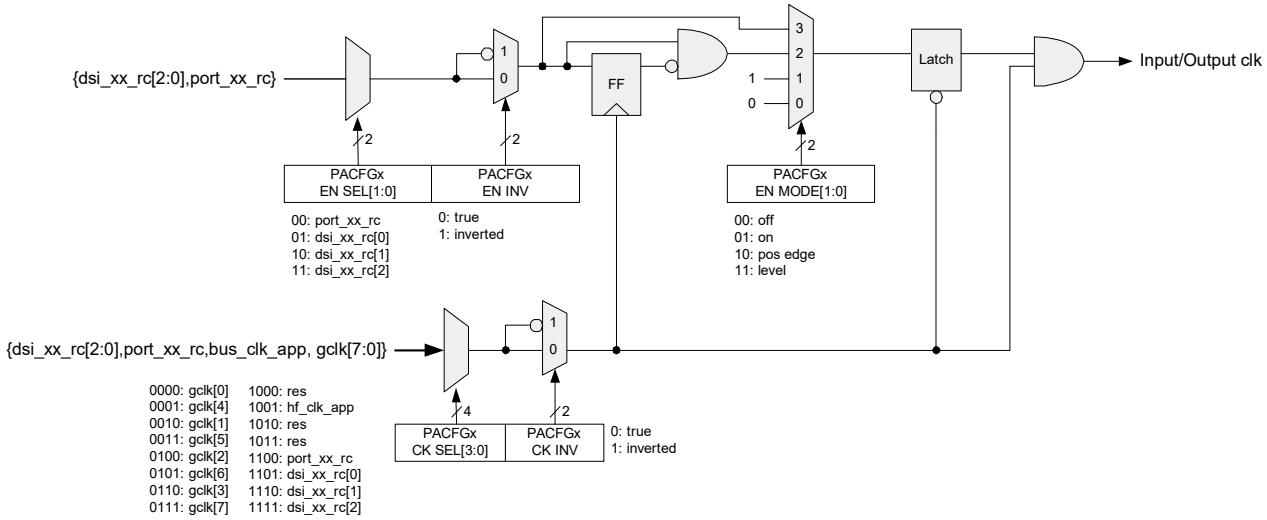


Note that due to the active low sense of the OE signals at the ports, there is an additional inversion in the path between the OE sync logic and the OE multiplexers.

16.3.5 PA Clock Multiplexer

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

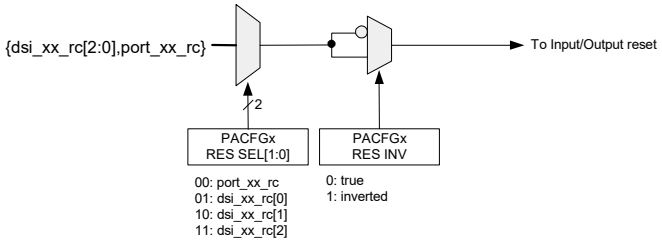
Figure 16-53. PA Clock Multiplexer Detail



16.3.6 PA Reset Multiplexer

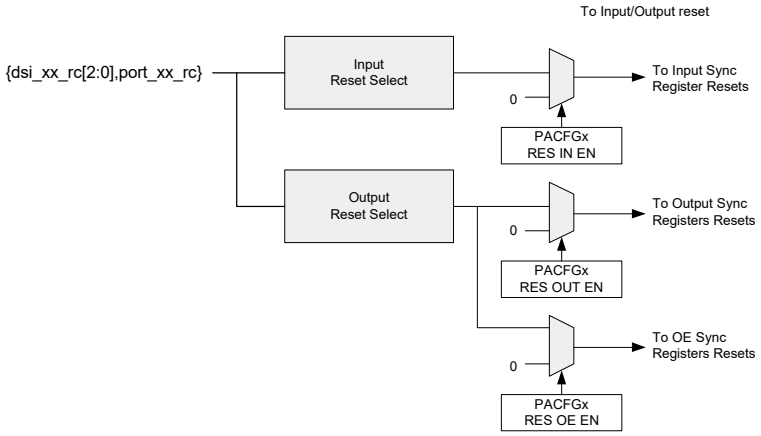
The structure of the PA reset multiplexer is shown in Figure 16-54.

Figure 16-54. PA Reset Multiplexer Detail



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

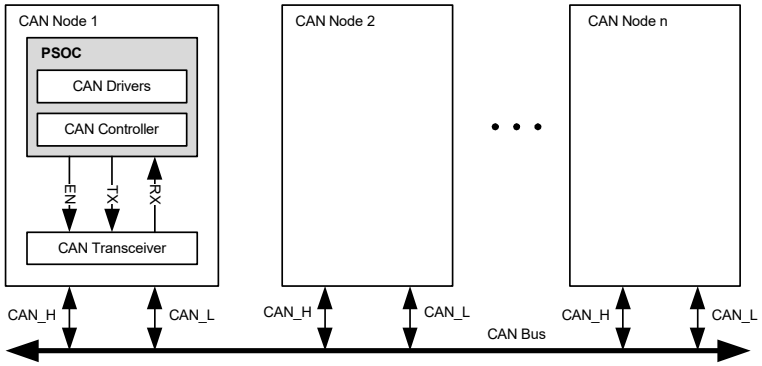
Figure 16-55. PA Reset System



17. Controller Area Network (CAN)

The CAN peripheral is a fully functional Controller Area Network (CAN) supporting communication baud rates up to 1 Mbps. The PSoC 4200L device has two identical CAN controller blocks, which can be routed to different sets of pins. These CAN controllers are CAN2.0A and CAN2.0B compliant to the ISO-11898 specification. The CAN protocol was originally designed for automotive applications with a focus on a high level of fault detection and recovery. This ensures high communication reliability at a low cost. Because of its success in automotive applications, CAN is used as a standard communication protocol for motion-oriented embedded control applications (CANOpen) and factory automation applications (DeviceNet). The CAN features allow the efficient implementation of higher level protocols without affecting the performance of the microcontroller CPU.

Figure 17-1. CAN Bus System Implementation



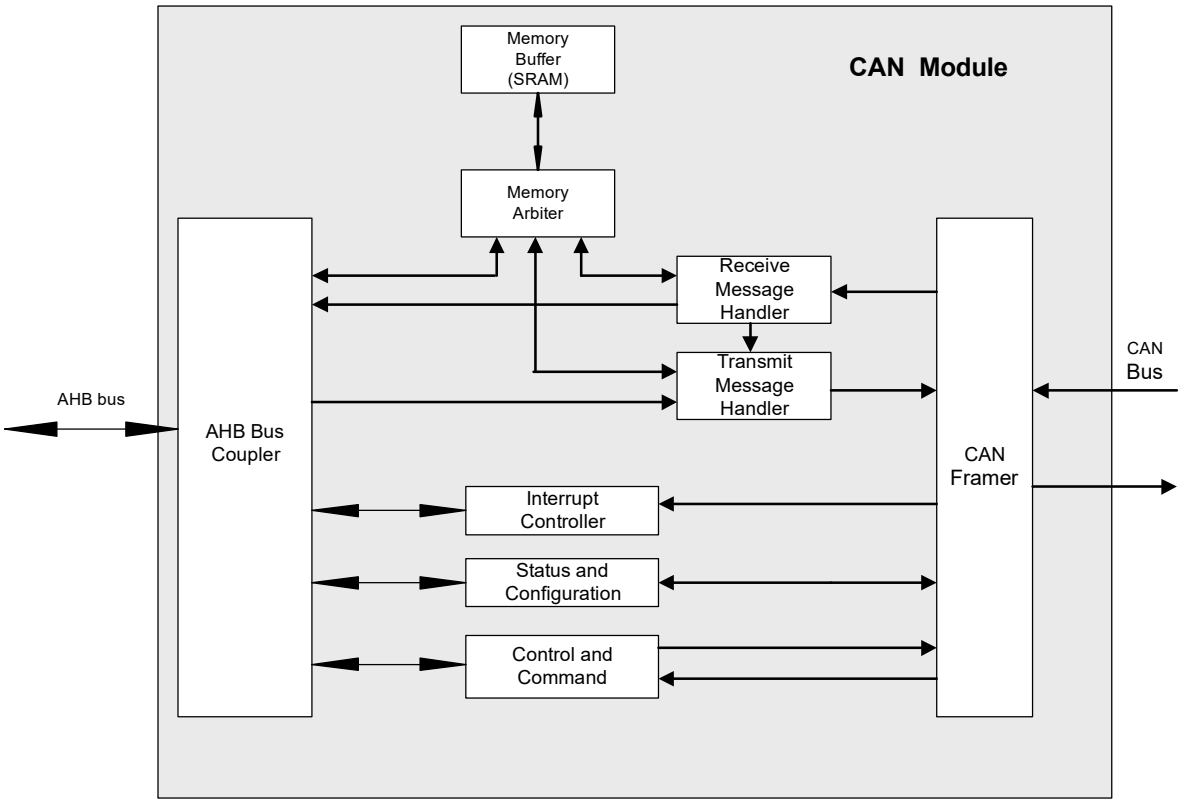
17.1 Features

- Compliant with CAN2.0A/B protocol specification:
  - Standard and extended frames
  - Remote transmission request (RTR) support
  - Programmable bit rate up to 1 Mbps
- Receive path:
  - Sixteen receive message buffers
  - Sixteen acceptance filters and acceptance masks
  - DeviceNet addressing support
  - Option to link multiple receive buffers to form a hardware FIFO
- Transmit path:
  - Eight transmit message buffers
  - Programmable priority for each transmit message buffer
  - Supports single shot transmission of messages
- Listen-Only mode for auto baud detection
- Internal and external loopback modes for block level testing
- Ability to wake up the device from Sleep mode on bus activity
- Counter for implementing time-triggered CAN

17.2 Block Diagram

To transmit a message, the host controller stores a message in the transmit message buffer and informs the transmit message handler, which transmits the message. When a message is received, it is stored in the memory buffer and the host controller can process it on demand. The transmission and reception are mainly governed by the status and configuration registers. The interrupt controller unit handles various interrupts of the CAN module. Figure 17-2 illustrates this process.

Figure 17-2. CAN Block Diagram



17.3 CAN Message Frames

In CAN, four main frame types govern the transmission and reception of messages:

- Data frames
- Remote frame
- Error frame
- Overload frame

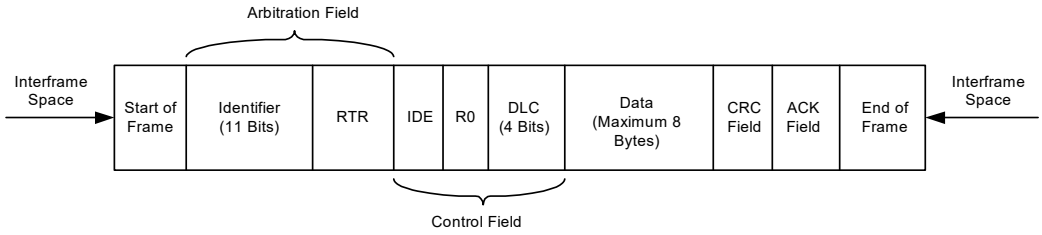
17.3.1 Data Frames

Data frames are mainly used to transfer data between transmitter and receiver. CAN supports mainly two types of data frames: Standard Data Frame and Extended Data Frame. For a CAN frame, '0' is referred to as the dominant bit and '1' as a recessive bit.

17.3.1.1 Standard Data Frame

Figure 17-3 illustrates the standard data frame for CAN.

Figure 17-3. Standard Data Frame



Start of frame. The beginning of a data frame is indicated by the start of frame bit. It is a single dominant bit.

Identifier. For a basic CAN data frame, the identifier is 11 bits long. It is mainly used to filter the data at the receiver side.

Remote Transmission Request Bit (RTR). Set the RTR bit '0' (dominant) for a data frame and set to '1' (recessive) for a remote frame. The identifier and RTR bit are known as the Arbitration Field.

Extended Identifier Bit (IDE). This bit must be a '0' (dominant for a standard data frame and a '1' (recessive) for extended CAN data frame.

R0. Reserved bit.

Data Length Code (DLC). These four bits indicate the number of data bytes in the data field. The IDE, R0, and DLC bits constitute the Control Field.

Data Field. This field contains the message data. It is of variable length and can have a maximum of 8 bytes.

Cyclic Redundancy Check (CRC). Frame checking is carried out by the method of cyclic redundancy check (CRC). The field consists of a 15-bit CRC code followed by a CRC delimiter.

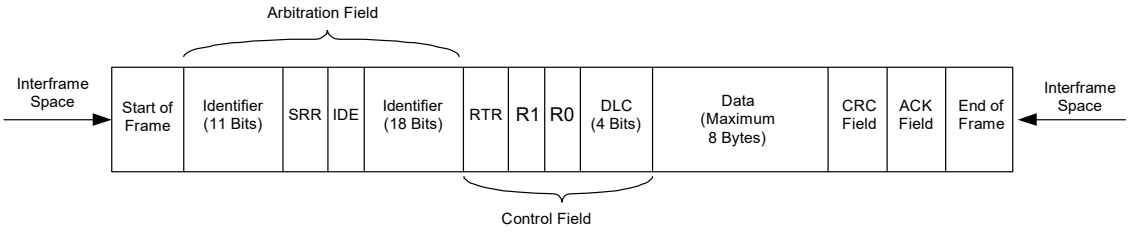
Acknowledgement Field (ACK). The ACK field is two bits long and recessive by default. When a receiver receives a message correctly, it overwrites the ACK field with a dominant bit.

End of Frame. The end of every frame is indicated by the End of Frame field and it consists of seven recessive bits.

17.3.1.2 Extended Data Frame

The extended CAN frame format is illustrated in Figure 17-4. The extended CAN has a 29-bit identifier. It is arranged as an 11-bit identifier field and an 18-bit identifier field separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit is in the same position as the RTR bit in the standard frame, and is recessive. The IDE bit is set for extended frames. The Control Field of the extended data frame has an additional reserve bit 'R1' compared to the standard data frame.

Figure 17-4. Extended Data Frame



17.3.2 Remote Frame

The CAN bus allows a destination node to request data from the source by sending a remote frame. There are two differences between a data frame and a remote frame: the RTR bit is transmitted as a recessive bit in the remote frame and there is no Data Field in the remote frame.

For extended remote frames, the SRR bit is also transmitted as a recessive bit.

Interframe Space. Interframe space separates the data frames and remote frames from the preceding frames.

17.3.3 Error Frame

When a node detects a bus error, it generates an error frame. The error frame consists of an error flag and error delimiter. The error flag is classified into two types: error active flag and error passive flag.

Error Active Flag. When an error active node detects an error, it sends six dominant bits as an active error flag. The format of the error flag thus violates the rule of bit stuffing. This forces all other nodes to send out error flags resulting in a series of six to twelve dominant bits on the bus.

Error Passive Flag. An error passive flag consists of six recessive bits. When an error passive node detects an error it sends a passive error flag. A passive error does not affect any other nodes and the error is detected only if the transmitting node detects a bus error.

Error Delimiter. The error delimiter consists of eight recessive bits. After transmission of an ERROR FLAG, each station sends 'recessive' bits and monitors the bus until it detects a 'recessive' bit. Later, it transmits seven more 'recessive' bits.

17.3.4 Overload Frame

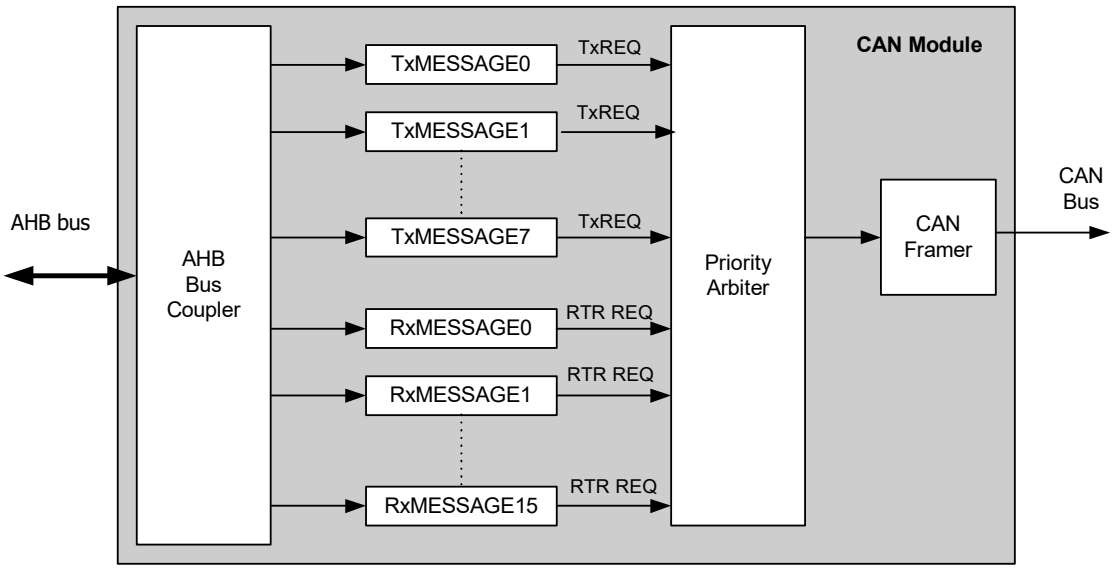
The overload frame (OF) consists of an overload flag and an overload delimiter. PSoC 4200L CAN controller supports reactive overload frame, which is activated when the following conditions occur:

- Detection of a dominant bit during the first two bits of intermission
- Detection of a dominant bit in the last bit of EOF by a receiver
- Detection of a dominant bit by any node at the last bit of error delimiter or overload delimiter

17.4 Transmitting Messages in CAN

The CAN module supports eight transmit message holding buffers. An internal priority arbiter selects the message according to the chosen arbitration scheme. The arbitration scheme is either a round robin or fixed priority scheme. When a message is transmitted or when there is a message arbitration loss, the priority arbiter re-evaluates the message priority of the next message. The receive message buffers can also transmit remote transmit requests, which are explained later in this chapter.

Figure 17-5. Transmit (Tx) Block Diagram



17.4.1 Message Arbitration

The priority arbiter supports a round robin and fixed priority arbitration. The arbitration mode is selected using the configuration register.

Round Robin. In a round robin scheme, Buffer 0 is selected first, then Buffer 1, and so on until Buffer 7; this continues again with Buffer 0 thus forming a cycle. A particular buffer is selected only if its TX_REQ bit is set. This scheme guarantees that all buffers receive the same probability to send a message.

Fixed Priority. Buffer 0 has the highest priority. Designate Buffer 0 as the buffer for critical messages to guarantee that message is sent first. Priority arbitration is selected using the CFG_ARBITER bit in the Configuration register (CAN_CONFIG[12]).

Note: RTR message requests are served before TxMessage buffers are handled.

17.4.2 Message Transmit Process

Figure 17-6 shows the registers associated with a message that is transmitted.

Figure 17-6. Transmit (Tx) Message Registers

REGISTERS											
COMMAND REGISTER (CAN_Txn_CMD)	Reserved [31:24]	WPN2 [23]	Reserved 1 [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:4]	WPN1 [3]	Tx INT ENBL [2]	Tx ABORT [1]	Tx REQ [0]
IDENTIFIER (CAN_Txn_ID)	ID [31:3]									Reserved [2:0]	
DATA REGISTER High (CAN_Txn_DH)	D0 [63:56]			D1 [55:48]			D2 [47:40]			D3 [39:32]	
DATA REGISTER Low (CAN_Txn_DL)	D4 [31:24]			D5 [23:16]			D6 [15:8]			D7 [7:0]	

n = چھ

The main steps in transmitting a standard data frame are:

1. Write the message into an empty transmit message holding buffer. An empty buffer is indicated by the TX_REQ bit equal to zero.
   a. For standard data frame, write '0' (dominant) to the RTR and IDE bit.
   b. Write the DLC bits appropriately to specify the number of data bytes to be transferred. The maximum number of data bytes is limited to eight. Data bytes with MSB (most significant bit) first in each byte are written in D0, D1…D7 locations.
   c. The 11-bit message identifiers are written to the ID[31:21] bit field.
2. Choose an appropriate priority arbitration scheme. The internal message priority arbiter selects the message according to the chosen arbitration scheme.
3. Request transmission by setting the respective TX_REQ bit to '1'.
4. The TX_REQ bit remains set as long as the message transmit request is pending. The content of the message buffer must not be changed while the TX_REQ bit is set.

After the message is transmitted, the TX_REQ bit is cleared and the TX_MSG interrupt status bit. [CAN_INT_STATUS[11] in the interrupt status register CAN_INT_STATUS is asserted. The interrupt status bit is only asserted if the TxINT ENBL (CAN_TX[n].CONTROL[2]) is set to '1'.

17.4.3 Message Abort

A message is aborted by setting the TX_ABORT bit (CAN_TX[n]_CONTROL[1]) in the CAN_TX[n]_CONTROL register. This bit is automatically cleared by the hardware when the message is aborted.

Notes

- The CAN Buffer register (CAN_BUFFER_STATUS) is used to read whether any transmission requests are pending.
- If the write protect bit wpn2 (CAN_TX[n].CONTROL[23]) is '0', then bits [21:16] of the Command register cannot be modified because they are protected and provides an undefined value on read back.
- If the write protect bit wpn1 (CAN_TX[n].CONTROL[3]) is '0', then bit [2] of the Command register cannot be modified. This bit gives a '0' upon read back.
- Using the WPN flags(wpn1 and wpn2) enables simple retransmission of the same message by only having to

set the TX_REQ bit without taking care of the special flags (RTR, IDE, DLC, and TxINTENBL).

17.4.4 Single Shot Transmission

The single shot transmission mode is used in systems where the retransmission of a CAN message due to an arbitration loss or a bus error must be prevented. This is particularly useful in time triggered CAN systems where all the messages are transmitted at a fixed time.

A single shot transmission request is set by asserting CAN_TX.CONTROL.TX_REQ and TX_ABORT bits at the same time. Upon a successful message transmission, both bits are cleared.

If an arbitration loss or a bus error happened during the transmission, the TX_REQ bit is cleared, but the TX_ABORT bit remains asserted. At the same time, the single shot transmission failure (sst_failure) interrupt is asserted.

17.4.5 Transmitting Extended Data Frames

To transmit an extended data frame, certain register settings must change compared to that of a standard data frame. These changes are as follows.

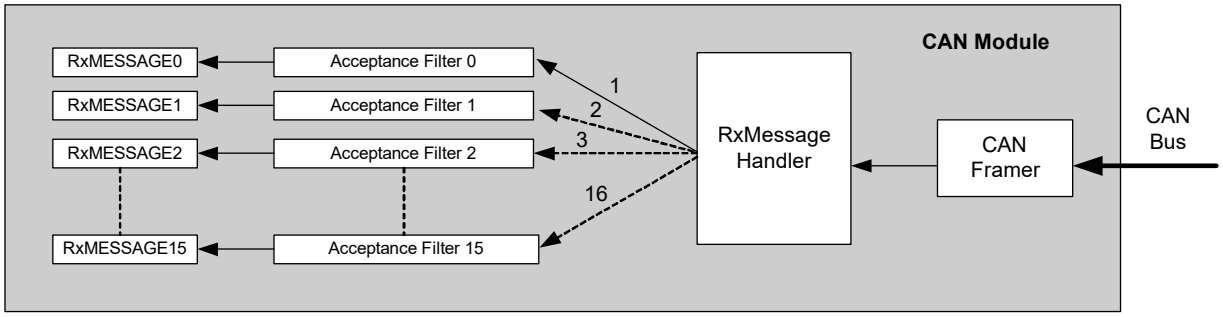
- For extended date frame, write '1' (recessive) to the IDE bit.
- The message identifiers are written to the ID[31:3] bit field.

17.5 Receiving Messages in CAN

The CAN module has 16 receive message buffers as illustrated in Figure 17-7. Each message buffer has a dedicated acceptance filter. The CAN message is received by the CAN frame and then the received message is simultaneously compared with all the acceptance filters and the accepted message is stored in the respective receive message buffer. The message available (MSG_AV) bit in the message buffer is set to indicate the availability of the new message. Message receipt must be acknowledged by clearing the MSG_AV bit to allow receipt of another message.

The acceptance filter is configured by the Acceptance Mask Register (AMR) and the Acceptance Code Register.

Figure 17-7. Receive (Rx) Block Diagram



17.5.1 Message Receive Process

Figure 17-8 shows the registers associated with a received message.

Figure 17-8. Receive (Rx) Message Registers

REGISTERS															
COMMAND REGISTER (CAN_RXn_CMD)	Reserved [31:24]	WPNH [23]	Reserved1 [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:8]	WPNL [7]	LINK FLAG [6]	RX INT ENBL [5]	RTR REPLY [4]	BUFFER EN [3]	RTR ABORT [2]	RTR REPLY PNDG [1]	MSG AV [0]
IDENTIFIER (CAN_RXn_ID)	ID [31:3]													Reserved [2:0]	
DATA REGISTER High (CAN_RXn_DH)	D0 [63:56]					D1 [55:48]			D2 [47:40]				D3 [39:32]		
DATA REGISTER Low (CAN_RXn_DL)	D4 [31:24]					D5 [23:16]			D6 [15:8]				D7 [7:0]		

n = عجشد□

The main steps in receiving a message are:

1. After receipt of a new message, the RxMessageHandler hardware (as seen in Figure 17-7) searches all receive buffer starting from RxMessage0 until it finds a valid buffer. A valid buffer is indicated by:
   a. Receive buffer is enabled indicated by BUFFER EN = '1' (CAN_RX[n].CONTROL[3]).
   b. Acceptance filter of the receive buffer matches incoming message.
2. If the RxMessageHandler finds a valid buffer that is empty, then the message is stored and the MSG AV bit of this buffer is set to '1'.
3. If the RX INT ENBL bit is set, then the RX_MSG flag (CAN_INT_STATUS[12]) of the interrupt controller is asserted.
4. If the receive buffer already contains a message indicated by MSG AV = '1' and the LINK_FLAG bit is not set, then the RX_MSG_LOSS interrupt flag (CAN_INT_STATUS[10]) is asserted. The new received message will be discarded.

Note: The CAN buffer register (CAN_BUFFER_STATUS) determines if any receive message buffer is available.

17.5.2 Acceptance Filter

Each receive buffer has its own acceptance filter that is used to filter incoming messages. An acceptance filter is configured by the Acceptance Mask register (AMR) and the Acceptance Code register (ACR). The AMR defines which bits of the incoming message must match the respective ACR bits for accepting the message.

AMR: '0'. The incoming bit is checked against the respective ACR bit. The message is not accepted when the incoming bit does not match respective ACR bit.

AMR: '1'. The incoming bit is Do Not Care.

Following message fields are covered:

- Identifier
- IDE
- RTR
- Data byte 1(D0) and data byte 2(D1) (DATA[63:48])[1]

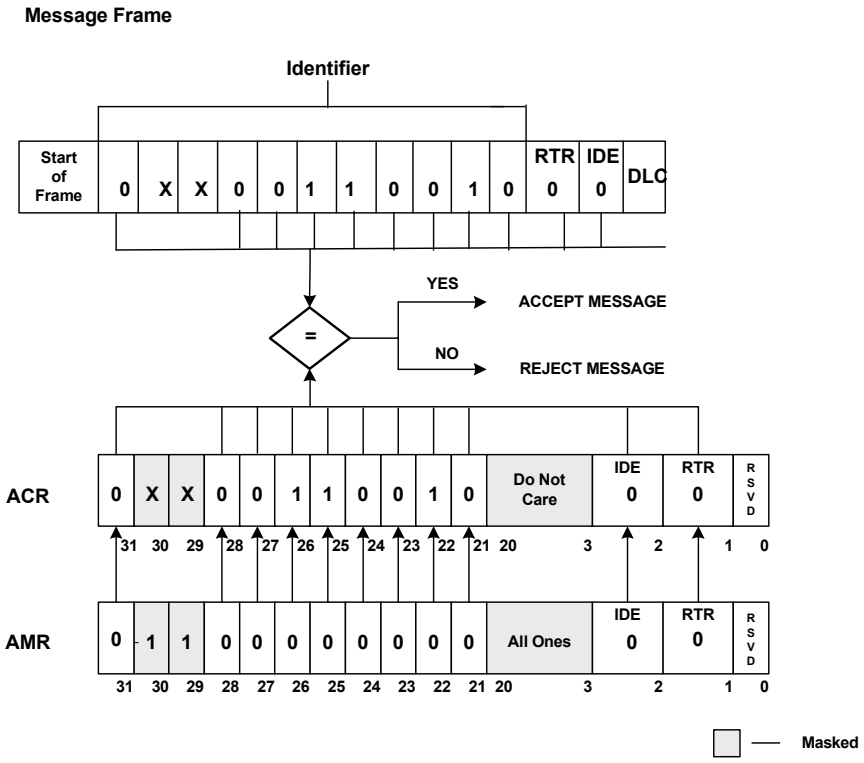
For a standard CAN message when IDE=0, the 11-bit identifier are the bits [31:21] of AMR and ACR.

1. Useful for DeviceNet filtering as given in "DeviceNet Filtering" on page 197.

17.5.2.1 Example

A message and the acceptance filter settings to accept that message are shown in Figure 17-9.

Figure 17-9. Acceptance Filter



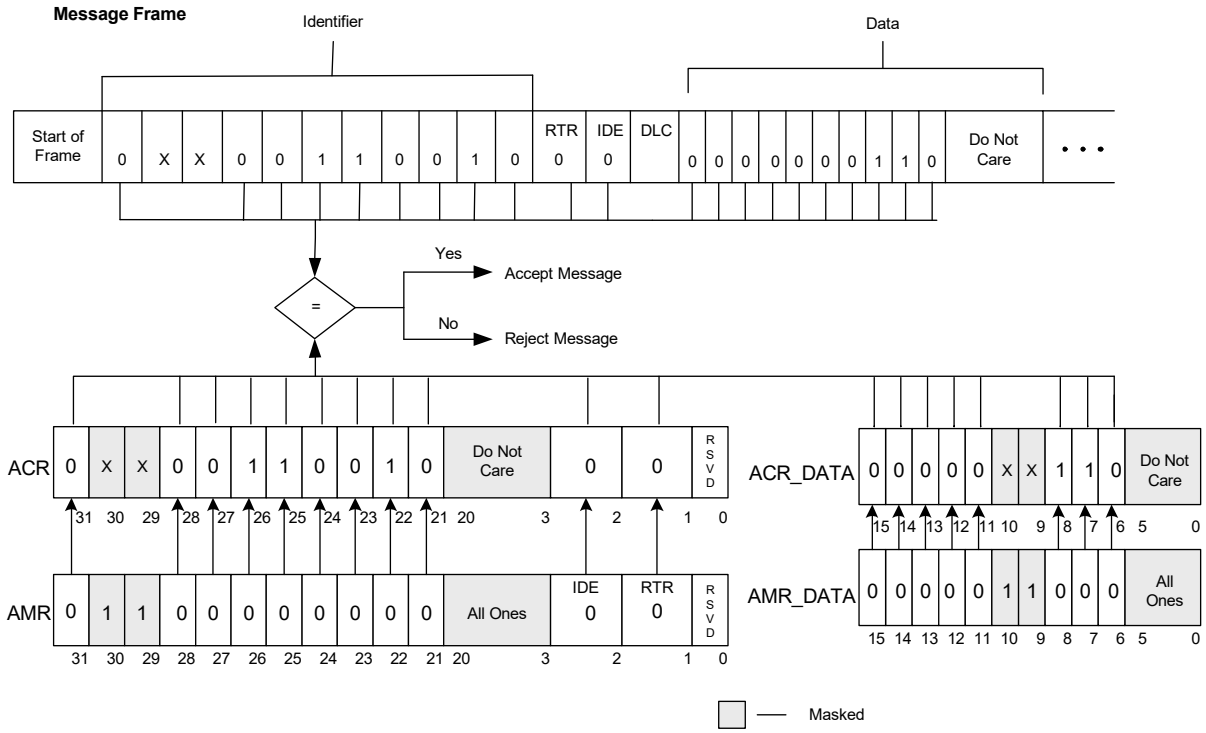
As seen in Figure 17-9, the shaded areas are masked bits. When a bit is set to '1' in the AMR register, the corresponding bit in the ACR register is not checked against the received message frame. In the example, bits 30, 29, and bits from 3 to 20 are set to '1' and are masked. Because other bits in the AMR register are written as '0', the respective bits in the ACR register are compared with message bits as shown in Figure 17-9. If the corresponding bits in ACR match with that of the message, the message is then stored in the receive message buffer. If the corresponding bits in ACR do not match with the message, the incoming message is rejected.

AMR Settings:	**ACR Settings:**
ID[31], ID[28:21] = 0	ID[31:21] = 182h
ID[30], ID[29] = 1	ID[20:3] = Do Not Care
ID[20:3] = All Ones	IDE = 0
IDE = 0	RTR = 0
RTR = 0	

17.5.3 DeviceNet Filtering

For some CAN high-level protocols such as DeviceNet, additional protocol related information is contained in the first and second data bytes. The acceptance filters provide additional coverage of these two bytes for a more efficient implementation of the protocol. The data bits of the first two bytes of the incoming message are compared with the ACR_DATA register (CAN_RX[n]_ACR_DATA) and the respective bits that are compared are specified using AMR_DATA register (CAN_RX[n]_AMR_DATA). Using the "Example" on page 197, DeviceNet filtering is illustrated in Figure 17-10.

Figure 17-10. DeviceNet Filter



In Figure 17-10, the data field of the message frame is compared with those bits of the ACR_DATA register, which are not masked by the AMR_DATA register.

To accept this message, the acceptance filter settings are as follows:

AMR Settings:		**ACR Settings:**
ID[28:21],ID[31] = 0		ID[31:21] = 182h
ID[30],ID[29] = 1		ID[20:3] = Do Not Care
ID[20:3] = All Ones		IDE = 0
IDE = 0		RTR = 0
RTR = 0		ACR_DATA[15:6] = 06h
AMR_DATA[15:11], DATA[8:6] = 0	AMR_-	ACR_DATA[5:0] = Do Not Care
AMR_DATA[10:9], DATA[5:0] = All ones	AMR_-	

The example in Figure 17-10 shows the filtering using 10 data bits. Using AMR_DATA, up to 16 data bits, can be used for filtering.

17.5.4 Filtering of Extended Data Frames

Filtering the extended data frame is similar to the standard date frame with the following exception: the IDE bit in AMR and ACR registers must be set to check for extended data frame.

17.5.5 Receiver Message Buffer Linking

Several receive buffers can link together to form a receive buffer array that acts similar to a receive FIFO. To accomplish this, do the following:

- Set the LINK_FLAG bit in CAN_RX[n].CONTROL[6] for the buffers that need to be linked.
- Make sure that all buffers of the same array have the same message filter setting (AMR and ACR are identical).
- Do not set the LINK_FLAG bit of the last buffer of an array.

When a receive buffer already contains a message (MSG AV='1') and a new message arrives for this buffer, then this message is discarded (RX_MSG_LOSS Interrupt). To avoid this situation, several receive buffers are linked together. When the CAN controller receives a new message, the RxMessageHandler searches for a valid receive buffer. If one is found that is already full (MSG AV = '1') and the 'LINK_FLAG' is set, the search for a valid receive buffer is continued. If a valid receive buffer is found, the message is transferred to that buffer thereby forming an array. If no other buffer is found, then the RX_MSG_LOSS interrupt is set.

It is possible to build several message arrays. Each of these arrays must use the same AMR and ACR.

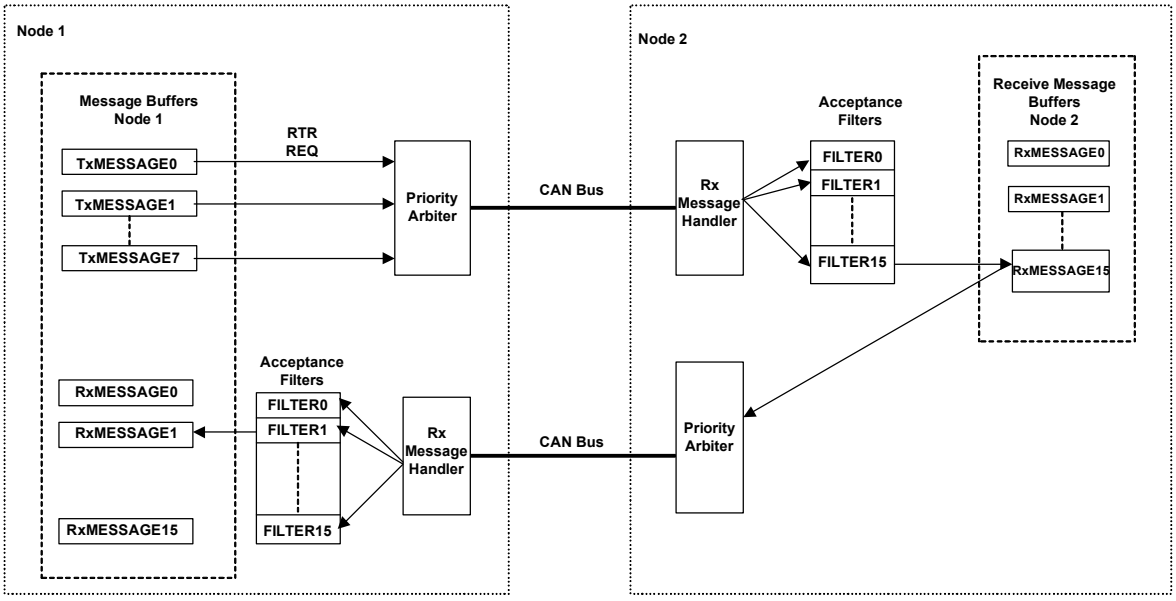
17.6 Remote Frames

Remote frames are used for initiating transmission between two nodes and the node acting as a receiver sends the remote frame. A remote frame can use either standard format or extended format. A remote frame is different from a data frame in that the RTR bit is always equal to '1' and the data field is absent, independent of the value of DLC field. The flow of a remote transmit request is illustrated in Figure 17-11.

As shown in Figure 17-11:

- The message buffer0 of node1 transmits a remote frame into the CAN bus.
- The RTR request is received by the RxMessageHandler of node 2 and sends it to the acceptance filters.
- The acceptance filter settings of the receive message buffer 15 matches with that of the message and then the message is moved to the receive message buffer 15.
- If the RTR Auto Reply feature is enabled, the receive message buffer 15 will transmit the message with the same identifier as it received (without CPU intervention).
- On successful transmission of RTR reply MSG_AV_RTR_SENT bit in the CAN_RX.CONTROL register is set, and RTR_MSG bit in the interrupt status register is set.
- The acceptance filter of the receive message buffer 1 of node1 has the same identifier settings as that of the transmitted message node 1. Hence, the RTR message will be stored in the receive message buffer 1 of node 1.

Figure 17-11. Remote Transmit Request



17.6.1 Transmitting a Remote Frame by the Requesting Node

The process to transmit a remote frame by a requesting node (Node 1 as shown in Figure 17-11) is as follows.

1. Write a message to an empty transmit buffer. An empty buffer is indicated by TX_REQ = '0' (CAN_TX[n]_CONTROL[0]).
2. Set the RTR bit (CAN_TX[n]_CONTROL[21]) to '1'.
3. Choose an appropriate priority arbitration scheme.
4. Set the transmit request flag to initiate transmission.
5. The Identifier transmitted in a message must be the same as the identifier of receiving message.

17.6.2 Receiving a Remote Frame

The process to receive a remote frame is as follows.

1. The acceptance filter must be configured to receive the desired message ID.
2. Enable the automatic RTR message handling by setting bit 'RTR REPLY' to '1'.
   a. If enabled, it will automatically transmit the remote frame with the same identifier.
   b. Otherwise, the remote frame must be transmitted following the standard routine as that of a data frame.
3. Set the requesting node that receives the replied RTR message to receive a normal message. Do not set the RTR REPLY bit.

17.6.3 RTR Auto Reply

The CAN module supports automatic answering of RTR message requests. All 16 receive buffers support this feature. If an RTR message is accepted in a receive buffer where the RTR REPLY FLAG is set, then this buffer automatically replies to this message with the content of the receive buffer. The 'RTR REPLY PNDG FLAG' is set when the RTR message request is received. It is reset when the message is sent or when the message buffer is disabled. To abort a pending RTR reply message, use the RTR ABORT command.

17.6.4 Remote Frames in Extended Format

The transmission and reception of remote frames in extended format is similar to standard format except for the following.

- The IDE bit (CAN_TX[n].CONTROL [20]) is set to '1' to make it an extended data frame.
- The identifier is 29 bits long compared to the 11 bits of a standard data frame.

17.7 Time-Triggered CAN

Time-triggered CAN (TTCAN) is a higher level protocol layer on top of the CAN protocol itself. TTCAN is useful for applications where the CAN messages are periodic in nature. In a TTCAN system, the synchronization of messages is done with respect to a reference message from the time master in the network. The following features available in CAN controller makes it suitable for the implementation of level 1 TTCAN systems:

- Ability for single shot transmission, as explained in 17.4.4 Single Shot Transmission.
- An internal timer for timing the TTCAN messages

Note: For TTCAN receive buffer configured for receiving remote frames, automatic message reply should be disabled by clearing the CAN_RX.CONTROL.RTR_REPLY bit.

17.7.1 TTCAN Timer

A 16-bit timer sub block is included in the CAN block, which can be used as the timer for TTCAN system. This block is enabled based on the CAN control register setting (CNTL.TT_ENABLE). The timer counts on nominal CAN bit time and captures the timer count on detection of Start of Frame (SOF) on the CAN bus. The registers associated with TTCAN timer are the following:

- TTCAN_COUNTER

This is the 16-bit local timer counter register (TTCAN_COUNTER.LOCAL_TIME). It counts on nominal bit time, based on the bit timing settings in the TTCAN_TIMING Register.

- TTCAN_COMPARE

This is the compare value of the local counter to generate a time event. When TTCAN_COUNTER.LOCAL_TIME counts to TTCAN_COMPARE register value, tt_compare hardware event is triggered. If the TT_COMPARE bit in the interrupt register (INTR_CAN_SET) is enabled, the corresponding bit in the interrupt status register (INTR_CAN) is set. See 17.9.4.2 Interrupt Routing with TT_ENABLE = 1 for details.

- TTCAN_CAPTURE

This register captures the value of the TTCAN_COUNTER when an SOF is detected on the CAN bus (CAN Rx input). This will also trigger the tt_capture hardware event; if the TT_CAPTURE bit in the interrupt register (INTR_CAN_SET) is enabled, the corresponding bit in the interrupt status register (INTR_CAN) is set. See 17.9.4.2 Interrupt Routing with TT_ENABLE = 1 for details.

The SOF of the reference frame is used as the synchronization signal for a TTCAN system.

The AMR/ACR registers can be used to filter the message ID of reference message. The TTCAN_CAPTURE register value can be read on detection of reference message to synchronize the time of the reference message.

- TTCAN_TIMING

This register configures the nominal bit timing to generate the clock for TTCAN counter. This register must be configured to the same bit time settings as the CAN configuration register.

Note: The TTCAN system design must be taken care at the application level. The systems designer needs to decide on how to make use of the available hardware resources to build a complete TTCAN system.

17.8 Bit Time Configuration

The CAN module operates on a single clock input SYSCLK. This section explains how to configure the programmable bit-rate divider to achieve the desired bit rate and its relationship with SYSCLK.

17.8.1 Allowable Bit Rates and System Clock (SYSCLK)

Across the industry, most implementations of CAN-Bus use one of 10 bit rates:

- 1 Mbps
- 800 Kbps
- 500 Kbps
- 250 Kbps
- 125 Kbps
- 100 Kbps
- 50 Kbps

- 20 Kbps
- 10 Kbps
- 5 Kbps

All these bit rates are configurable if SYSCLK is a multiple of 8 MHz. A minimum SYSCLK of 10 MHz is required to support the maximum 1-MHz bit rate. All except 800 Kbps are configurable if SYSCLK is 10 MHz or a multiple. With a few exceptions, all 10 bit rates are not possible if SYSCLK is not evenly divisible by 1,000,000 Hz. For bit-rate generation, the accuracy for SYSCLK must be at least 1.58 percent for 125 Kbps and slower bit rates, and 0.5 percent or better for bit rates faster than 125 Kbps. To meet the accurate clocking requirements of the CAN block, use an external crystal oscillator (ECO) as the clock source for SYSCLK or route an accurate external clock into the device and use it as the SYSCLK source. Figure 17-12 shows a table of the 10 bit rates that are supported for any given fclk frequency from 48 MHz to 100 MHz. The maximum possible frequency for PSoC 4200L is 48 MHz.

Figure 17-12. Bit Rate Versus SYSCLK

clk_bus Freq (MHz)	1 Mb	800 Kb	500 Kb	250 Kb	125 Kb	100 Kb	50 Kb	20 Kb	10 Kb	5 Kb
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										
22										
23										
24										
25										
26										
27										
28										
29										
30										
31										
32										
33										
34										
35										
36										
37										
38										

clk_bus Freq (MHz)	1 Mb	800 Kb	500 Kb	250 Kb	125 Kb	100 Kb	50 Kb	20 Kb	10 Kb	5 Kb
39										
40										
41										
42										
43										
44										
45										
46										
47										
48										

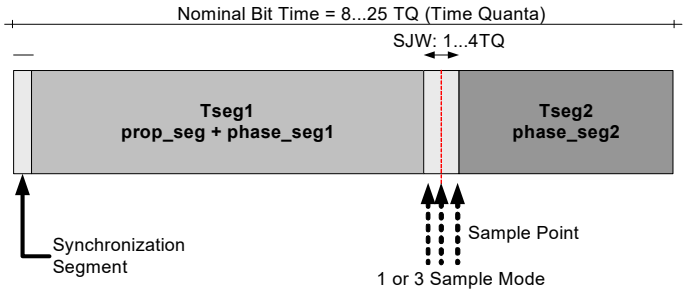
Configurable Bit Rates

Non Configurable Bit Rates

17.8.2 Setting Bit Rate TSEG1 and TSEG2

The bit rate is defined as the number of bits transmitted on a CAN bus per second. Bit time is the reciprocal of bit rate. Bit time is divided into three segments as shown in Figure 17-13. Each segment is represented in terms of fixed units of time called Time Quanta (TQ), which is derived from the system clock (SYSCLK).

Figure 17-13. Bit Time



$$BitTime = (1 + tseg1 + 1 + tseg2 + 1)TQ$$ **Equation 17-1**

$$TQ = \frac{BRP + 1}{SYSCLK}$$ **Equation 17-2**

Note: Bit rate prescaler is a register that performs a prescaling function on SYSCLK to generate the clock for the CAN module. See Figure 17-14.

Synchronization Segment. This is the first segment with one TQ length and is mainly used for synchronization. An edge is expected to fall within this segment.

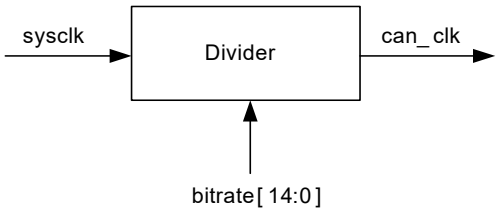
Tseg1, Tseg2. These segments compensate for the edge phase shift errors. The tseg1 also takes in the propagation time, which includes any delays in the network. The length of the segments is increased or decreased to compensate for the error due to phase shift of edges, which is known as resynchronization.

Sample Point. This is the point at which the state of the bus is read and the bit is interpreted. It is located at the end of tseg1.

Synchronization Jump Width. By resynchronization, the tseg1 is lengthened or tseg2 is shortened. Synchronization jump width puts a limit to this resynchronization. The length of tseg2 must be greater than the synchronization jump width.

The Configuration register CAN_CONFIG is used to set the bit rate prescaler (BRP), tseg1, tseg2, and the synchronization jump width. The CAN peripheral clock (CAN_CLK) is generated by dividing the system clock (SYSCLK) by (BRP+1). See the Clocking System chapter on page 72 for detailed information on available options to generate the system clock. For N time quanta in a bit time, the CAN peripheral clock frequency must be configured to N time the CAN bus bit rate.

Figure 17-14. Bit Timing Block Diagram



17.8.2.1 Example

An example to achieve 1 Mbps speed with 40 MHz is described as follows:

The speed is 1 MHz and the bit time is 1 µs.

Choosing a minimum value of 8 TQ in the bit time, 1TQ = 0.125 µs.

BRP = ((time quanta * SYSCLK) – 1) = 4.

Therefore, write a value of '4' into the CFG_BITRATE bits in the configuration register.

Choose the sampling point to be 60 percent of the bit time, which is approximately equal to 5TQ. Because the sampling point is at the end of tseg1, this implies that (tseg2+1) = 3TQ or tseg2 = 2TQ.

To fix the sampling point synchronization jump width, use a value '1' by writing to the bits CFG_SJW = '1'.

Write to the bits cfg_tseg2 a value of '2' to set the value of tseg2 to 2TQ.

Now tseg1 is calculated using the following equation: tseg1 = ((BitTime - (1TQ + tseg2 + 1TQ)) - 1TQ,

which is tseg1 = 3TQ.

Therefore, write a value of '3' into the bits cfg_tseg1 in the configuration register.

This procedure is applied to achieve the standard bit rates using the clock frequencies as specified in Figure 17-12.

Observe the following conditions for setting tseg1 and tseg2:

- tseg1 = 0 or tseg1 = 1 are not allowed.
- tseg2 = 0 is not allowed; tseg2 = 1 is only allowed in direct sampling mode

Note 1: Sampling_mode bit in the Configuration register (CAN_CONFIG) specifies whether one sampling point is used in the receiver path or three sampling points with majority decision are used.

Note 2: Edge_mode bit in the Configuration register (CAN_CONFIG) specifies whether the high to low edge is used for synchronization or both edges are used.

17.9 Errors and Interrupts in CAN

According to the CAN protocol specification, there are five different types of errors. Each CAN node in the bus tries to detect an error, and when it does, it sends out an error frame. The following sections describe the different types of errors and the process of error handling.

17.9.1 Types of Errors

17.9.1.1 BIT Error

A CAN unit sending a bit on the bus also monitors the bus. When the bit value that is monitored is different from the bit value that is sent, a BIT error is detected. An exception is the sending of a 'recessive' bit during the stuffed bit stream of the Arbitration field or during the ACK Slot. A transmitter sending a Passive Error Flag and detecting a 'dominant' bit does not interpret this as a BIT error.

17.9.1.2 FORM Error

A FORM error is detected when there is an error in the CAN message format. The fixed format fields in the message frame such as End of Frame and Interframe Space contain illegal bits.

17.9.1.3 ACKNOWLEDGE Error

A transmitter sending a recessive bit during the ACK slot monitors the ACK slot for a dominant bit. If a receiver receives a message correctly, a dominant bit is written in the ACK slot. Therefore, if the transmitter does not find a dominant bit in the ACK slot after transmission, then an ACKNOWLEDGE error is detected.

17.9.1.4 CRC Error

A transmitting node performs certain calculations to generate a CRC code and transmits it in the CRC field. A receiving node also performs the same calculations to generate a CRC code. If the code generated by the receiver does not match the code transmitted then a CRC error is detected.

17.9.1.5 STUFF Error

When there are six consecutive equal bit levels in a message field that is coded by the message of bit stuffing, a STUFF error is detected during the bit time of the sixth consecutive bit level.

17.9.2 Error Capture Register

PSoC 4200L CAN controller has a dedicated Error Capture Register (ECR) that can be used for CAN bus diagnostics. The error capture register is as shown in Figure 17-15.

Figure 17-15. Error Capture Register

Reserved [31:17]	FIELD [16:12]	BIT [11:6]	TX MODE [5]	RX MODE [4]	ERROR TYPE [3:1]	ECR STATUS [0]

The Error Capture Register has two modes:

Free running mode: In this mode, the ECR captures the field and bit position within the current CAN frame.

Error capture mode: In this mode, the ECR samples the field and bit position when a CAN error is detected. To sample such an event, the ECR needs to be armed by performing a write access to it. When armed, the ECR only captures one error event. For successive error captures, the ECR needs to be armed again by writing a '1' to the error capture register.

Note: The IDE bit is viewed as an arbitration field in ECR.FIELD.

17.9.3 Error States in CAN

CAN has three error states:

- **Error Active.** An error active node can take part in normal bus communication. When it detects an error, it sends out an ERROR ACTIVE FLAG.
- **Error Passive.** An error passive node takes part in bus communication. When it detects an error, it sends out an ERROR PASSIVE FLAG. After sending out the ERROR PASSIVE FLAG, it waits before proceeding with further transmission. An error passive node sends an additional eight recessive bits during the interframe space. This period is also known as suspend transmission because no transmission takes place.
- **Bus Off.** A node that is in this state does not take part in any bus communication. It has no effect on the bus.

The error status in CAN is indicated by the error status register (CAN_ERROR_STATUS). The bits ERROR_STATE (CAN_ERROR_STATUS[17:16]) indicate which error state the CAN node is in. The error states in CAN are determined according to the values of two counters:

- Transmit Error Counter (CAN_ERROR_STATUS[7:0])
- Receive Error Counter (CAN_ERROR_STATUS[15:8])

The error counters are modified according to the CAN 2.0B Specification.

A node is in 'error active' state if the Transmit Error Counter and the Receive Error Counter are less than or equal to 127 decimal. A node is in 'error passive' state if the Transmit or Receive Error Counter value exceeds or equals 128 decimal. A node is in 'Bus Off' state if the Transmit Error Counter exceeds or equals the value of 256 decimal.

An 'error passive' node becomes 'error active' again when both the Transmit Error Count and the Receive Error Count are less than or equal to 127.

A node in 'Bus Off' state becomes 'error active' with its error counters both set to '0' after 128 occurrences of 11 consecutive 'recessive' bits are monitored on the bus.

The error status register has two bits: 'txgte96' (CAN_ERROR_STATUS[18]) and 'rxgte96' (CAN_ERROR_STATUS[19]). These bits indicate if the Transmit Error Counter and Receive Error Counter, respectively, are greater than or equal to 96 decimal. This feature serves as an error warning because an error count value greater than and around 96 indicates a heavily disturbed bus.

17.9.4 Interrupt Sources in CAN

CAN interrupt sources can be classified into two categories: CAN Core Interrupts and TTCAN interrupts. The TT_ENABLE bit in the control register (CAN.CNTL) controls the interrupt routing:

- If TT_ENABLE = 0, the interrupt is routed directly from the CAN core interrupts (INT_STATUS and INT_ENBL)
- If TT_ENABLE = 1, the interrupt is generated from the core interrupts and TTCAN timer interrupts

17.9.4.1 Core Interrupts from CAN

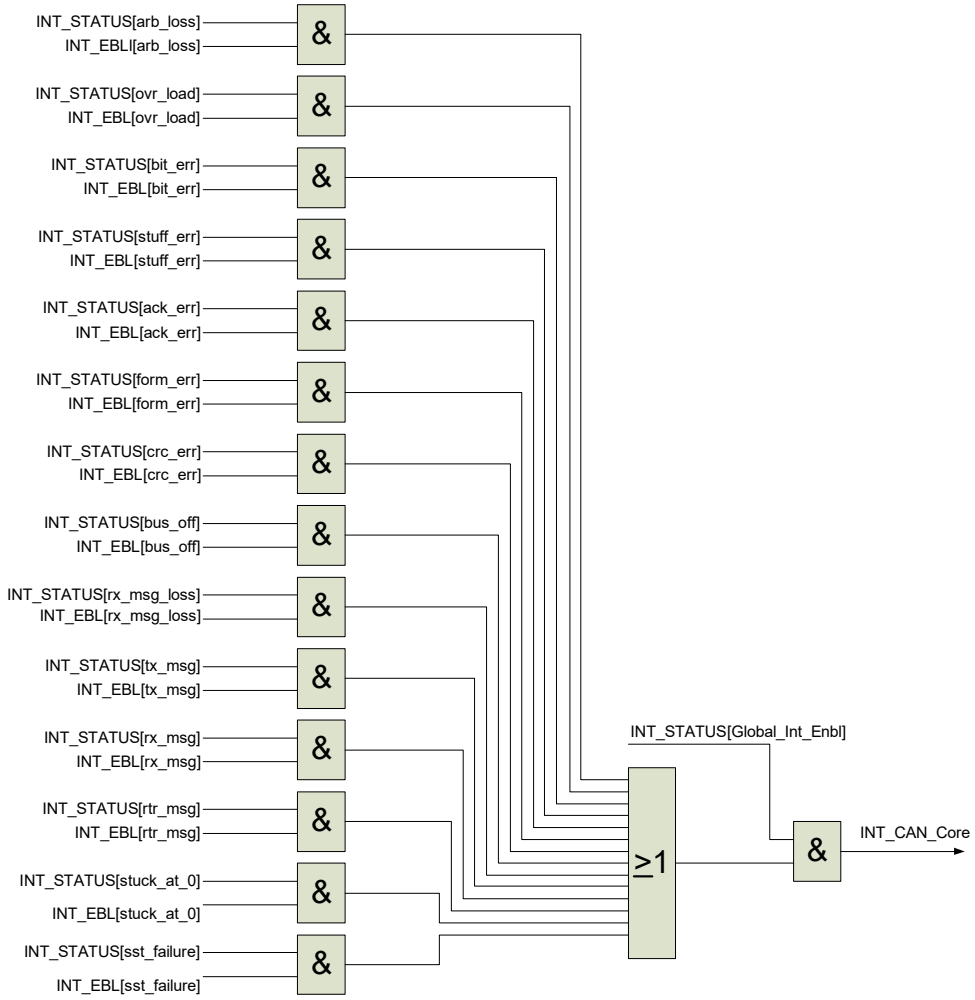
The core interrupts are controlled by an interrupt status (CAN_INT_STATUS) and an interrupt enable register (CAN_ INT_EBL) as shown in Figure 17-16. The interrupt status register stores core interrupt events. When a bit is set, it remains set until it is cleared by writing a '1' to it. The interrupt enable register has no effect on the interrupt status register.

The interrupt enable register (INT_EBL) controls which particular bits from the interrupt status register are used to assert the interrupt output (INT_CAN_Core). INT_CAN_Core is asserted if a particular interrupt status bit and the respective enable bit are set. The INT_CAN_Core routing is controlled by the CAN_CNTL.TT_ENABLE bit. The INT_CAN_Core signal is directly routed to the CAN block interrupt output, if the TT_ENABLE bit is zero. If the TT_ENABLE bit is set (1), the interrupt is routed based on the INTR_CAN_SET register setting, as shown in Figure 17-11.

The various core interrupt sources in CAN are as follows:

- rx_msg. Indicates a message received.
- tx_msg. Indicates a message sent.
- rx_msg_loss. Is set when a new message arrives but the RxMessage flag MSG AV is set and the LINK_FLAG bit is not set. The new message is discarded, because there is no buffer to save it.
- bus_off. The CAN has reached the bus off state.
- crc_err. A CAN CRC error detected.
- form_err. A CAN message format error detected.
- ack_err. A CAN message acknowledge error detected.
- stuff_err. A bit stuffing error detected.
- bit_err. A bit error detected.
- ovr_load. An overload frame received.
- arb_loss. The arbitration lost while sending a message.
- stuck_at_0. Stuck at dominant(0) error detected
- rtr_msg. RTR auto-reply message sent
- sst_failure. Single shot transmission

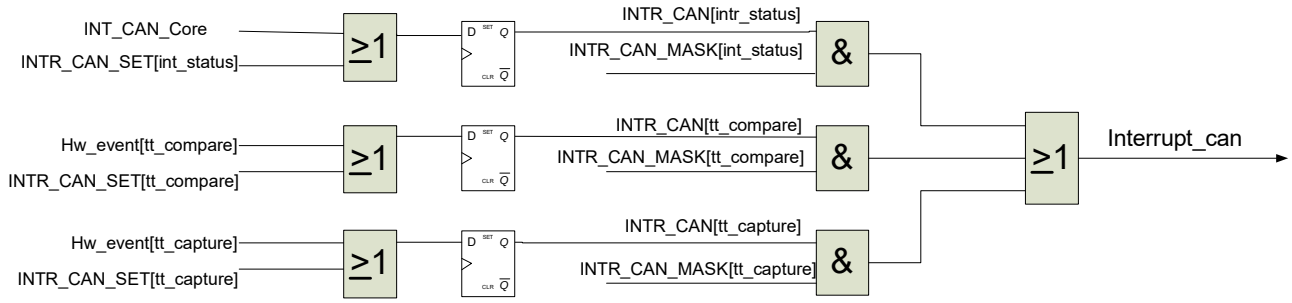
Figure 17-16. Interrupts from CAN Core



17.9.4.2 *Interrupt Routing with TT_ENABLE = 1*

If CNTL.TT_ENABLE bit is set to '1', then the CAN interrupt signal is generated from the CAN core interrupt or additional TTCAN timer hardware interrupts based on the INT_CAN_SET and INT_CAN_MASK register settings as shown in Figure 17-17.

Figure 17-17. CAN Interrupts with CAN_CNTL.TT_ENABLE = 1



The logical AND of INTR_CAN_SET and INTR_CAN_MASK is available in the INTR_CAN_MASKED register. The INTR_-CAN register holds the status of the CAN core interrupt and TTCAN timer compare and capture interrupts, if those are enabled using the INTR_CAN_SET register.

17.10 Operating Modes in CAN

The CAN module operates in three different modes. The command register CAN_COMMAND is used to select the operating modes by setting the corresponding bit for each mode. The three operating modes are as follows:

- Run/Stop Mode: CAN_COMMAND[0]
- Listen Only Mode: CAN_COMMAND[1]
- Loopback Test mode: CAN_COMMAND[2]

17.10.1 Run/Stop Mode

The CAN controller is in Run mode when it is operating normally. The CAN controller can be put into Run mode by setting the CAN_COMMAND[0] bit and stopped by clearing the same bit.

17.10.2 Listen Only Mode

In Listen Only mode, the CAN controller only listens to the CAN receive line without acknowledging the received messages on the bus. It does not send any messages in this mode. However, the error flags are updated so that the bit timing is adjusted until no error occurs.

The steps for automatic baud rate detection are as follows.

1. The CAN controller is initialized for acceptance of all messages (the global/local mask is set to '0').
2. The bit timing values of the first possible CANOpen bit rate (10 Kbps) is loaded and the controller is switched into "Listen Only" mode.
3. Assuming that there is traffic on the network and the bit rate is correct, the message is accepted.
4. The error registers will not change and the flag for message reception is set inside the CAN controller. This means the correct bit rate is detected.
5. Assuming the bit rate is not correct, the error flags are updated (stuff-, CRC, or form-error).
6. The CAN controller is switched off and the next possible bit timing values are loaded from the bit rate table.

17.10.3 Loopback Test Mode

Loopback mode is used for testing purpose. Two types of loopback modes are supported by the PSoC 4200L CAN controller block based on the LOOPBACK bit (CAN_COMMAND[2]) and LISTEN bit (CAN_COMMAND[1]) settings of the command register.

17.10.3.1 External Loopback Mode

External loopback mode is enabled when COMMAND_LOOPBACK = 1 and COMMAND_LISTEN = 0. In this mode, CAN_TX output pin can be connected to the CAN_RX input pin externally; this IP can process receive its transmitted transactions.

17.10.3.2 Internal Loopback Mode

Internal loopback mode is enabled when COMMAND_LOOPBACK = 1 and COMMAND_LISTEN = 1. In this mode, the transmitted transactions are internally routed back to the receiver logic.

18. Universal Serial Bus (USB)

The PSoC® USB block acts as a USB device that communicates with a USB host. The USB block is available as a fixed-function digital block in the PSoC device. It supports full-speed communication (12 Mbps) and is designed to be compliant with the USB Specification Revision.2.0. USB devices can be designed for plug and play applications with the host and also support hot swapping. This chapter details the PSoC USB block and transfer modes. For details about the USB specification, see the USB Implementers Forum web site.

18.1 Features

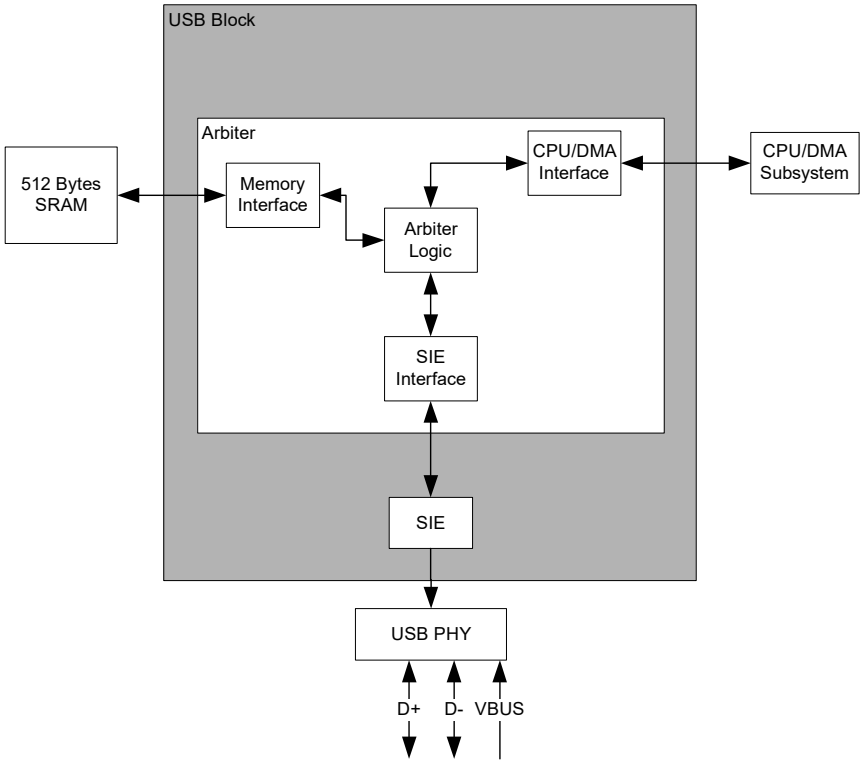
The PSoC USB has these features:

- Complies with USB Specification 2.0
- Supports full-speed peripheral device operation with a signaling bit rate of 12 Mbps
- Supports eight data endpoints and one control endpoint
- Supports four types of transfers – bulk, interrupt, isochronous, and control
- Supports bus- and self-powered configurations
- USB suspend mode for low power
- Supports two types of logical transfer modes:
  - Store and Forward mode
  - Cut Through mode
- Differential signal (D+ and D-) output
- Integrated 22 Ω USB termination resistors on D+ and D– lines, and 1.5-kΩ pull-up resistor on the D+ line
- Battery charger detection (BCD) compliant with USB Battery Charging Specification Rev 1.2 (peripheral detect only)
- Supports maximum packet size of 64 bytes using the Store and Forward mode and maximum packet size of 1023 for isochronous transfer using the Cut Through mode

18.2 Block Diagram

Figure 18-1 illustrates the architecture of the USB block. It consists of the USB Physical Layer (USB PHY), Serial Interface Engine (SIE), Arbiter, and the local 512-byte memory buffer.

Figure 18-1. USB Block Diagram



18.2.1 USB Physical Layer (USB PHY)

This block takes care of physical layer communication with the USB host through the D+, D-, and VBUS pins. It handles the differential mode communication with the host, VBUS detection, and monitoring events such as SE0 on the USB bus.

18.2.2 Serial Interface Engine (SIE)

The Serial Interface Engine (SIE) is responsible for handling the decoding and creating of data and control packets during transmit and receive. It decodes the USB bit streams into USB packets during receive and creates USB bit streams during transmit. The following are the features of the SIE block:

- Conforms to the USB 2.0 Specification
- Supports one device address
- Supports eight data endpoints and control endpoint
- Supports interrupt trigger event for each endpoint
- Integrates an 8-byte buffer in the control endpoint

The registers for this block are mainly used to configure the data endpoint operations and the control endpoint data buffers. This block also controls the interrupt events available for each endpoint.

18.2.3 Arbiter

The Arbiter is the block that handles access of the SRAM memory by the endpoints. The SRAM memory can be accessed by the CPU, DMA, or the SIE. The Arbiter handles the arbitration between the CPU, DMA, and the SIE. The Arbiter consists of the following blocks:

- SIE Interface Module
- CPU/DMA Interface Module
- Memory Interface
- Arbiter Logic

The Arbiter registers are used to handle the endpoint configurations, Read address, and Write address for the endpoints. It also configures the logical transfer type required for each endpoint.

18.2.3.1 SIE Interface Module

This module handles all the transactions with the SIE block. The SIE reads data from the SRAM memory and transmits to the host. Similarly, it writes the data received from the host to the SRAM memory. These requests are registered in the SIE Interface module and are handled by this block.

18.2.3.2 CPU/DMA Interface Block

This module handles all the transactions with the CPU and DMA. The CPU makes requests for the reads and writes to the SRAM memory for each endpoint. These requests are registered in this interface and are handled by the block. When DMA is configured, this interface is responsible for all transactions back and forth between the DMA and USB. The block supports the DMA request line for each data endpoint. The behavior of the DMA depends on the type of logical transfer mode configured in the Configuration register.

18.2.3.3 Memory Interface

The memory interface is used to control the interface between the USB block and the SRAM memory unit. The maximum memory size supported is 512 bytes organized as 256 x a 16-bit memory unit. This is a dedicated memory for the USB. The memory access can be requested by the SIE or by the CPU/DMA. The SIE Interface block and the CPU/DMA Interface block handle these requests.

18.2.3.4 Arbiter Logic

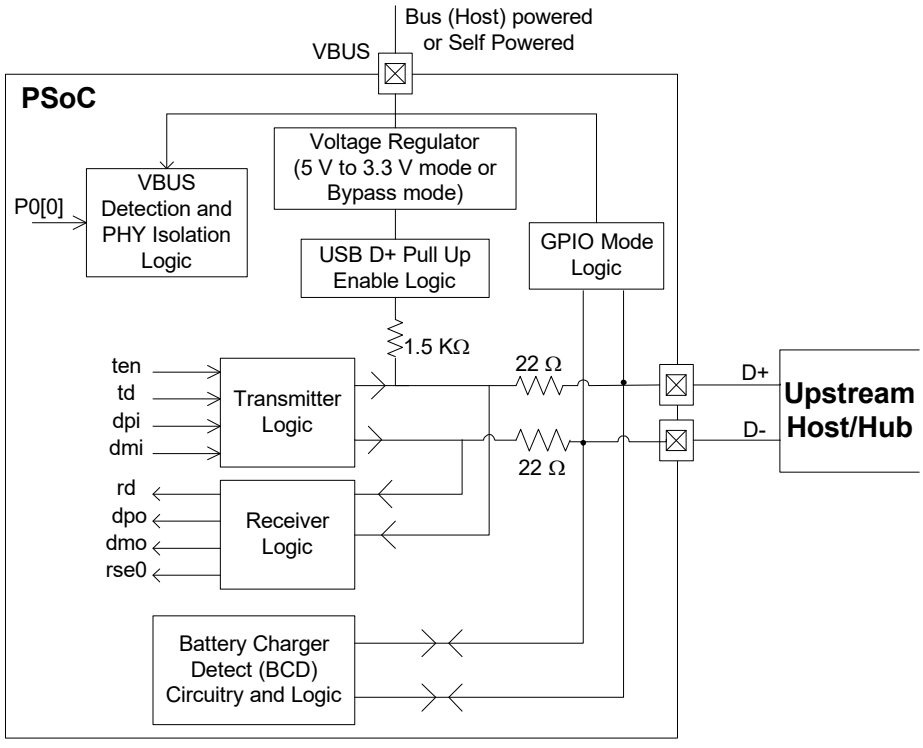
This is the main block of the Arbiter. It is responsible for arbitrations for all the transactions that happen in the Arbiter. It arbitrates the CPU, DMA, and SIE access to the memory unit and the registers. This block also handles the memory management. The memory management is either "Manual" or "Automatic." In Manual memory management, the read and write address manipulations are done by the firmware. In Automatic management, all the memory handling is done by this block itself. This block takes care of the buffer size allocation. It also handles common memory area. This block also handles the interrupt requests for each endpoint.

18.3 How it Works

18.3.1 USB Physical Layer (USB PHY)

The USB block includes the transmitter and the receiver (transceiver), which corresponds to the USB PHY. Figure 18-2 shows the PHY architecture. The USB PHY in PSoC also includes the pull-up resistor on the D+ line to identity the device as full-speed type to the host. The PHY integrates the 22-Ω series termination resistors on the USB lines. The signal between the USB device and the host is a differential signal. The receiver receives the differential signal from the host and converts it to a single-ended signal for processing by the SIE. The transmitter converts the single-ended signal from the SIE to the differential signal, and transmits it to the host. The differential signal is given to the upstream devices at a nominal voltage range of 0 V to 3.3 V.

Figure 18-2. USB PHY Architecture



18.3.1.1 Power Scheme

The USB PHY is powered by the VBUS power pad of the PSoC device. The VBUS pad can be driven either by the host VBUS (bus-powered) or an external power supply (self-powered).

The USB PHY needs a nominal voltage of 3.3 V for its communication with the host. The REG_ENABLE bit in the USB_CR1 register controls the operation of the internal voltage regulator. For applications with VBUS supply voltage in the 5-V range (4.35 V to 5.25 V), the REG_ENABLE bit must be set high to enable the internal regulator. For VBUS supply voltage in the 3.3-V range (3.15 V to 3.45 V), this register bit must be cleared to connect the transceiver directly to the supply.

18.3.1.2 VBUS Detection

USB devices have the option of deriving power from the VBUS power signal from the host (bus-powered configuration) or having an external power supply independent of host VBUS (self-powered configuration). The PSoC USB system has two signals to detect the presence of VBUS.

VBUS Power Pad:

The VBUS power pad pin in PSoC powers the USB PHY and the USB I/Os (D+ and D– pins). The VBUS power pad is designated as P13[2] in the PSoC pin mapping. Note that P13[2] (VBUS power pad) cannot be used as a general-purpose I/O – it can only be used as a power signal for the USB PHY. This pin can either be powered by the host VBUS or a power supply independent of VBUS. An internal comparator in the USB PHY indicates the presence/absence of VBUS as the P13[2] GPIO status signal. This is reflected as bit 2 of the GPIO_PRT13_PS port status register. Additionally, the GPIO_PRT13_INTR_CFG register can be configured to generate PICU interrupt on the rising/falling edge of the P13[2] pin. The GPIO_PRT13_INTR interrupt status register can be read in the "GPIO Interrupt - All Port" interrupt to know if any of the events on P13[2] triggered the interrupt.

P0[0] GPIO Pin:

P0[0] GPIO pin should be used for VBUS detection instead of the VBUS power pad when any of the following conditions are true:

- When the VBUS power pad is supplied by an external power supply independent of host VBUS, P0[0] should be used for host VBUS detection
- When the VBUS from the host is not in the 4.35 V to 5.25 V range, P0[0] should be used for VBUS detection. P0[0] is powered by V_{DDD} and is a GPIO. Refer to the PSoC device datasheet for the GPIO specifications on input voltage threshold levels and maximum input voltage on the GPIO pin. When using P0[0] for VBUS detection, appropriate overvoltage protection circuitry such as a resistive divider should be placed while connecting the host VBUS to P0[0] pin.

Similar to the P13[2] (VBUS pad), the firmware can read the P0[0] pin status and configure the PICU interrupt based on the Port 0 registers.

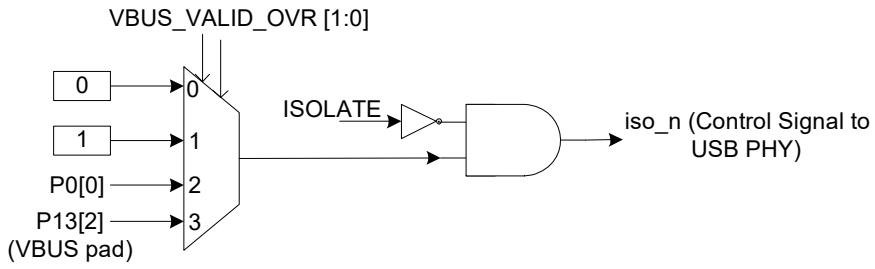
18.3.1.3 USB PHY Isolation Logic

The USB PHY has an isolation logic to disable the PHY output signals when the host VBUS is not present.

Figure 18-3 shows the isolation logic signal (iso_n) generation. iso_n is an active low signal. The device firmware can set the ISOLATE bit field in the USB_POWER_CTRL register to isolate the PHY outputs irrespective of the state of VBUS. For dynamic isolation based on the state of the VBUS, the device firmware should configure the VBUS_VALID_OVR[1:0] bit fields in the USB_POWER_CTRL register to select either the VBUS pad (P13[2]) or the P0[0] GPIO signal depending on which of the two I/Os are used for VBUS detection.

- The device firmware should enable the USB block operation only after the host VBUS is present – detected by reading the pin status register of P13[2] or P0[0].
- When VBUS is present, the device firmware should clear the ISOLATE bit field in the USB_POWER_CTRL register after a delay of at least 2 µs. Thereafter, the signal selected by VBUS_VALID_OVR[1:0] will take care of dynamic PHY output isolation. When the USB block is stopped in the device firmware, the ISOLATE bit field should be set as part of the stop sequence.

Figure 18-3. USB PHY Output Isolation Logic



18.3.1.4 USB D+ Pin Pull-up Enable Logic

When a USB device is self-powered, the USB specification warrants that the device enable the pull-up resistor on its D+ pin to identify itself as a full-speed device to the host. When the host VBUS is removed, the device should disable the pull-up resistor on the D+ line so as to not back power the host. The USB PHY block includes an internal 1.5-kΩ pull-up resistor on the D+ line to indicate to the host that the PSoC is a full-speed device. The pull-up resistor can be enabled or disabled by configuring the USBPUEN bit in the USBIO_CR1 register. Depending on whether the VBUS pad (P13[2]) or the P0[0] pin is used for VBUS detection, the firmware should read that pin status to enable/disable the pull-up resistor on the D+ pin.

18.3.1.5 Transmitter and Receiver Logic

The transmitter logic takes care of differential transmission to the USB host.

The transmitter can be manually forced to transmit signals. The USB_USBIO_CR0 register is used to manually transmit the signals. Examples are as follows:

- When the manual transmission is enabled, the register can be configured to transmit Single-Ended Zero signal (that is, D+ and D– are low).
- Configurable to transmit the USB signals. The USB signals can be two types:
  - D+ low and D– high = J
  - D+ high and D– low = K
- The register also has a bit, which is used to read the received signal levels. The bit can show if D+ < D– or D+ > D–.

18.3.1.6 Battery Charger Detection (BCD)

The USB PHY provides support to detect the type of host downstream port. The details of the detection algorithm and downstream port types are available in Revision 1.2 of the Battery Charging Specification, which can be downloaded from USB Implementers Forum website. The USB PHY can detect the following types of downstream ports – standard downstream port (SDP), charging downstream port (CDP), and dedicated charging port (DCP). The sequence of steps to detect the port type are as follows.

1. VBUS Detection: The device is required to detect VBUS assertion as the first step of the BCD algorithm. This should be implemented as an interrupt event to the CPU using rising/falling edge signal on the VBUS monitoring pin. The VBUS monitoring pin can either be the VBUS power pad or P0[0] pin as discussed in 18.3.1.2 VBUS Detection on page 210.
2. Configuration of the USB IP for BCD: The next step is to ensure the USB IP is properly configured to perform BCD. This includes the following:
   a. All the DMA channels associated with USB endpoints and the three USB interrupt vectors need to be disabled.
   b. The USB_ENABLE bit in the USBDEVv2_CR0 register should be cleared. The pull-up resistor on the D+ line should be disabled by clearing the USBPUEN bit in USBDEVv2_USBIO_CR1 register.
   c. The CSR_CLK_EN bit in USBDEVv2_USB_CLK_EN should be set to enable the clocking for the block. The D+ and D– pins should be configured for bit-banged mode by clearing the IOMODE bit in the USBDEVv2_USBIO_CR1 register.
   d. Configure the USBDEVv2_USB_POWER_CTRL register as follows: Disable the pull-down resistors on the VBUS power pad and D– pin. Configure the VBUS_VALID_OVR[1:0] bits to generate the vbus_valid signal from either the VBUS power pad (PHY detector) or the P0[0] GPIO pin depending on the pin used for VBUS monitoring. If VBUS monitoring is not used in the design, configure the VBUS_VALID_OVR[1:0] bits to force vbus_valid signal to always be 1. Set the SUSPEND, SUSPEND_DEL, ENABLE_RCVR, ENABLE_DPO, ENABLE_DMO, ENABLE_CHGDET, and ENABLE bits.
   e. Clear the SUSPEND bit in the USBDEVv2_USB_POWER_CTRL register; after a 2 µs delay, clear the SUSPEND_DEL bit.
3. Data line Contact Detect (DCD): The USB device is required to detect data line contact before performing charger detection in case there is a time lag between VBUS contact and data line contact. This time lag is accounted for by introducing a hardcoded delay of 400 ms before proceeding to the next step of the BCD algorithm.
4. Primary Detection: Primary detection is used to differentiate between SDP and DCP/CDP ports. To perform primary detection, set only the following bits in the USBDEVv2_USB_CHGDET_CTRL register: REF_EN, REF_DP, COMP_EN, and COMP_DM. The remaining register bits should be cleared. After a delay of 40 ms, the status of the D– line should be monitored by reading the DMO bit in the USBDEVv2_USBIO_CR1 register. The comparator output is indicated by the COMP_OUT bit in the USBDEVv2_USB_CHGDET_CTRL register. The following table specifies the port type for the different combinations of these two signals.

Comparator Output	D– State	Port Type
0	0	SDP
0	1	Not a possible combination. Abort charger detection.
1	0	DCP or CDP. Need to perform secondary detection.
1	1	May be connected to a proprietary charger, where D+ and D– are pulled up. CPU should also read D+ status to confirm.

5. Secondary Detection: Secondary detection is used to differentiate between CDP and DCP ports. To perform secondary detection, set only the following bits in the USBDEVv2_USB_CHGDET_CTRL register: REF_EN, REF_DM, COMP_EN, and COMP_DP. The remaining register bits should be cleared. After a delay of 40 ms, the status of the D+ line should be monitored by reading the DPO bit in the USBDEVv2_USBIO_CR1 register. The comparator output is indicated by the COMP_OUT bit in the USBDEVv2_USB_CHGDET_CTRL register. The following table specifies the port type for the different combinations of these two signals.

Comparator Output	D+ State	Port Type
0	0	CDP
0	1	Not a possible combination. Abort charger detection.
1	0	DCP
1	1	May be connected to a proprietary charger, where D+ and D– are pulled up. CPU should also read D– status to confirm.

6. Restoring Register Values: After the port type detection, all the bits in the USBDEVv2_USB_CHGDET_CTRL register should be cleared. The ENABLE_CHGDET bit in the USBDEVv2_USB_POWER_CTRL register should be cleared. The USBDEVv2_USB_POWER_CTRL register should be restored to the value it had before the start of the BCD algorithm. The CSR_CLK_EN bit in USBDEVv2_USB_CLK_EN should be cleared. The IOMODE bit in the USBDEVv2_USBIO_CR1 register should be set. The USB device can undergo the enumeration process if the port type detected is SDP or CDP.

18.3.1.7 Link Power Management (LPM)

The USB PHY supports Link Power Management (LPM), which is similar to the suspend mode, but has transitional latencies in tens of microseconds between power states compared to the greater than 20 ms associated with suspend/resume modes. More details on LPM can be found in the USB 2.0 specification (LPM ECN document and LPM errata document). The following features are supported for LPM.

- The USBDEVv2_USB_LPM_CTRL register should be configured to enable/disable LPM, type of response when LPM is enabled, and response when a sub PID other than the LPM token is received from the host.
- The USBDEVv2_USB_LPM_STAT register stores the values of the Best Effort Service Latency (BESL) and the remote wakeup feature as sent by the host. The firmware should read this register on the LPM interrupt event and enter the appropriate low-power mode (Deep-Sleep or Sleep) based on the BESL value from the host.

18.3.2 Endpoints

The SIE and arbiter support eight data endpoints (EP1 to EP8) and one control endpoint (EP0). The data endpoints share the SRAM memory area of 512 bytes. The endpoint memory management can be either manual or automatic. The endpoints are configured for direction and other configuration using the SIE and arbiter registers. The endpoint read address and write address registers are accessed through the arbiter.

The endpoints can be individually made active. In the Auto Management mode, the register USB_EP_ACTIVE is written to control the active state of the endpoint. The endpoint activation cannot be dynamically changed during runtime. In Manual Memory Management mode, the firmware decides the memory allocation, so it is not required to specify the active endpoints. The USB_EP_ACTIVE register is ignored during the manual memory management mode. The USB_EP_TYPE register is used to control the transfer direction (IN, OUT) for the endpoints. The control endpoint has separate eight bytes for its data (USB_EP0_DR registers).

18.3.3 Transfer Types

The PSoC USB supports full-speed transfers and is compliant with the USB 2.0 Specification. It supports four types of transfers:

- Interrupt Transfer
- Bulk Transfer
- Isochronous Transfer
- Control Transfer

For further details about these transfers, refer to the USB Specification 2.0.

18.3.4 Interrupts

The USB block has three general-purpose interrupt lines – INTR_LO, INTR_MED, and INTR_HI. Refer to the Interrupts chapter on page 45 for the vector numbers of these interrupt lines. The USB block has a predefined set of 13 interrupt trigger events that can be mapped to either one of the three interrupts. The interrupt line to which each of the trigger event is mapped is configured through the USB_INTR_LVL_SEL register. Each of these three interrupt lines has a status register to identify the event that cause of the interrupt. These are the USB_INTR_CAUSE_LO, USB_INTR_CAUSE_MED, and USB_INTR_CAUSE_HI status registers.

The following 12 events can be used to generate an interrupt on one of the three interrupt lines.

- USB Start of Frame (SOF) Event
- USB Bus Reset Event
- Eight Data Endpoint (EP1 – EP8) interrupt events
- Control Endpoint (EP0) interrupt event
- Arbiter Interrupt event
- Link Power Management (LPM) event

The following sections provide details on each of these interrupt events.

18.3.4.1 Data Endpoint Interrupt Events

These are eight interrupt events corresponding to each data endpoint (EP1-EP8). Each of the endpoint interrupt events can be enabled/disabled by using the corresponding bit in the USB_SIE_EP_INT_EN register. The interrupt status of each endpoint can be known by reading the USB_SIE_EP_INT_SR status register. An endpoint whose interrupt is enabled can trigger the interrupt on the following events:

- Successful completion of an IN/OUT transfer
- NAK-ed IN/OUT transaction if the corresponding NAK_INT_EN bit in the SIE_EPx_CR0 register is set
- When there is an error in the transaction, the bit ERR_IN_TXN in the SIE_EPx_CR0 register is set and interrupt is generated. Refer to the register description for details on the conditions under which this bit can be set.
- If the STALL bit in SIE_EPx_CR0 is set, then stall events can generate interrupts. This stall event can happen if an OUT packet is received for an endpoint whose mode bits in SIE_EPx_CR0 are set to ACK_OUT or if an IN packet is received with mode bits set to ACK_IN.

18.3.4.2 Control Endpoint (EP0) Interrupt Event

The interrupt event corresponding to the control endpoint (EP0) is generated under the following events:

- Successful completion of an IN/OUT transfer
- When a SETUP packet is received on the control endpoint

18.3.4.3 Arbiter Interrupt Event

The arbiter generates an interrupt event for the endpoints during the following events – the final arbiter interrupt event is the logical OR of these events.

DMA Grant:

This event is applicable in any of the DMA modes – Store and Forward DMA mode (Mode 2) or Cut Through mode (Mode 3).This event is triggered when the DMA controller pulses the "Burstend" signal corresponding to that endpoint, for which a DMA request had been raised to the DMA controller earlier. The request could have been either via a Manual DMA Request or an automatic Arbiter DMA Request. A common grant status exists for both modes of requests. This grant status indicates completion of the DMA transaction. This status indication can be used by firmware to determine when the next Manual DMA Request can be raised. Multiple requests raised for the same endpoint before the DMA Grant status is set will be dropped by the block. Only the first of multiple requests will be transmitted to DMA controller.

IN endpoint local buffer full:

This event status is set on different conditions in Store and Forward modes (Modes 1 and 2) and the Cut Through Mode (Mode 3).

- Store and Forward Mode: This status is set when the entire packet data has been transferred to the local memory. The check is that data written for the particular endpoint is equal to the programmed Byte Count for that endpoint in the USB.SIE_EPx.CNT0 and USB.SIE_EPx.CNT1 registers.
- Cut Through Mode: In this mode, the IN buffer full status is set when the IN endpoint's dedicated buffer is filled with the packet data. The size of this buffer is determined by the value programmed in bits [3:0] of the USB_BUF_SIZE register. This status indication can be used to determine when the Mode value in the USB_SIE_EPx_CR0 register can be programmed to acknowledge an IN token for that endpoint.

Buffer Overflow:

This event is applicable only in the Cut Through Mode. This can occur due to buffer overflow either in the dedicated buffer space or the common buffer space. The buffer overflow conditions are explained here.

- In the case of an IN endpoint, the dedicated buffer can overflow if the DMA transfer writes a larger number of bytes than the space available in the dedicated buffer. Until an IN token has been received for that endpoint, it cannot use the common buffer area, hence resulting in an overflow of data. The possible causes of this buffer overflow can be incorrect programming of either the DMA transfer descriptor transfer size or the USB_BUF_SIZE register.
- In the case of an OUT endpoint, the dedicated buffer can overflow if two OUT transactions occur back to back. The data from the previous transaction is still present in the common area and the current ongoing transaction fills up the OUT endpoint's dedicated buffer space and overflows. The possible causes of this overflow can be the overall DMA bandwidth constraint due to other DMA transactions or reduced size of the dedicated OUT buffer size.
- In the case of an IN endpoint, the common buffer can overflow if the DMA transfer writes a larger number of bytes than the space available in the common area. The possible causes of this buffer overflow can be incorrect programming of either the DMA transfer descriptor transfer size or the USB_DMA_THRES and USB_DMA_THRES_MSB registers.
- In the case of an OUT endpoint, the common buffer area can overflow if the data written in to the common area has not yet been read and the new data begins to overwrite the existing data. The possible causes of this overflow can be the overall DMA bandwidth constraint due to other DMA transactions or the lower OUT endpoint DMA priority.

Buffer Underflow:

This event is applicable only in the Cut Through Mode. This

underflow condition can occur only for an IN endpoint. The underflow condition can occur either in the dedicated buffer space or common buffer space. The underflow condition on the dedicated buffer space can either be due to the reduced dedicated IN buffer size or DMA bandwidth constraint. The underflow condition can occur on the common buffer space due to DMA bandwidth constraint and/or lower DMA channel priority.

Setting up and Processing the Arbiter Interrupt Event: There is only one arbiter interrupt event common for all the endpoints – this is the logical OR of the above listed events for each of the endpoints. The master enable for arbiter interrupt event for each endpoint can be configured using the USB_ARB_INT_EN register. The status register ARB_INT_SR can be used to identify the endpoints that caused the arbiter interrupt event to trigger.

The register USB_ARB_EPx_INT_EN (where x = 1 to 8 for each endpoint) is used to enable or disable the arbiter interrupt events for each endpoint. The status of each of these sub events can be read using the USB_ARB_EPx_INT_SR register.

18.3.4.4 USB Start of Frame (SOF) Event

- Generated whenever the SOF is received. The SOF interrupt is enabled by setting the SOF_INTR_MASK bit in the USB_INTR_SIE_MASK register. The SOF interrupt status is reflected in the SOF_INTR status bit in the USB_INTR_SIE status register.
- The SOF interrupt status is also available in the SOF_INTR_MASKED bit of the USB_INTR_SIE_MASKED register – this bit is the logical AND of corresponding SOF bits in the USB_INTR_SIE_MASK register and the USB_INTR_SIE register.
- The USB packet decoder in the SIE decodes the SOF PID and asserts a pulse of width one 12-MHz clock cycle. This pulse is synchronized to the system clock (SYSCLK) and can be routed as a digital signal through the Digital System Interconnect (DSI) interface. This routing feature is in addition to the interrupt feature.

18.3.4.5 USB Bus Reset Event

- Generated whenever a USB bus reset condition occurs. The bus reset interrupt is enabled by setting the BUS_RESET_INTR_MASK bit in the USB_INTR_SIE_MASK register. The bus reset interrupt status is reflected in the BUS_RESET_INTR status bit in the USB_INTR_SIE status register.
- The bus reset interrupt status is also available in the BUS_RESET_INTR_MASKED bit of the USB_INTR_SIE_MASKED register – this bit is the logical AND of corresponding bus reset bits in the USB_INTR_SIE_MASK register and the USB_INTR_SIE register.
- The 32-kHz low-frequency clock (LFCLK) is used to detect the USB bus reset condition. It is required that the LFCLK is enabled in the application firmware. The SIE logic triggers on counter running on LFCLK when a SE0 condition is detected on the USB bus. When the counter reaches the count value configured in the USB_BUS_RST_CNT register, the bus reset interrupt is triggered. It is recommended to set the count value in the USB_BUS_RST_CNT register to three to detect the bus reset condition.

18.3.4.6 Link Power Management (LPM) Event

Generated whenever the LPM token packet is received. The LPM interrupt is enabled by setting the LPM_INTR_MASK bit in the USB_INTR_SIE_MASK register. The LPM interrupt status is reflected in the LPM_INTR status bit in the USB_INTR_SIE status register.

The LPM interrupt status is also available in the LPM_INTR_MASKED bit of the USB_INTR_SIE_MASKED register; this bit is the logical AND of corresponding LPM bits in the USB_INTR_SIE_MASK and USB_INTR_SIE registers.

The firmware needs to read the USBDEVv2_USB_LPM_STAT register to read the BESL, remote wakeup values and appropriately enter the desired low-power mode. It is recommended to enter the low-power mode in the main code. The exit from LPM is identical to the resume event wakeup in the case of suspend mode.

18.3.5 DMA Support

Each of the eight data endpoints has one DMA channel available to transfer data between the endpoint buffer and the SRAM memory. The USB IP generates the DMA request signals (usb.dma_req[7:0]) to the respective DMA channels to initiate the data transfer for the endpoint. This goes to the Trigger Group 1 multiplexer as input triggers that can be routed to one of the trigger inputs of DMA channels 8-15. Details on Trigger Group 1 can be referred to in the DMA Controller Modes chapter on page 31. The transfer complete signals from the endpoint DMA channels (cpuss.dmac_tr_out[8:15]) are routed to the USB IP as input signals usb.dma_burstend[0:7]. The Trigger Group 3 multiplexer is used to route these signals. The eight output signals of these trigger multiplexers are usb.dma_burstend[0:7]. Each of these outputs can be routed from nine input signals as given in Table 18-1. Each output can be selected by configuring the PERI_TR_GROUP3_TR_OUT_CTRLx register where x can be between 0 and 7.

Table 18-1. Input Sources for Trigger Group 3 Multiplexer Outputs (usb.dma_burstend[0:7])

Input Source	Description
0	Software trigger - hardwired to logic zero
1	cpuss.dmac_tr_out[8]
2	cpuss.dmac_tr_out[9]
3	cpuss.dmac_tr_out[10]
4	cpuss.dmac_tr_out[11]

Table 18-1. Input Sources for Trigger Group 3 Multiplexer Outputs (usb.dma_burstend[0:7])

Input Source	Description
5	cpuss.dmac_tr_out[12]
6	cpuss.dmac_tr_out[13]
7	cpuss.dmac_tr_out[14]
8	cpuss.dmac_tr_out[15]

18.4 Logical Transfer Modes

The USB block in PSoC devices supports two types of logical transfers. The logical transfers can be configured using the register setting for each endpoint. Any of the logical transfer methods can be adapted to support the three types of data transfers (Interrupt, Bulk, and Isochronous) mentioned in the USB 2.0 Specification. The Control transfer is mandatory in any USB device.

The logical transfer mode is a combination of memory management and DMA configurations. The Logical Transfer modes are related to the data transfer within the USB block (to and from the SRAM memory unit for each endpoint). It does not represent the transfer methods between the device and the host (the transfer types specified in the USB 2.0 Specification).

The USB block supports two basic types of transfer modes, which are detailed in Table 18-2.

- Store and Forward mode
- Cut Through mode

Table 18-2. USB Transfer Modes

Feature	Store and Forward Mode	Cut Through Mode
SRAM Memory Usage	Requires more memory	Requires less memory
SRAM Memory Management	Manual	Auto
SRAM Memory Sharing	512 bytes of SRAM shared between endpoints. Sharing is done by firmware.	Each endpoint is allocated less share of memory automatically by the block. Rest of memory is available as “Common Area.” This Common Area is used during the transfer.
IN Command	Entire packet present in SRAM memory before the IN command is received.	Memory filled with data only when SRAM IN command is received. Data is given to host when enough data is available (based on DMA configuration). Does not wait for the entire data to be filled.
OUT Command	Entire packet is written to SRAM memory on OUT command. After entire data is available, it is copied from SRAM memory to the USB device.	Waits only for enough bytes (depends on DMA configuration) to be written in SRAM memory. When enough bytes are present, it is immediately copied from SRAM memory to the USB device.
Transfer of Data	Data is transferred when all bytes are written to the memory.	Data is transferred when enough bytes are available. It does not wait for the entire data to be filled.
Types Based on DMA	No DMA mode Manual DMA mode	Only Auto DMA mode
Supported Transfer Types	Best suited for Interrupt and Bulk transfers	Best suited for Isochronous transfer

Every endpoint has a set of registers that need to be handled during the modes of operation, as detailed in Table 18-3.

Table 18-3. Endpoint Registers

Register	Comment	Content	Usage
ARB_RWx_WA	Endpoint Write Address register	Address of the SRAM	This register indicates the SRAM location to which the data in the Data register is to be written.
ARB_RWx_RA	Endpoint Read Address register	Address of the SRAM	This register indicates the SRAM location from which the data must be read and stored to the Data register.
ARB_RWx_DR	Endpoint Data Register	8-Bit Data	Data register is read/ written to perform any transaction. IN command: Data written to the Data register is copied to the SRAM location specified by the WA register. After write, the WA value is automatically incremented to point to the next memory location. OUT command: Data available in the SRAM location pointed by the RA register is read and stored to the DR. When the DR is read, the value of RA is automatically incremented to point to the next SRAM memory location that must be read.
SIE_EPx_CNT0 and SIE_EPx-_CNT1	Endpoint Byte Count Register	Number of Bytes	Holds the number of bytes that can be transferred. IN command: Holds the number of bytes to be transferred to host. OUT command: Holds the maximum number of bytes that can be received. The firmware programs the maximum number of bytes that can be received for that endpoint. The SIE updates the register with the number of bytes received for the endpoint.
“Mode” bits in SIE_EPx_CR0	Mode Values	Response to the Host	Controls how the USB device responds to the USB traffic and the USB host. Some examples of mode include ACK, NAK, STALL, etc. See Table 18-2 on page 215 for additional details.

In Manual Memory Management, the endpoint read and endpoint write address registers are updated by the firmware. So the memory allocation can be done as required by the user and the memory allocation decides which endpoints are active; that is, the user can decide to share the 512 bytes for all the eight endpoints or a lesser number of endpoints.

In Automatic memory management, the endpoint read and endpoint write address registers are updated by the USB block. The block assigns memory to the endpoints that are activated using the USB_EP_ACTIVE register. The size of memory allocated depends on the value in the USB_BUF_SIZE register. The rest of the memory, after allocation, is called the "Common Area" memory and used for the transfer of data.

In all of these modes, either the 8-bit endpoint data register or the 16-bit endpoint data register can be used to read/write to the endpoint buffer. While transferring data to the 16-bit data registers, it must be ensured that the corresponding SRAM memory address locations are also 16-bit aligned.

In the following text, the algorithm for the IN and OUT transaction for each mode is discussed. An IN transaction is when the data is read by the USB host (for example, PC). An OUT transaction is when the data is written by the USB host to the USB device. The choice of using the DMA and memory management can be configured using the USB_ARB_CFG register and the mode is common to all endpoints.

18.4.1 Store and Forward Mode

18.4.1.1 No DMA Access

This is the Manual Memory Management mode with no DMA access.

IN Transaction (CPU Write, SIE Read): The steps for an IN transaction on an IN endpoint are shown in Figure 18-4. OUT Transaction (CPU Read, SIE Write): The steps for an OUT transaction on an OUT endpoint are shown in Figure 18-5.

Figure 18-4. No DMA Access IN Transaction

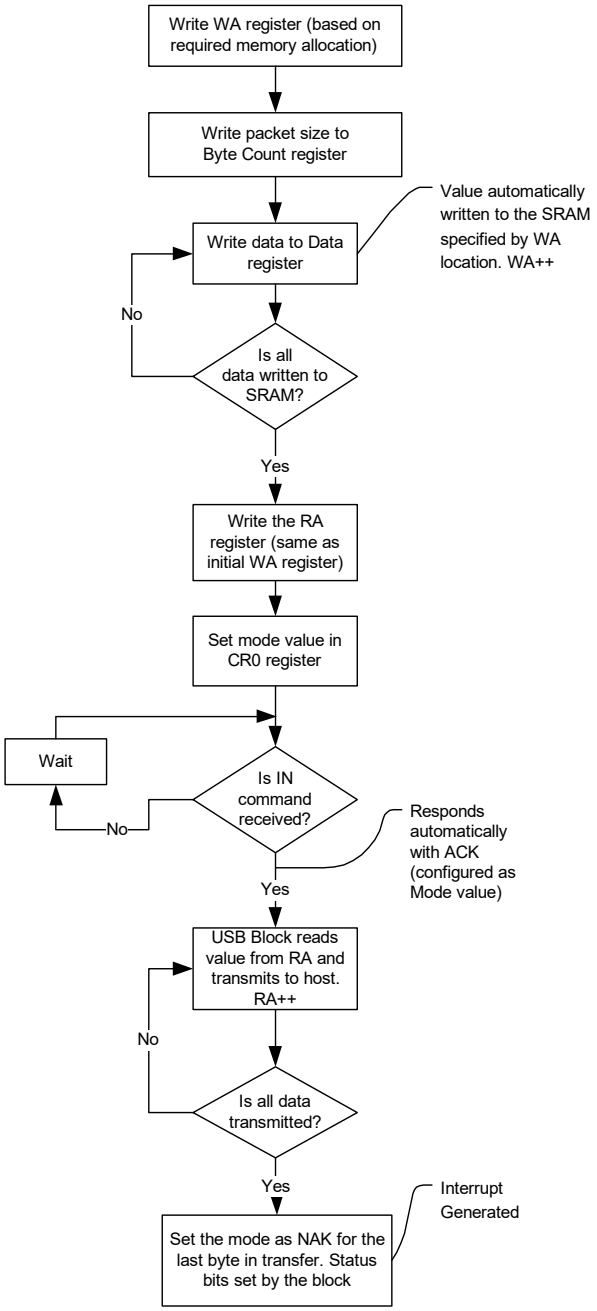
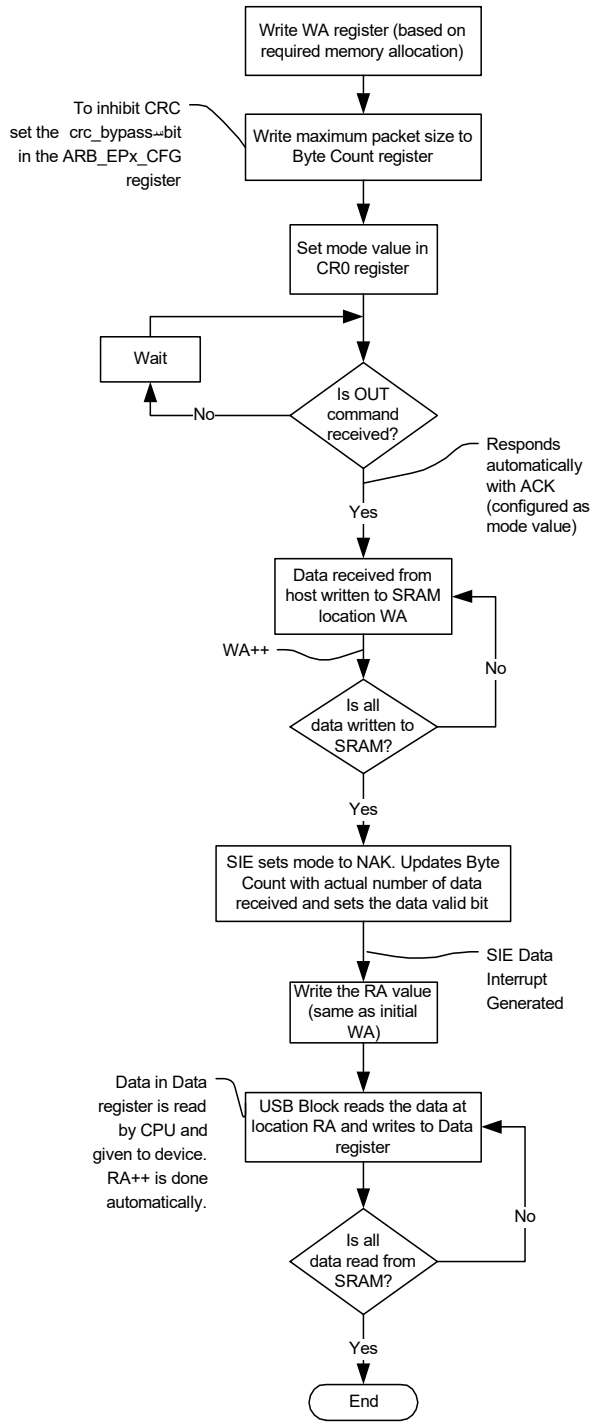


Figure 18-5. No DMA Access OUT Transaction

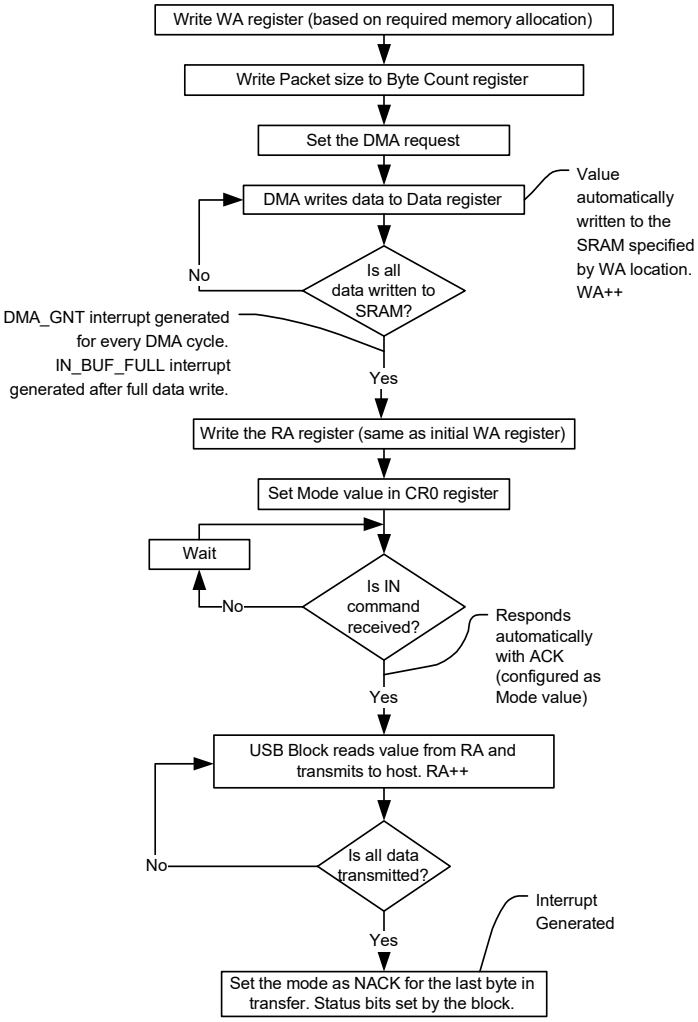


18.4.1.2 Manual DMA Access

This is the Manual Memory Management mode with Manual DMA Access. This mode requires the configuration of the DMA controller. For information on DMA configuration, refer to the DMA Controller Modes chapter on page 31. This mode is similar to the No DMA Access except that the write/read of packets is performed by DMA. A DMA request for an endpoint is generated by setting the DMA_CFG bit in the USB_ARB_EPx_CFG register. When the DMA service is granted and is done (DMA_GNT), an arbiter interrupt can be programmed to occur. The transfer is done using a single DMA cycle or multiple DMA cycles. After completion of every DMA cycle the arbiter interrupt (DMA_GNT) is generated. Similarly, when all the bytes of data (programmed in the byte count) are written to the memory, the arbiter interrupt occurs and the IN_BUF_FULL bit is set.

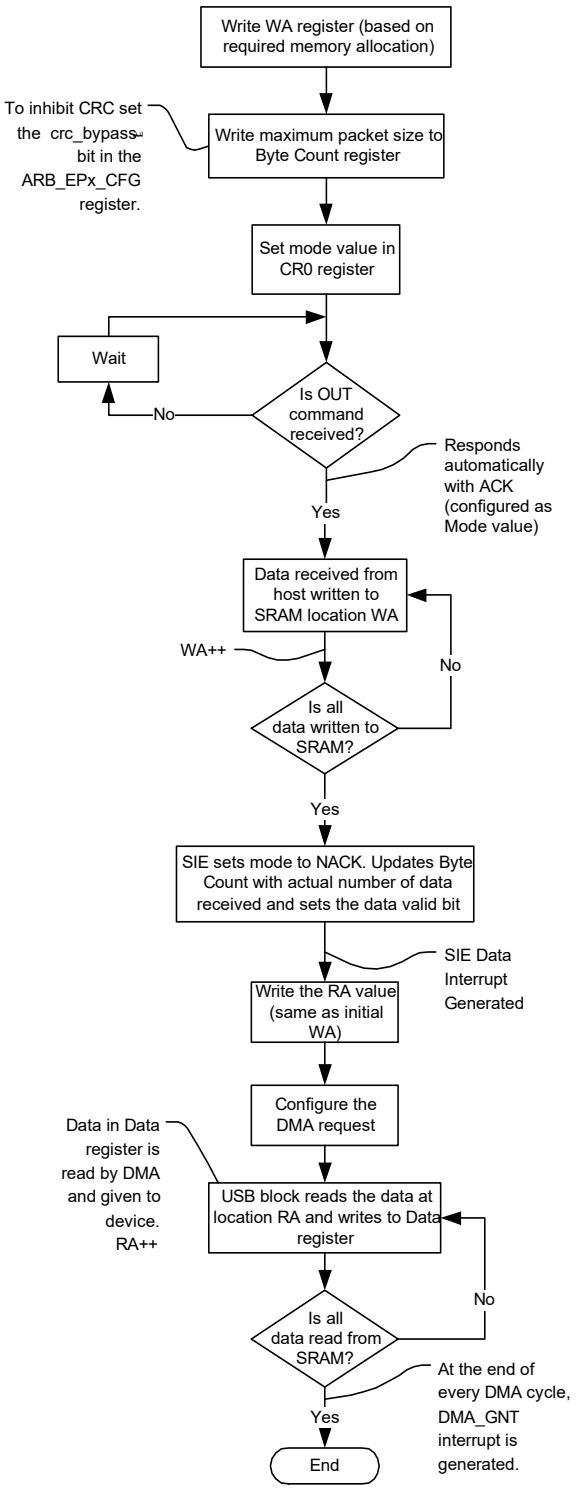
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 18-6.

Figure 18-6. Manual DMA IN Transaction



OUT Transaction (CPU Read, SIE Write): The steps for an OUT transaction on an OUT endpoint are shown in Figure 18-7.

Figure 18-7. Manual DMA OUT Transaction



18.4.2 Cut Through Mode

This is the Auto Memory Management mode with Auto DMA Access. The CPU programs the initial buffer size requirement for IN/OUT packets and informs the Arbiter block of the endpoint configuration details for the particular application being considered. The block then controls memory partitioning and handling of all memory pointers. During memory allocation, each active IN endpoint (set by the USB_EP_ACTIVE and USB_EP_TYPE registers) is allocated a small amount of memory configured using the USB_BUF_SIZE register (32 bytes for each of the eight endpoints). The remaining memory (256 bytes) is left as "Common Area" and is common for all endpoints.

In this mode, the memory requirement is less and it is suitable for the full-speed isochronous transfer up to 1023 bytes.

When an IN command is sent by the host, the device responds with the data present in the dedicated memory area for that endpoint. It simultaneously issues a DMA request for more data for that EP. This data fills up in the common area. The device does not wait for the entire packet of data to be available. It only waits for the (USB_DMA_THRES_MSB, USB_DMA_THRES) number of data available in the SRAM memory and begins the transfer from the common area.

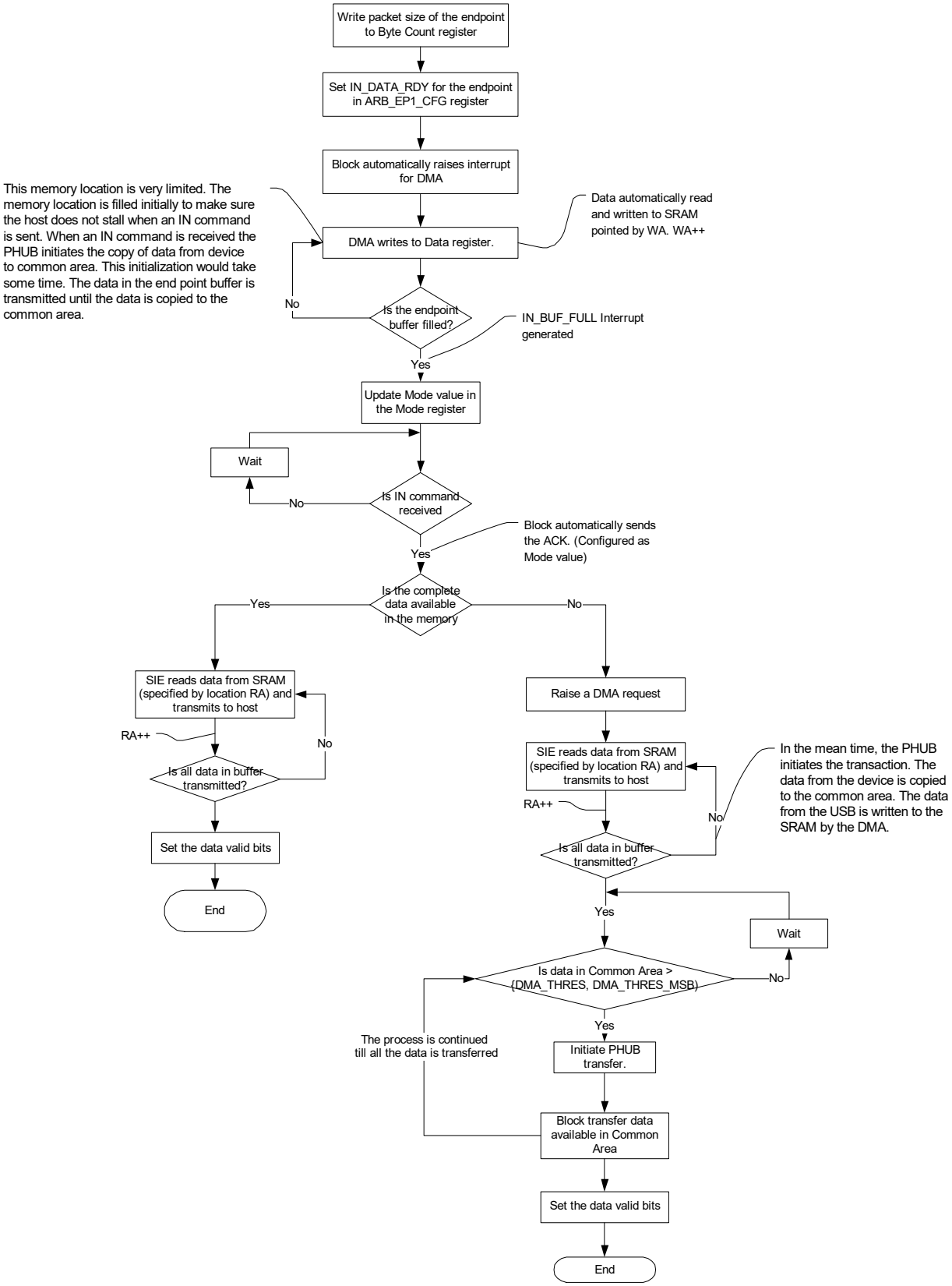
Similarly, when an OUT command is received, the data for the OUT endpoint is written to the common area. When some data (greater than USB_DMA_THRES_MSB, USB_DMA_THRES) is available in the common area, the Arbiter block initiates a DMA request and the data is immediately written to the device. The device does not wait for the common area to be filled.

This mode requires the configuration of the USB_DMA_THRES and USB_DMA_THRES_MSB registers to hold the number of bytes that can be transferred in one DMA transfer (32 bytes). Similarly, the burst count of the DMA should always be equal to the value set in the USB_DMA_THRES registers. Apart from the DMA configuration, this mode also needs the configuration of the USB_BUF_SIZE for the IN and the OUT buffers and the USB_EP_ACTIVE and the USB_EP_TYPE registers.

Each DMA channel has two descriptors and both of them are used in this mode. Each descriptor is considered as a data chunk of 32 bytes and it executes according to the trigger mechanism. The descriptors are chained and hence 64 bytes can be transferred without firmware interaction. When both descriptors complete, the endpoint DMA done interrupt and the DMA error interrupt triggers (due to the lack of data to transfer). The descriptors are updated to advance the source SRAM (IN endpoint) or destination SRAM (OUT endpoint) pointer locations and then enabled again. The firmware also triggers the DMA transfer in software. This sequence continues till all data is transferred. See the DMA Controller Modes chapter on page 31 for details on DMA configuration.

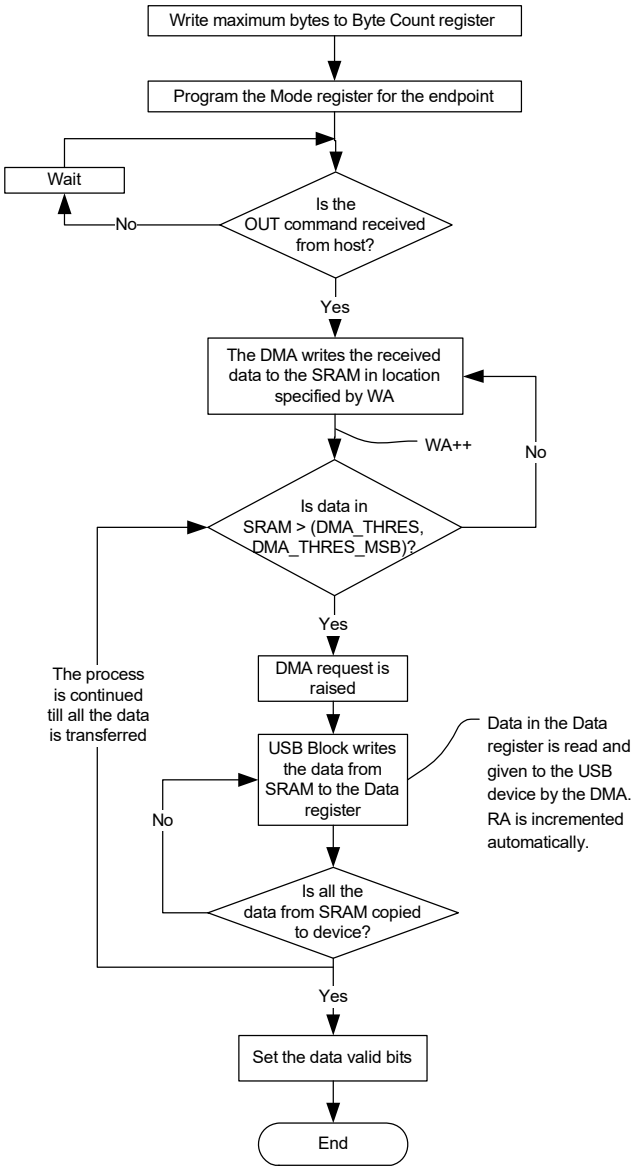
IN Transaction (CPU Write, SIE Read): The steps for an IN transaction on an IN endpoint are shown in Figure 18-8.

Figure 18-8. Cut Through Mode IN Transaction



OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in Figure 18-9.

Figure 18-9. Cut Through Mode OUT Transaction



18.4.3 Control Endpoint Logical Transfer

The control endpoint has a special logical transfer mode. It does not share the 512 bytes of memory. Instead, it has a dedicated 8-byte register buffer (USB_EP0_DRx registers). The IN and OUT transaction for the control endpoint is detailed in the following figures.

Figure 18-10. Control Endpoint IN Transaction

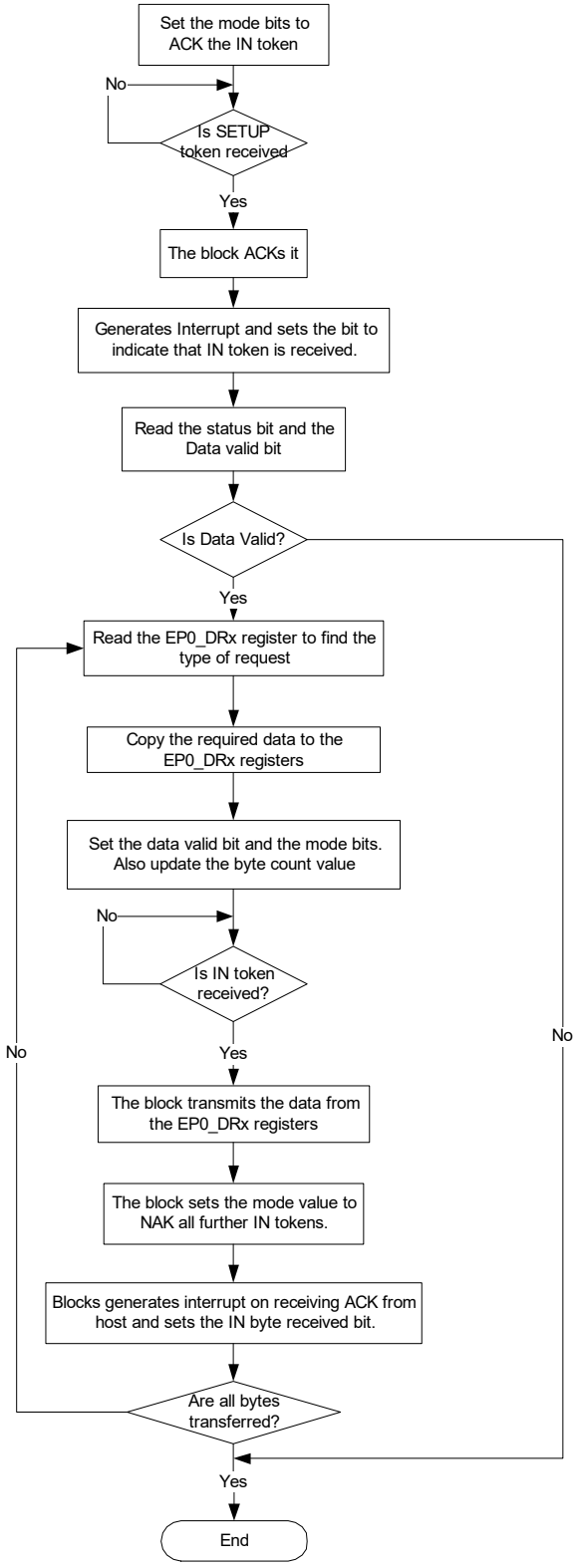
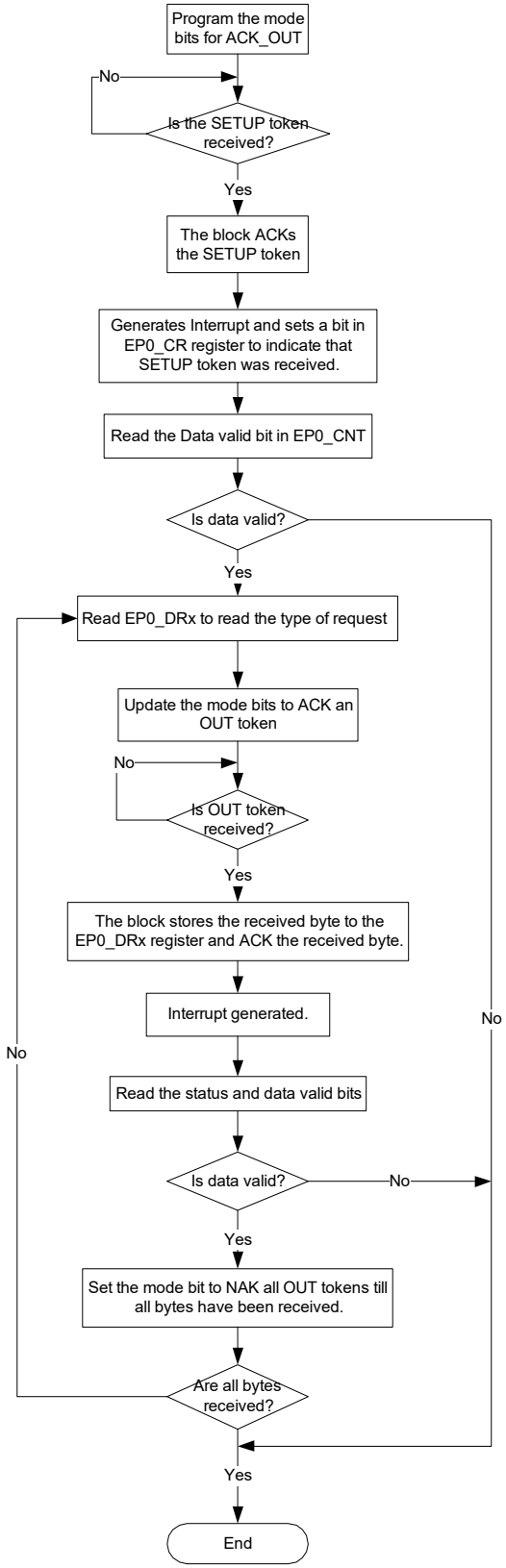


Figure 18-11. Control Endpoint OUT Transaction



18.5 Register Summary

Register Name	Description
USBDEVv2_EP0_DRx	Control Endpoint EP0 Data Register. 'x' can be 0–7 for the 8-byte EP0 data buffer
USBDEVv2_CR0	USB Control 0 Register. Register to enable the USB device and specify the 7-bit device address
USBDEVv2_CR1	USB Control 1 Register. Register to configure the USB block regulator, read bus activity status, and enable the internal oscillator to the USB traffic
USBDEVv2_SIE_EP_INT_EN	USB SIE Data Endpoints Interrupt Enable Register
USBDEVv2_SIE_EP_INT_SR	USB SIE Data Endpoint Interrupt Status
USBDEVv2_SIE_EPx_CNT0	Non-control Endpoint Byte Count Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register stores the most significant 3-bits of the 11-bit byte counter, the toggle state of the data packet, and data valid status.
USBDEVv2_SIE_EPx_CNT1	Non-control Endpoint Byte Count Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register stores the lower eight bits of the 11-bit byte count value.
USBDEVv2_SIE_EPx_CR0	Non-control Endpoint Control Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register contains the endpoint operating mode, error status, stall control, and the NAK interrupt generation.
USBDEVv2_USBIO_CR0	USBIO Control 0 Register. This register contains the control and configuration bits for the USB I/Os (D+ and D– pins respectively) for single-ended and differential mode operation.
USBDEVv2_USBIO_CR1	USBIO Control 1 Register. This register contains the control and configuration bits for the USB I/Os (D+ and D– pins respectively) for selecting the USB operating mode, reading the single ended USBIO receiver outputs, and enabling pull-up resistor on the D+ line.
USBDEVv2_DYN_RECONFIG	USB Dynamic Reconfiguration Register. This register is used to control the status of dynamic reconfiguration of an endpoint.
USBDEVv2_SOF0	Start Of Frame Register. This register contains the lower eight bits [7:0] of the SOF frame number.
USBDEVv2_SOF1	Start Of Frame Register. This register contains the upper three bits [10:8] of the SOF frame number.
USBDEVv2_OSCLK_DR0	Oscillator Lock Data Register 0. This register contains the lower eight bits of the oscillator locking circuit's adder output.
USBDEVv2_OSCLK_DR1	Oscillator Lock Data Register 1. This register contains the upper seven bits of the oscillator locking circuit's adder output.
USBDEVv2_EP0_CR	Endpoint0 Control Register. This register contains operating mode of the control endpoint, and the status bits for different packet conditions on the control endpoint.
USBDEVv2_EP0_CNT	Endpoint0 Count Register. This register stores the 4-bit byte counter, the toggle state of the data packet, and data valid status.
USBDEVv2_ARB_EPx_CFG	Endpoint Configuration Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register contains the settings to reset the endpoint buffer pointers, CRC bypass feature, manual DMA request, and the data ready status.
USBDEVv2_ARB_EPx_INT_EN	Endpoint Interrupt Enable Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. Register to configure the conditions under which an interrupt should be generated for an endpoint.
USBDEVv2_ARB_EPx_SR	Endpoint Interrupt Status Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. Register to read the interrupt status of an endpoint.
USBDEVv2_ARB_RWx_WA	Endpoint Write Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Lower eight bits of the 9-bit write address pointer.
USBDEVv2_ARB_RWx_WA_MSB	Endpoint Write Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Most significant bit of the 9-bit write address pointer.
USBDEVv2_ARB_RWx_RA	Endpoint Read Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Lower eight bits of the 9-bit read address pointer.
USBDEVv2_ARB_RWx_RA_MSB	Endpoint Read Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Most significant bit of the 9-bit read address pointer.
USBDEVv2_ARB_RWx_DR	Endpoint Data Register. 'x' can be 1–8 corresponding to one of the eight data endpoints.
USBDEVv2_BUF_SIZE	Dedicated Endpoint Buffer Size Register
USBDEVv2_EP_ACTIVE	Endpoint Active Indication Register

Register Name	Description
USBDEVv2_EP_TYPE	Endpoint Type (IN/OUT) Indication register
USBDEVv2_ARB_CFG	Arbiter Configuration Register. This register is used to configure the buffer management mode of the USB block.
USBDEVv2_USB_CLK_EN	USB Block Clock Enable Register
USBDEVv2_ARB_INT_EN	Arbiter Interrupt Enable Register. This register contains the configuration to enable arbiter interrupt generation for each endpoint.
USBDEVv2_ARB_INT_SR	Arbiter Interrupt Status Register. This register contains the status of arbiter interrupt generation due to each endpoint.
USBDEVv2_CWA	Common Area Write Address Register. This register contains the lower eight bits of the 9-bit write address pointer for the common area.
USBDEVv2_CWA_MSB	Common Area Write Address Register. This register contains the most significant bit of the 9-bit write address pointer for the common area.
USBDEVv2_DMA_THRES	DMA Burst/Threshold Configuration Register. This register contains the lower eight bits of the 9-bit threshold byte count value.
USBDEVv2_DMA_THRES_MSB	DMA Burst/Threshold Configuration Register. This register contains the most significant bit of the 9-bit threshold byte count value.
USBDEVv2_BUS_RST_CNT	Bus Reset Count Register. This register contains the number of clock cycles of LFCLK that should elapse to detect a bus reset condition.
USBDEVv2_MEM_DATAx	This is the 512-byte data buffer for storing the non-control endpoint data. 'x' can be 0 to 511.
USBDEVv2_SOF16	16-bit version of the Start Of Frame Register
USBDEVv2_OSCLK_DR16	16-bit version of the Oscillator Lock Data Register
USBDEVv2_ARB_RWx_WA16	16-bit version of the Endpoint Write Address Value Register. 'x' can be 1 to 8.
USBDEVv2_ARB_RWx_RA16	16-bit version of the Endpoint Read Address Value Register. 'x' can be 1 to 8.
USBDEVv2_CWA16	16-bit version of the Common Area Write Address Register
USBDEVv2_ARB_RWx_DR16	16-bit version of the Endpoint Data Register. 'x' can be 1 to 8
USBDEVv2_DMA_THRES16	16-bit version of the DMA Burst/Threshold Configuration Register
USBDEVv2_USB_POWER_CTRL	Power Control Register
USBDEVv2_USB_CHGDET_CTRL	Charger Detection Control Register
USBDEVv2_USB_USBIO_CTRL	USB I/O Control Register
USBDEVv2_USB_FLOW_CTRL	Flow Control Register
USBDEVv2_USB_LPM_CTRL	LPM Control Register
USBDEVv2_USB_LPM_STAT	LPM Status Register
USBDEVv2_USB_INTR_SIE	USB SOF, BUS RESET, and EP0 Interrupt Status Register
USBDEVv2_USB_INTR_SIE_SET	USB SOF, BUS RESET, and EP0 Interrupt Set Register
USBDEVv2_USB_INTR_SIE_MASK	USB SOF, BUS RESET, and EP0 Interrupt Mask Register
USBDEVv2_USB_INTR_SIE_MASKED	USB SOF, BUS RESET, and EP0 Interrupt Masked Register
USBDEVv2_USB_INTR_LVL_SEL	USB Interrupt Level Select Register
USBDEVv2_USB_INTR_CAUSE_HI	High Priority Interrupt Cause Register
USBDEVv2_USB_INTR_CAUSE_MED	Medium Priority Interrupt Cause Register
USBDEVv2_USB_INTR_CAUSE_LO	Low Priority Interrupt Cause Register
USBDEVv2_USB_PHY_TRIM0	PHY Trim Control Register
USBDEVv2_USB_PHY_TRIM1	PHY Trim Control Register
USBDEVv2_USB_PHY_TRIM2	PHY Trim Control Register
USBDEVv2_USB_PHY_TRIM3	PHY Trim Control Register
USBDEVv2_USB_CHGDET_TRIM	Charger Detect Trim Values Register
USBDEVv2_USB_TRIM	Trim Values Register
USBDEVv2_USB_USBIO_TRIM	Trim Values for I/Os Register

19. Timer, Counter, and PWM

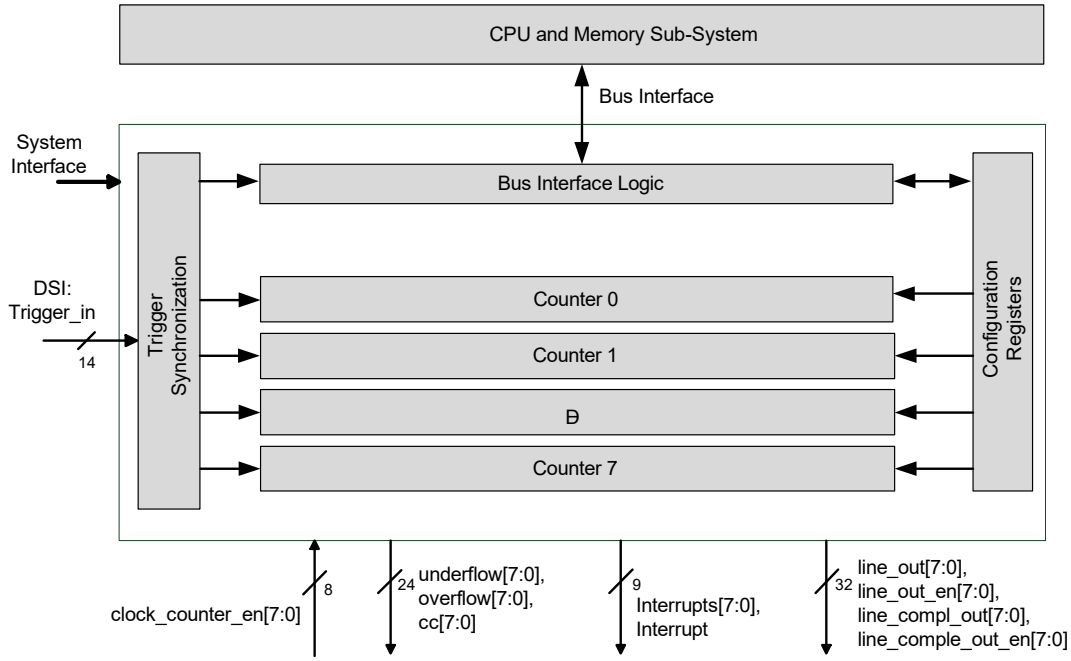
The Timer, Counter, and Pulse Width Modulator (TCPWM) block in PSoC® 4 implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular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

19.1 Features

- Eight 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 prescaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count – The final value in the counter register is reached
  - Capture/Compare – The count is captured to the capture/compare register or the counter value equals the compare value
- Synchronized counters – The counters can reload, start, stop, and count at the same time
- Complementary line output for PWMs

19.2 Block Diagram

Figure 19-1. 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 eight 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 247 for more information on all registers required for this block.

19.2.1 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.

19.2.2 Clocking

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

Clock Prescaling: counter_clock can be prescaled, with divider values of 1, 2, 4… 64, 128. This prescaling is done by modifying the GENERIC field of the counter control (TCPWM_CNT_CTRL) register, as shown in Table 19-1.

Table 19-1. Bit-Field Setting to Prescale Counter Clock

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

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

19.2.3 Events Based on Trigger Inputs

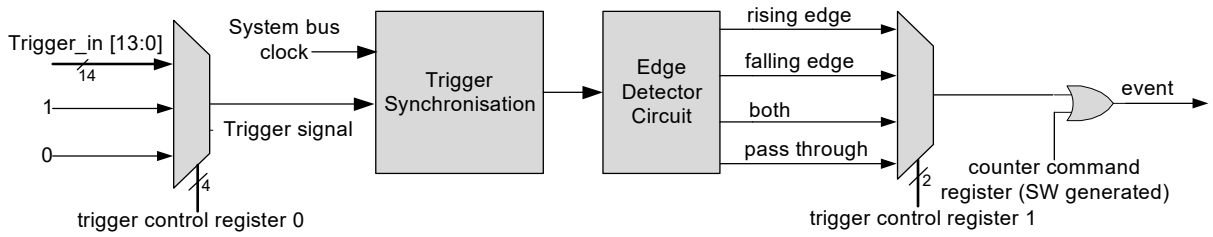
These are the events triggered by hardware or software.

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

Hardware triggers can be level signal, rising edge, falling edge, or both edges. Figure 19-2 shows the selection of edge detection type for any event trigger signal. The trigger control register 0 (TCPWM_CNT_TR_CTRL0) selects one of the 1416five trigger inputs as the event signal, which includes constant '0' and '1' signals.

Any edge (rising, falling, or both) or level (high) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNT_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 19-2.

Figure 19-2. Trigger Signal Edge Detection



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

- **Reload:** A reload event initializes and starts the counter.
  - In UP counting mode, the count register (TCPWM_CNT_COUNTER) is initialized with ‘0’.
  - In DOWN counting mode, the counter is initialized with the period value stored in the TCPWM_CNT_PERIOD register.
  - In UP/DOWN counting mode, the count register is initialized with ‘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 236.

- **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.

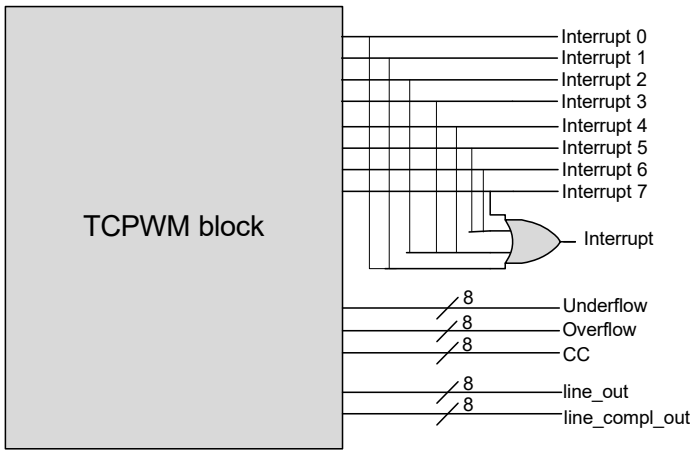
Notes

- 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.

19.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 19-3.

Figure 19-3. TCPWM Output Signals



19.2.4.1 Signals upon Trigger Conditions

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

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

19.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific. All eight interrupt output signals from the eight TCPWMs are also OR'ed together to produce a single interrupt output signal.

Four registers are used for interrupt handling in this block, as shown in Table 19-2.

Table 19-2. 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.

19.2.4.3 Outputs

The TCPWM has two outputs, line_out and line_compl_out (complementary of line_out). Note that the OV, UN, and CC conditions can be used to drive line_out and line_compl_out if needed, by configuring the TCPWM_CNT_TR_CTRL2 register (Table 19-3). The line_out and line_compl_out is enabled by the line_out_en and line_compl_out_en, one for each counter.

Table 19-3. 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	

19.2.5 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 19-4 for details.

Table 19-4. 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.

19.3 Modes of Operation

The counter block can function in six operational modes, as shown in Table 19-5. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

Table 19-5. Operational Mode Configuration

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

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the TCPWM_CNT_CTRL register, as shown in Table 19-6.

Table 19-6. Counting Mode Configuration

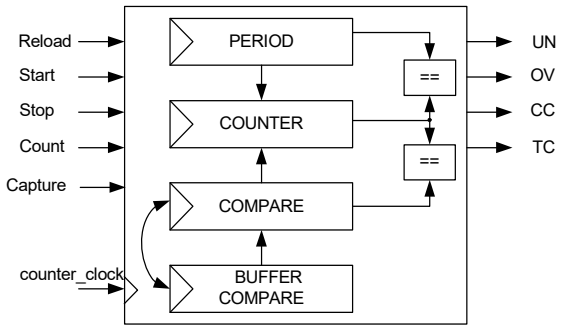
Counting Modes	UP_DOWN_MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when the counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until 0 is reached. A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

19.3.1 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.

19.3.1.1 Block Diagram

Figure 19-4. Timer Mode Block Diagram



19.3.1.2 How It Works

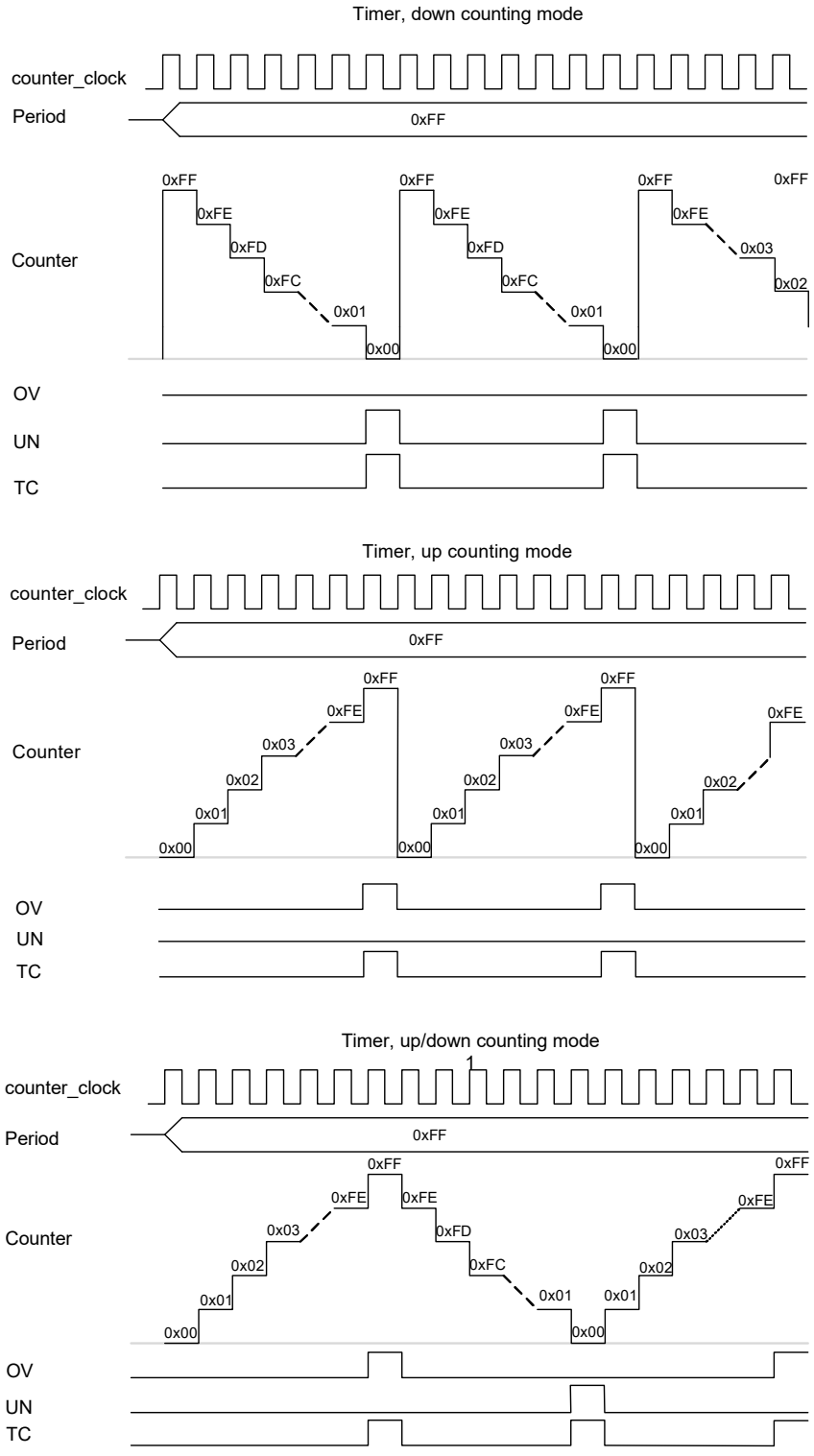
The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode. The following explains the working of the timer:

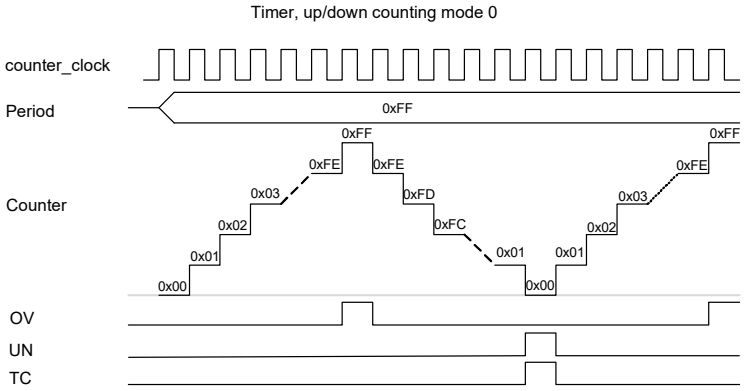
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).
    Note It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The counter 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 19-5 shows the timer operational mode of the counter in four different counting modes. The period register contains the maximum counter value.

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

Figure 19-5. 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 228. The figures in this chapter assume that HFCLK and counter clock are the same.

19.3.1.3 Configuring Counter for Timer Mode

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

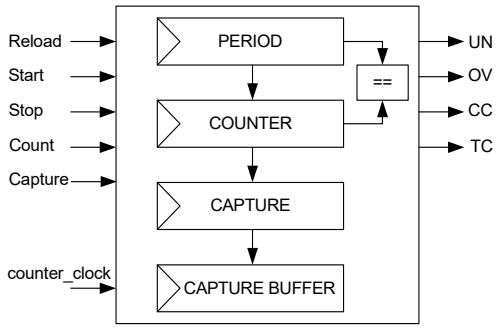
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set the 16-bit 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 prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 19-1.
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 19-6.
8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of TCPWM_CNT_CTRL.
9. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
10. Set the TCPWM_CNT_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
11. If required, set the interrupt upon TC or CC condition, as shown in "Interrupts" on page 228.
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.

19.3.2 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.

19.3.2.1 Block Diagram

Figure 19-6. Capture Mode Block Diagram



19.3.2.2 How it Works

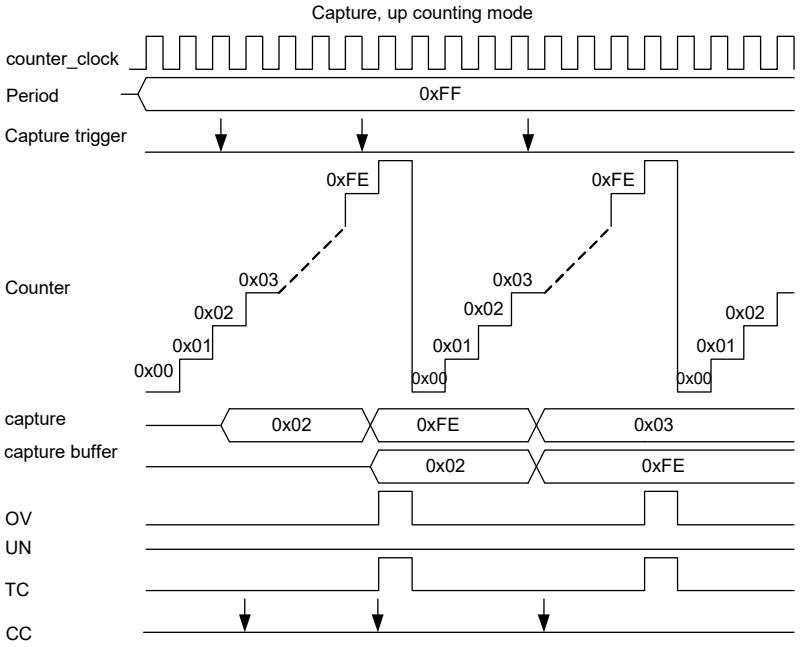
The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNT_CTRL).

Operation in capture mode occurs as follows:

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

Figure 19-7 illustrates the capture behavior in the up counting mode.

Figure 19-7. 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.

19.3.2.3 Configuring Counter for Capture Mode

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

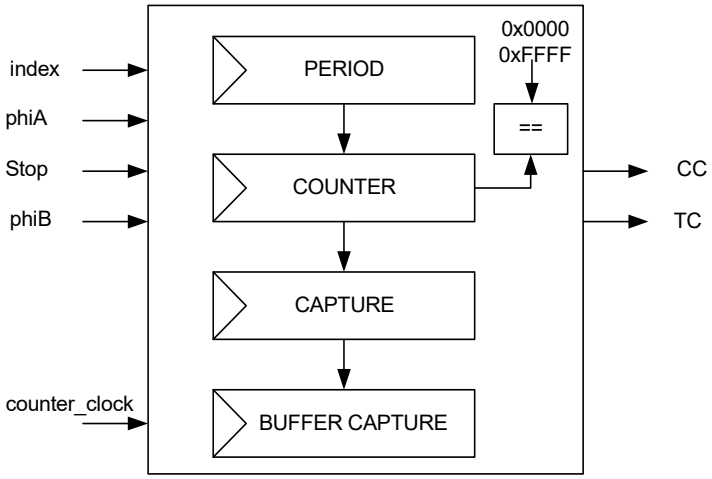
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNT_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 19-1.
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 19-6.
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 228.
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.

19.3.3 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.

19.3.3.1 Block Diagram

Figure 19-8. Quadrature Mode Block Diagram



19.3.3.2 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.

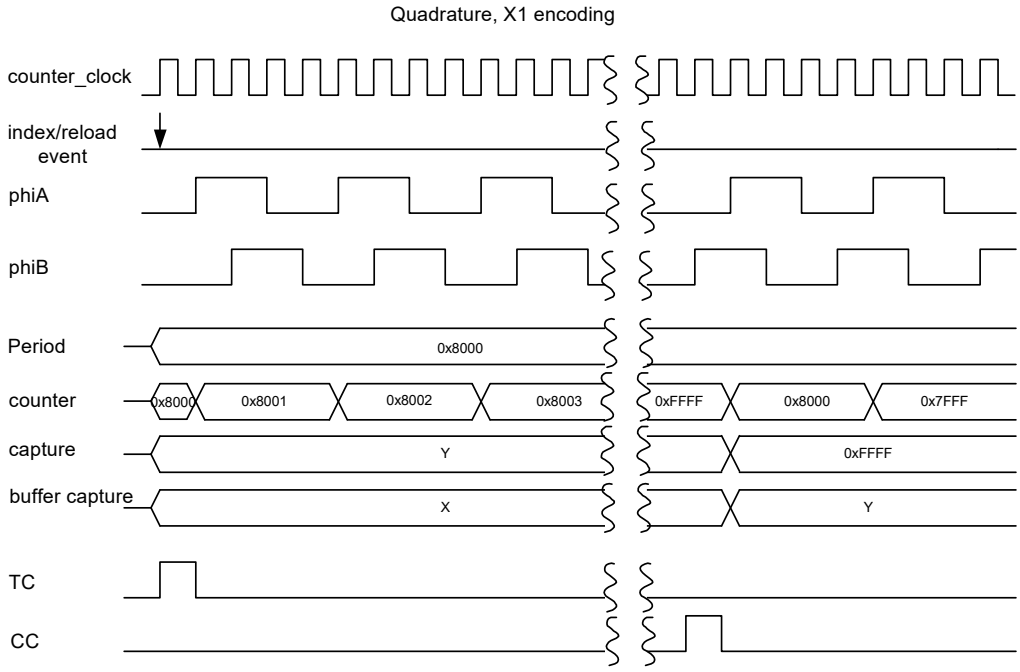
The Quadrature mode operation occurs as follows:

- 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 19-9.

  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 19-9 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.

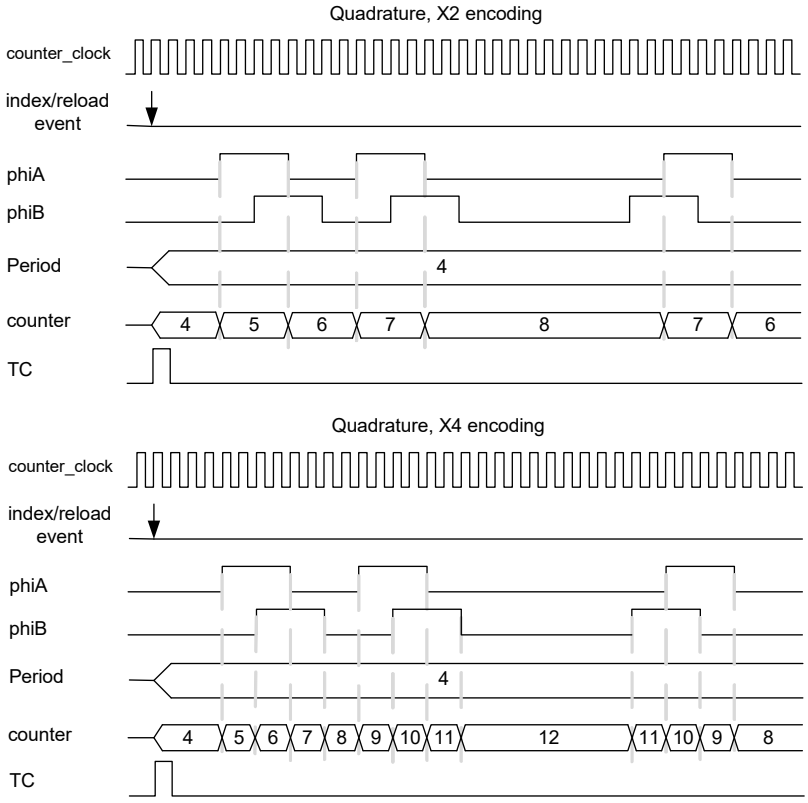
Figure 19-9. Timing Diagram for Quadrature Mode, X1 Encoding



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 19-10 illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

Figure 19-10. Timing Diagram for Quadrature Mode, X2 and X4 Encoding



19.3.3.3 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 228.
8. Enable the counter by writing '1' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.

19.3.4 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

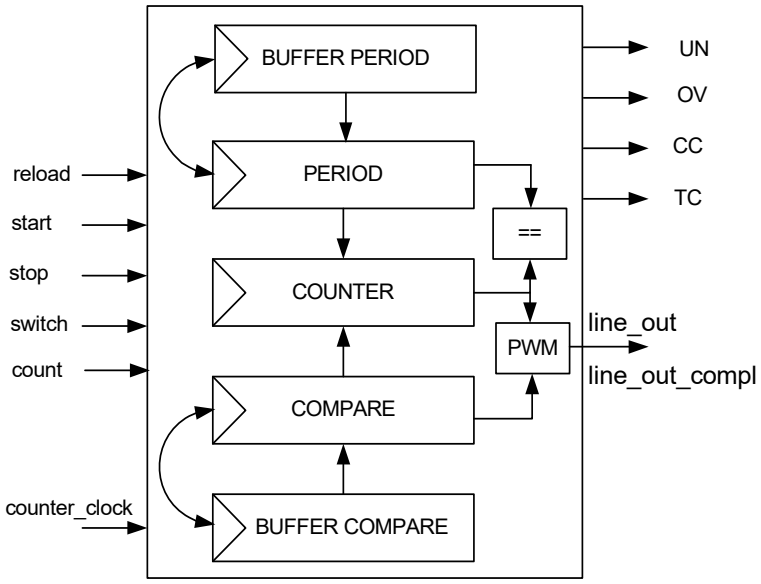
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

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

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

19.3.4.1 Block Diagram

Figure 19-11. PWM Mode Block Diagram



19.3.4.2 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 19-6.

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 19-13.

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 19-12 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 19-12. Timing Diagram for Center Aligned PWM

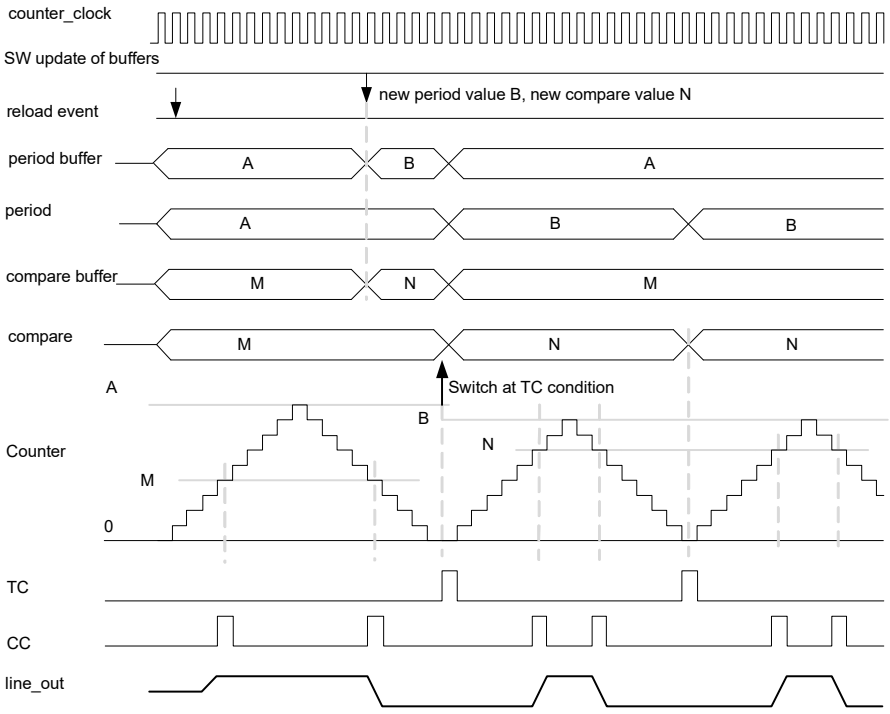
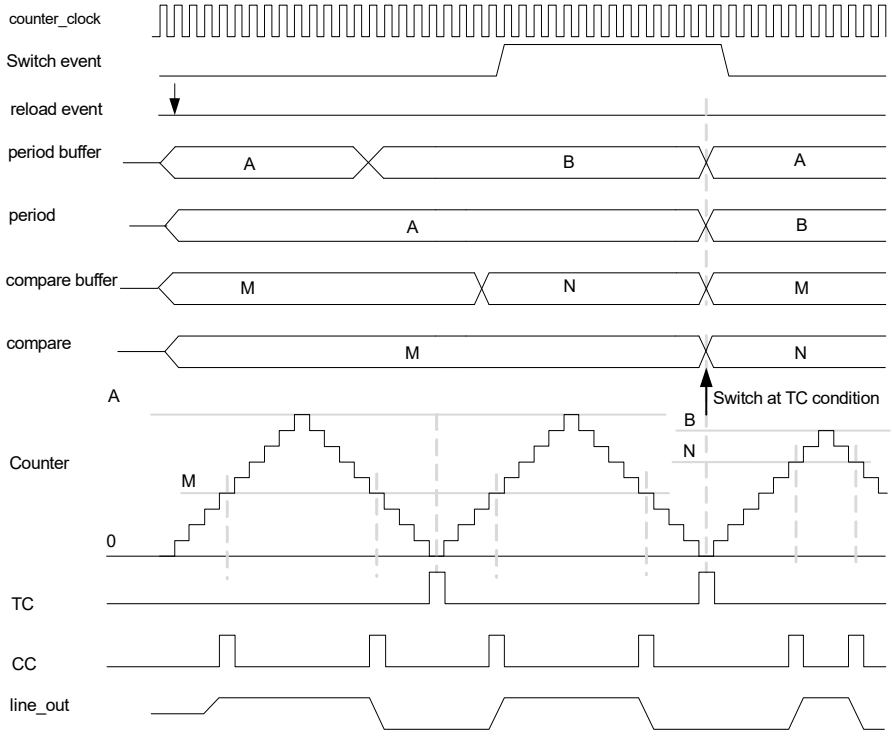


Figure 19-12 illustrates center-aligned PWM with software generated switch events:

- 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 19-13. Timing Diagram for Center Aligned PWM (software switch event



19.3.4.3 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 19-3.
- 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 19-3.

19.3.4.4 Kill Feature

The kill feature gives the ability to disable both output lines immediately. This event can be programmed to stop the counter by modifying the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the counter control register, as shown in Table 19-7.

Table 19-7. 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 19-8.

Table 19-8. 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 19-8). 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.

19.3.4.5 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.

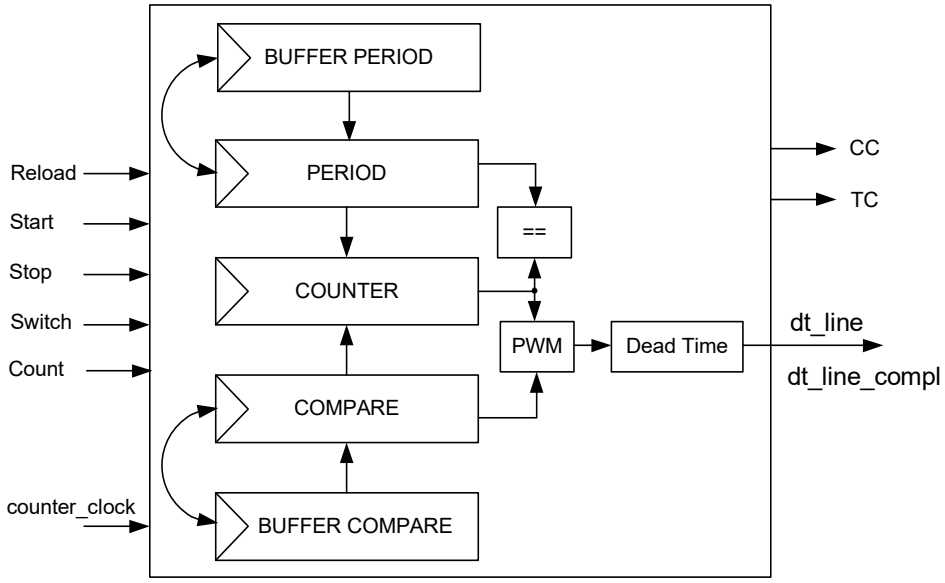
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 19-1.
4. Set the required 16-bit period in the TCPWM_CNT_PERIOD register and the buffer period value in the TCPWM_CNT_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNT_CC register and buffer compare value in the TCPWM_CNT_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNT_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 19-6.
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 228.
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.

19.3.5 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.

19.3.5.1 Block Diagram

Figure 19-14. PWM-DT Mode Block Diagram



19.3.5.2 How It Works

The PWM operation with Dead Time mode occurs as follows:

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

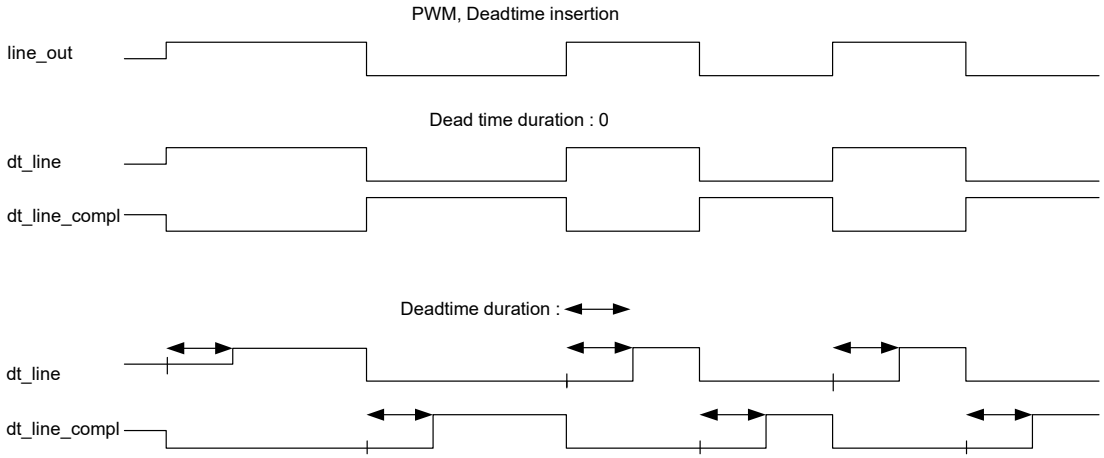
This mode follows PWM mode and supports the following features available with that mode:

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

This mode does not support clock prescaling.

Figure 19-15 illustrates how the complementary output lines dt_line and dt_line_compl are generated from the PWM output line, line_out.

Figure 19-15. Timing Diagram for PWM, with and without Dead Time



19.3.5.3 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:

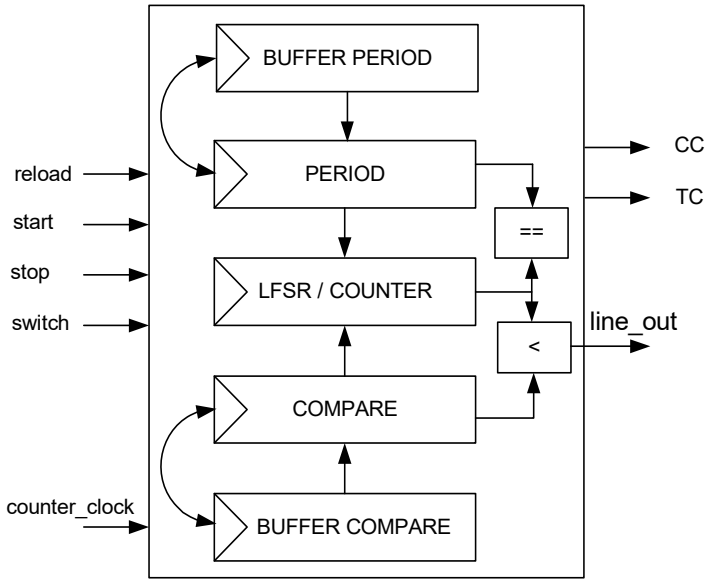
1. Disable the counter by writing '0' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNT_CTRL register, as shown in Table 19-1.
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 19-6.
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 239.
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 228.
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.

19.3.6 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.

19.3.6.1 Block Diagram

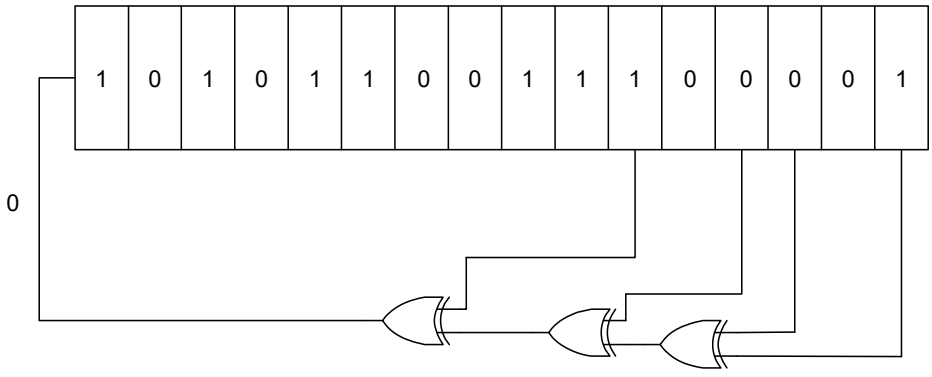
Figure 19-16. PWM-PR Mode Block Diagram



19.3.6.2 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 19-17. 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.

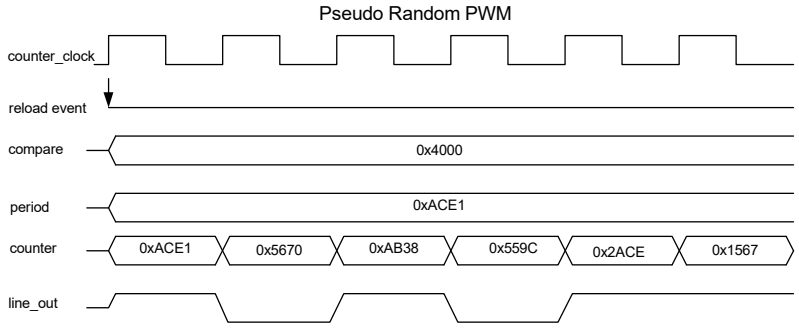
Figure 19-17. Pseudo-Random Sequence Generation using Counter Register



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 19-18 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 19-18. 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.

19.3.6.3 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 COUNTER_ENABLED of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNT_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNT_PERIOD register and buffer period value in the TCPWM_CNT_PE-RIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNT_CC register and the buffer compare value in the TCPWM_CNT_C-C_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNT_CTRL register as required.
6. Set the TCPWM_CNT_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNT_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNT_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in “Interrupts” on page 228.
10. Enable the counter by writing '1' to the corresponding bit in the COUNTER_ENABLED field of the TCPWM_CTRL register.

19.4 TCPWM Registers

Table 19-9. 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 prescaling, and counting direction
TCPWM_CNTx_STATUS	Counter status register	Reads the direction of counting, dead time duration, and clock prescaling; 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	Bit-wise AND of interrupt request and mask registers

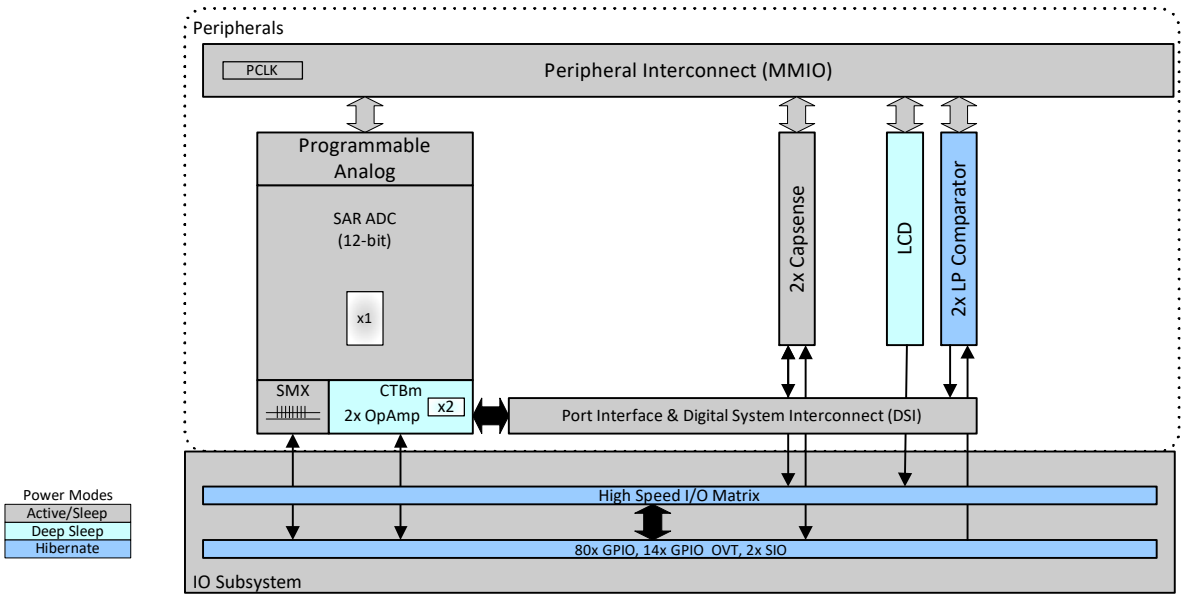
Section E: Analog System

This section encompasses the following chapter:

- Precision Reference chapter on page 249
- SAR ADC chapter on page 252
- Low-Power Comparator chapter on page 285
- Continuous Time Block mini (CTBm) chapter on page 291
- LCD Direct Drive chapter on page 301
- CapSense chapter on page 313
- Temperature Sensor chapter on page 314

Top Level Architecture

Analog System Block Diagram



20. Precision Reference

PSoC® 4 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

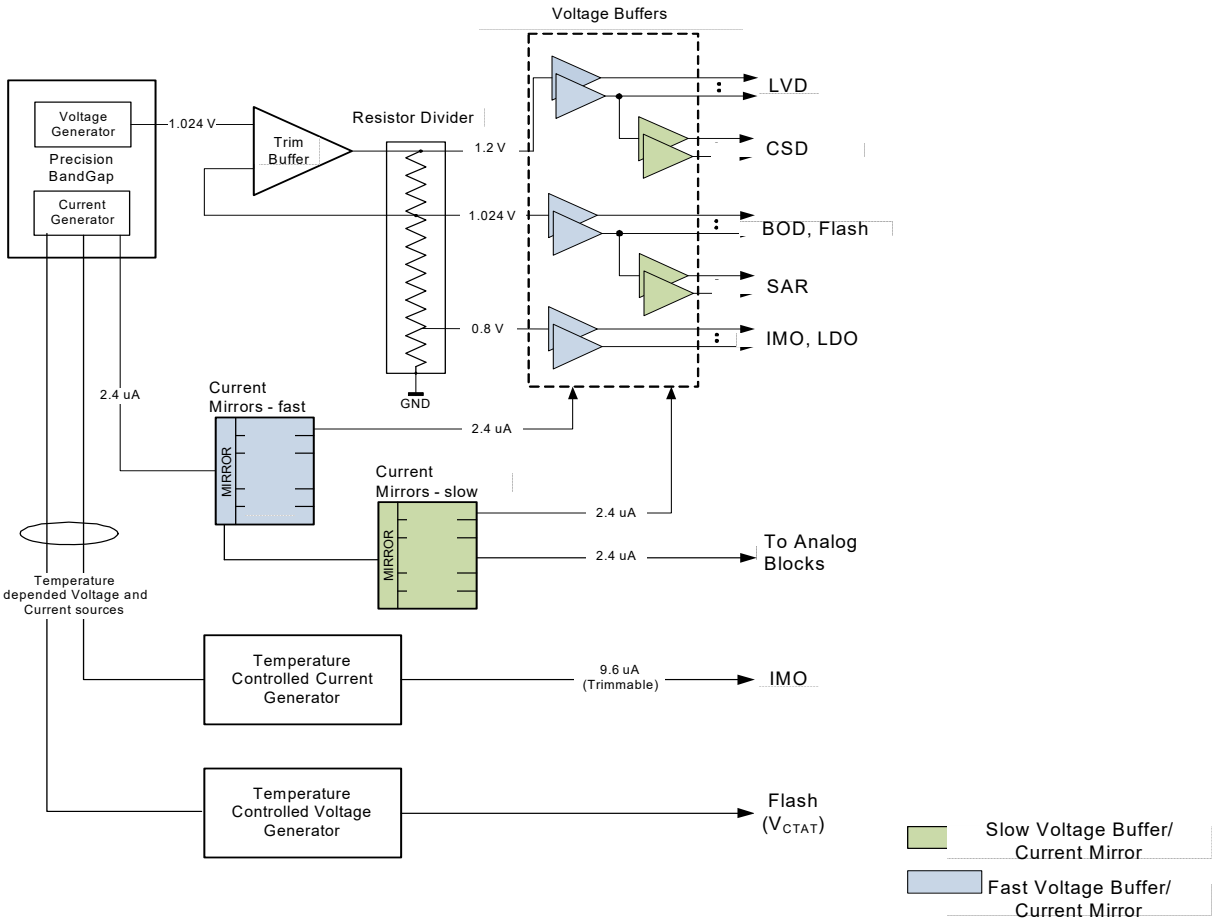
20.2 Block Diagram

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-µA 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

PSoC 4 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 µA 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 value and slow current mirror can settle in 40 µs. 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.

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 (V_{CTAT}) 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 References

Voltage References	Buffer Speed	Description
1.2 V	Fast	Reference to the LVD block
1.2 V	Slow	Reference to the USB regulator in 5-V mode
1.2 V	Slow	Reference to the CapSense block
1.024 V	Fast	Reference to the BOD block
1.024 V	Fast	Bias reference voltage to the flash block for flash read-out
1.024 V	Slow	Reference to the ECO block for amplitude control
1.024 V	Slow	Reference to the SAR ADC block
0.8 V	Fast	Comparator threshold for relaxation oscillator in the IMO
0.8 V	Fast	Reference to VCCD and VCCA regulators in the LDO block
VCTAT	Fast	Temperature dependent voltage reference for flash positive voltage (VPOS) pump

Table 20-2. Current References

Current References	Buffer Speed	Description
2.4 µA	Fast	Current reference for USB, LCD drive, BOD, and flash blocks, and bias for fast voltage buffers
2.4 µA	Slow	Current reference for analog blocks (SAR, CapSense, IDAC, LPCOMP, and CTBm) and bias for slow voltage buffers
9.6 µA	Fast	Current reference for the IMO block 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 PSoC® 4 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 ADC with sampling rate of 1 Msps in PSoC 4200L. Preceding the SAR ADC is the SARMUX, which can route external pins and internal signals (AMUXBUS-A/-B, CTBm, temperature sensor output) to the eight internal channels of SAR ADC. SARREF is used for multiple reference selection. The sequencer controller SARSEQ is used to control SARMUX and SAR ADC to do an automatic scan on all enabled channels without CPU intervention and for pre-processing, such as averaging the output data.

The ninth channel is an injection channel that is used by firmware for infrequent and incidental sampling of pins and signals, for example, the internal temperature sensor.

The result from each channel is double-buffered and a complete scan may be configured to generate an interrupt at the end of the scan. Alternatively, the data can be routed to programmable digital blocks (UDBs) for further processing without CPU intervention. The sequencer may also be configured to flag overflow, collision, and saturation errors that can be configured to assert an interrupt.

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

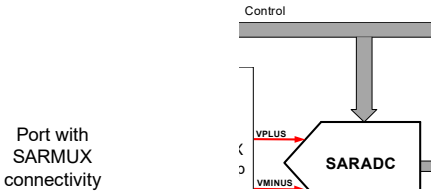
21.1 Features

- Operates across the entire device power supply range
- Maximum 1 Msps sample rate in PSoC 4200L
- Eight individually configurable channels and one injection channel
- Each channel has the following features:
  - Input from external pin (only for eight channels in single-ended mode and four channels in differential mode) or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Programmable acquisition times
  - Selectable 8-, 10-, and 12-bit resolution
  - Single-ended or differential measurement
  - Averaging
  - Results are double-buffered
  - Result may be left or right aligned
- Scan triggered by firmware, timer, pin, or UDB
  - One shot–periodic or continuous mode
- Hardware averaging support
  - First order accumulate
  - Samples averaging from 2 to 256 (powers of 2)

- Results represented in 16-bit sign extended values
- Selectable voltage references
  - Internal V_{DDA} and $V_{DDA}/2$ references
  - Internal 1.024-V reference with buffer
  - External reference
- Interrupt generation
  - Finished scan conversion
  - Saturation detect and over-range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect
- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single-ended or differential, averaging
- 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 have dedicated low power modes

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

PSoC 4 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

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

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

Note that Vn ± V_{REF} should be in the range of V_{SSA} to V_{DDA}. For example, if 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:

- Signed/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 (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 PSoC 4 pin. Because of this, the global configuration bit SINGLE_ENDED_SIGNED (SAR_SAMPLE_CTRL[2]) will be ignored and the result is always (0x000-0x7FF).

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

Note that single-ended conversions with Vminus connected to 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

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

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

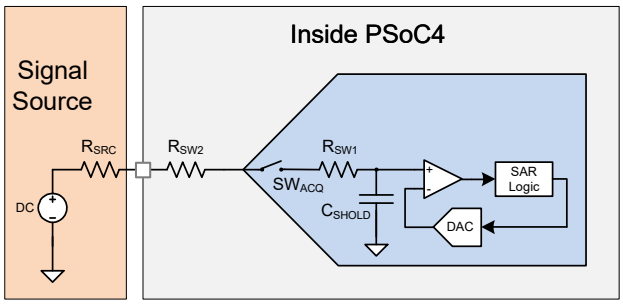
Total acquisition and conversion time (sar_clk) = acquisition time + resolution (bit) + 2

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

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:

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

Where:

C_{SHOLD} ~= 10 pF

$R_{SW2} + R_{SW1}$ = ~ 500 to 1000 ohms, depending on the routing path (See Analog Routing on page 257 for details).

R_{SRC} = series resistance of the signal source

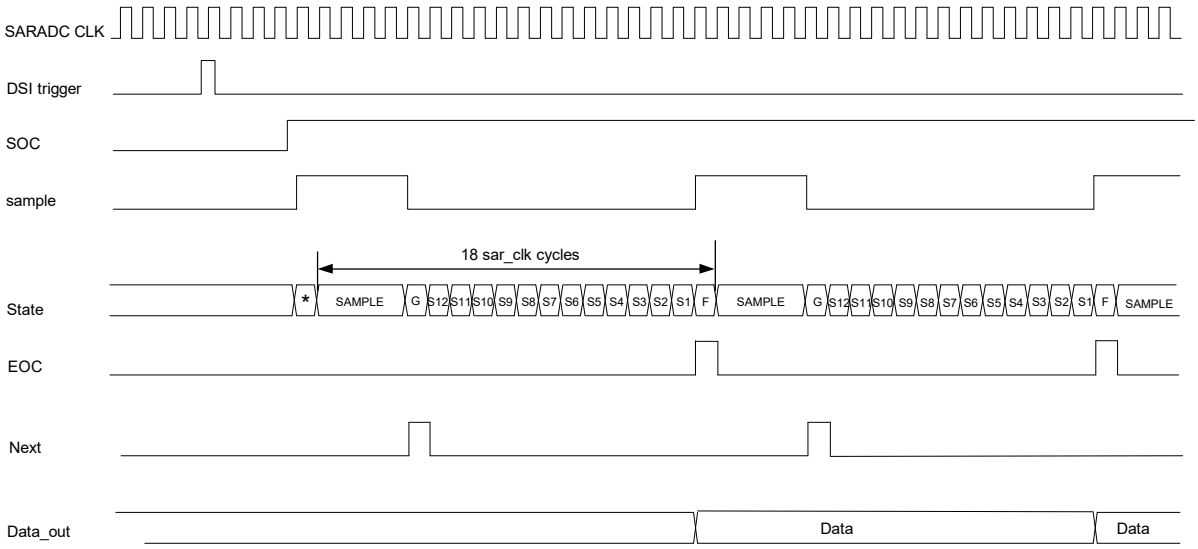
21.3.1.7 SAR ADC Clock

SAR ADC clock frequency must be between 1 MHz and 18 MHz for PSoC 4200L, which comes from the HFCLK via a clock divider. Note that a fractional divider is not supported for SAR ADC. To get a 1-Msps sample rate, an 18-MHz SAR ADC clock is required. To achieve this, the system clock (HFCLK) must be set to 36 MHz rather than 48 MHz. A 12-bit ADC conversion with the minimum acquisition time of four clocks (at 18 MHz) requires 18 clocks in which to complete. A 10-bit and 8-bit conversion requires 16 and 14 clocks respectively. Note that the minimum acquisition time of four clock cycles at 18 MHz is based on the minimum acquisition time supported by the SAR block (R_{SW1} and C_{SHOLD} in Figure 21-2), which is 194 ns.

21.3.1.8 SAR ADC Timing

As Figure 21-3 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 another pin and signal; this will be done automatically with sequencer control (see SARSEQ on page 265 for details).

Figure 21-3. SAR ADC Timing



21.3.2 SARMUX

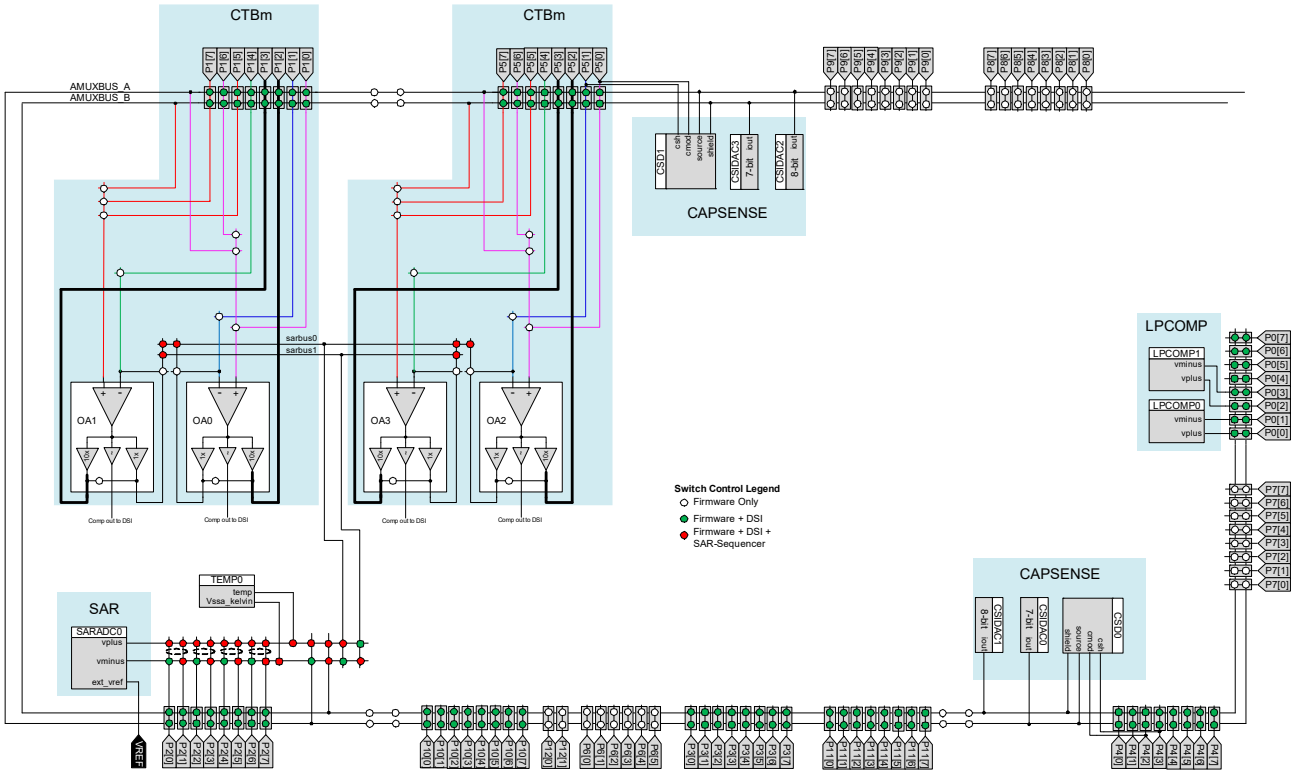
SARMUX is an analog dedicated programmable multiplexer. The main features of SARMUX are:

- Switch on resistance: 600 Ω (maximum)
- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ), firmware, or UDBs
- Charge pump inside:
  - If $V_{DDA} < 4.0$ V, charge pump should be turned on to reduce switch resistance
  - If $V_{DDA} \geq 4.0$ V, charge pump is turned off and delivers V_{DDA} as its output
- Multiple inputs:
  - Analog signals from pins (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), firmware, or the DSI. Sequencer and DSI are the hardware control method, which can be masked by the hardware control bit in the register, SAR_MUX_SWITCH_HW_CTRL. Different control methods have different control capability on the switches. See Figure 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 278.

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 278.

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 274.

21.3.2.2 Analog Interconnection

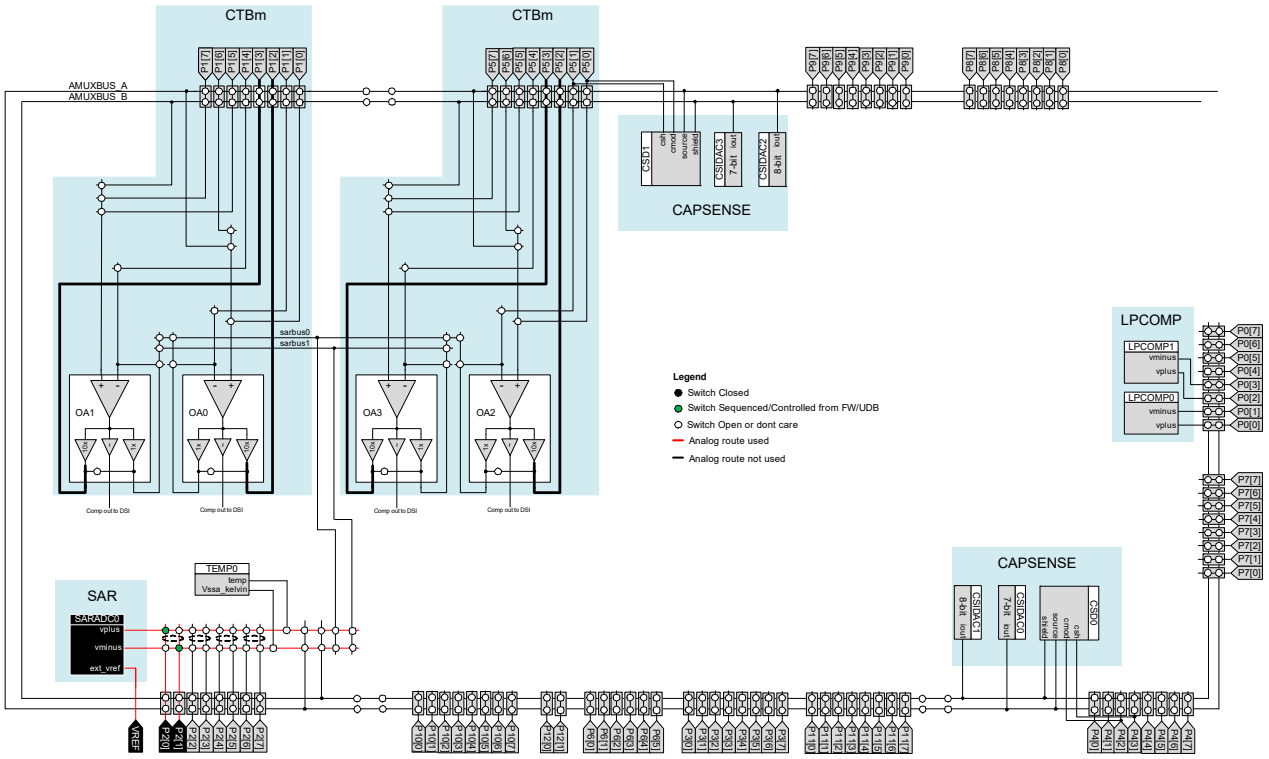
PSoC 4 analog interconnection is very flexible. SAR ADC can be connected to multiple inputs via SARMUX, including both external pins and internal signals. For example, it can connect to a neighboring block such as CTBm. It can also connect to other pins except port 2 through AMUXBUS_A/B, at the expense 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 P2[0] and P2[1], P2[2] and P2[3], and so on. If you need to use pins that are not paired as a differential pair, such as P2[1] and P2[2], the sequencer does not work; use firmware or DSI.

Figure 21-5. Input from External Pins

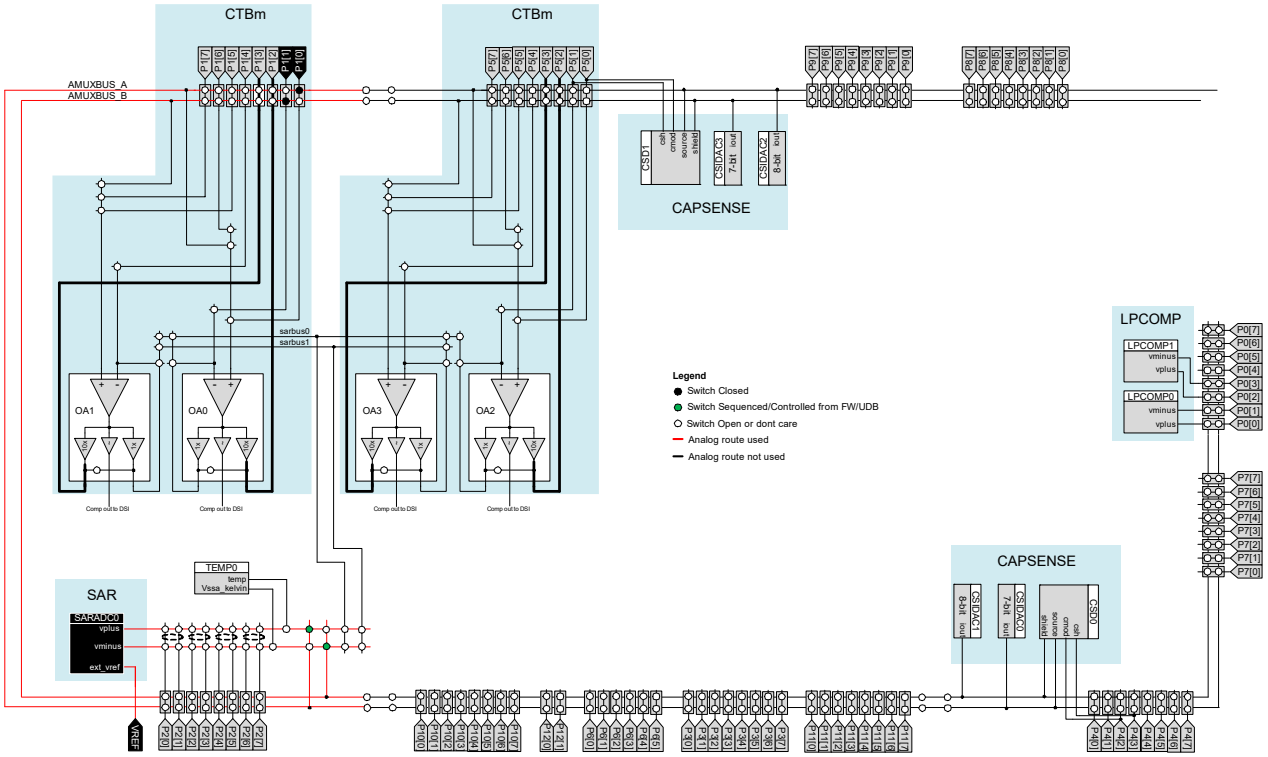


Input from Analog Bus (AMUXBUS_A/B)

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 two pins to AMUXBUS_A and AMUX-BUS_B, and then connect AMUXBUS_A and AMUXBUS_B to ADC.

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

Figure 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

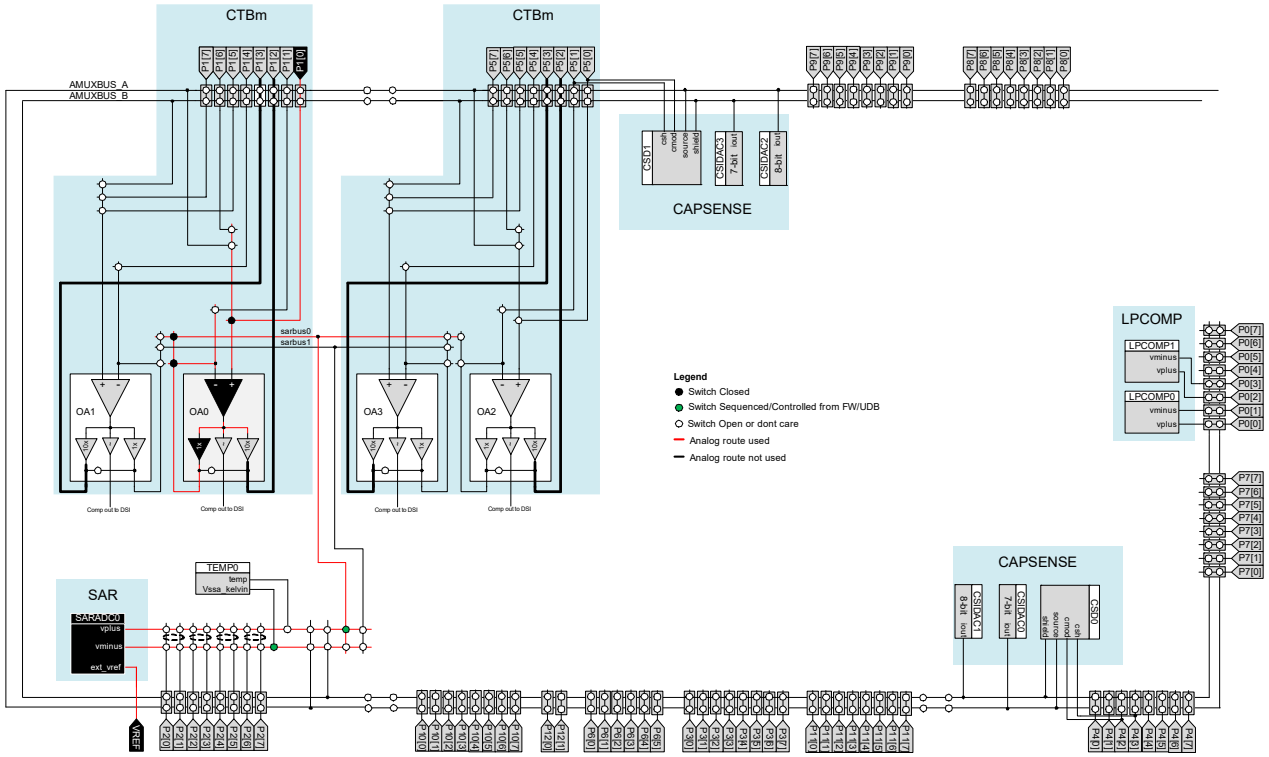
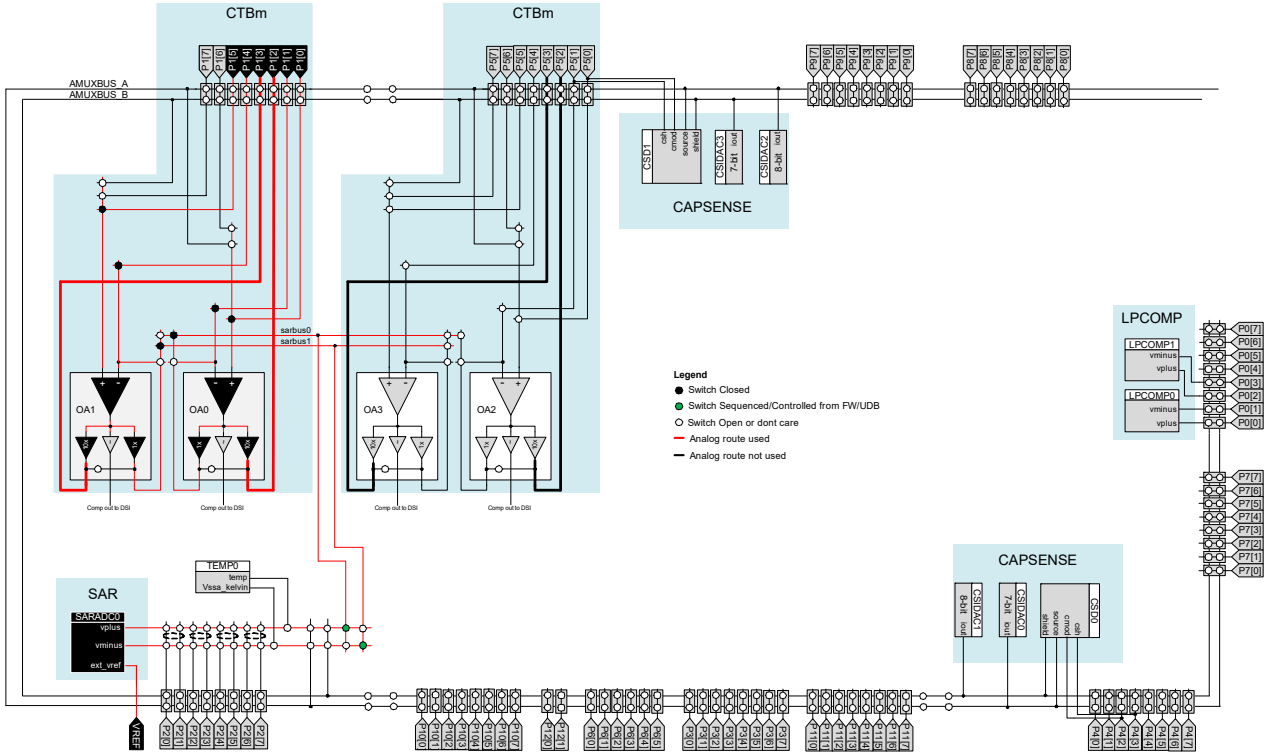


Figure 21-8. Inputs from CTBm Output via sarbus0 and sarbus1

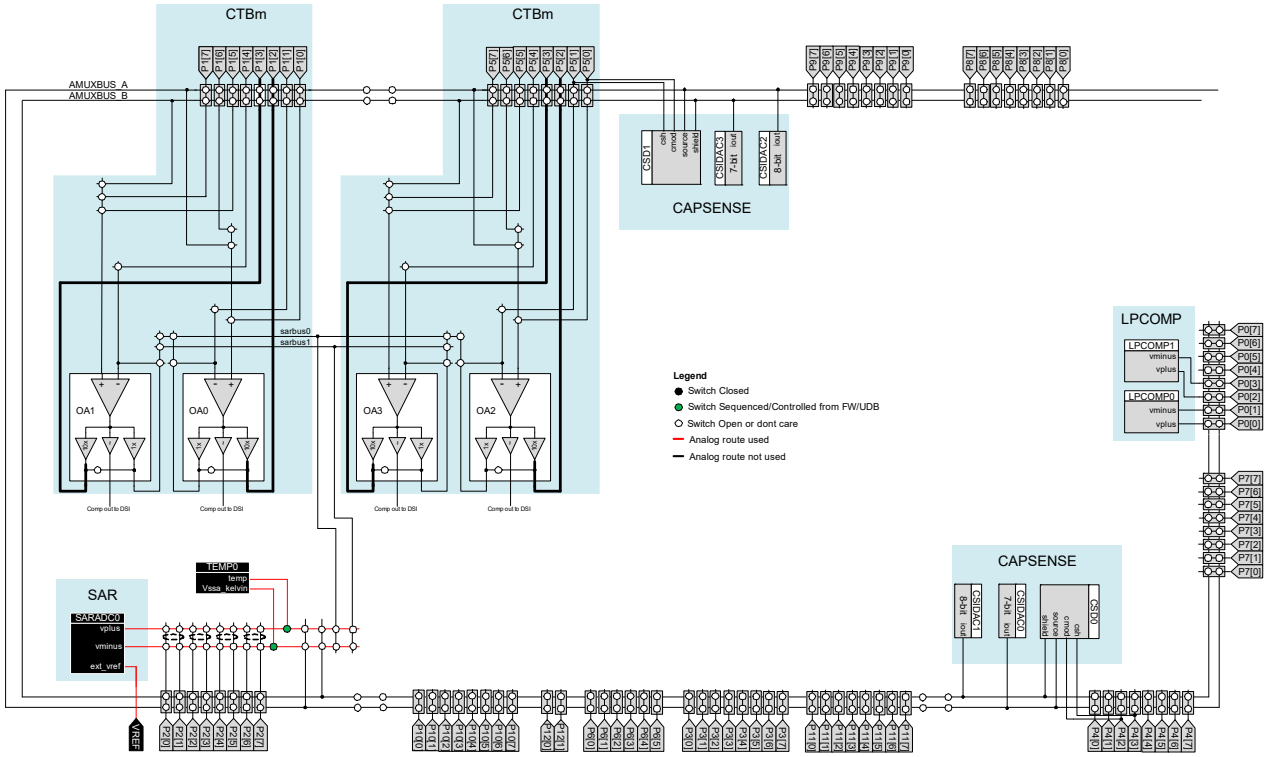


Input from Temperature Sensor

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.

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

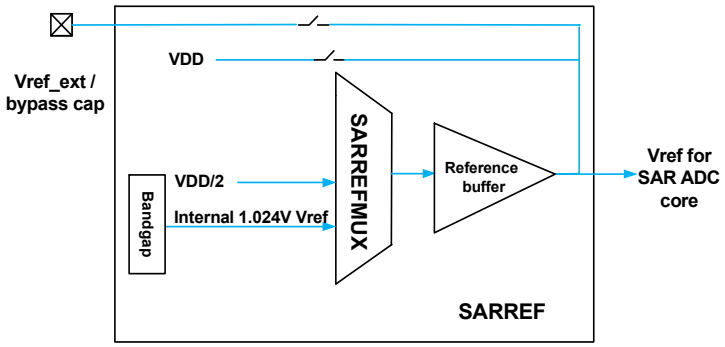


21.3.3 SARREF

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 is limited to 100 ksps (at 12-bit) 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 (12-bit)
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} (low-impedance path)	5	X	No	18 MHz	1 Msps
$V_{DDA}/2$ without bypass cap	6	0	Yes	1.6 MHz	100 ksps
$V_{DDA}/2$ with bypass cap	6	1	Yes	18 MHz	1 Msps
V_{DDA}	7	X	No	9 MHz	500 ksps

1.024-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 or when scanning after Sleep/Deep-Sleep, make sure the 1.024-V internal V_{REF} is settled when SAR ADC starts sampling. The worst case settling time (when V_{REF} is completely discharged) is the same as the startup time.

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

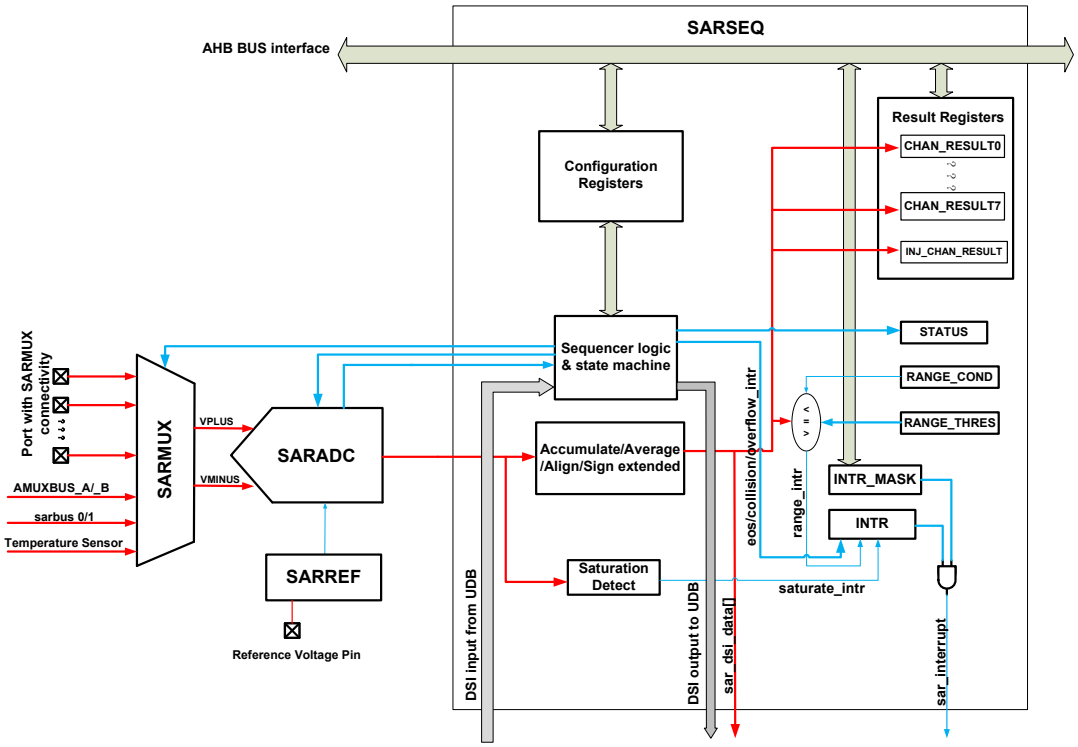
SARSEQ is a dedicated sequencer controller that automatically sequences the input mux from one channel to the next while placing the result in an array of registers, one per channel.

- Controls SARMUX analog routing automatically without CPU intervention
- Controls SAR ADC core (such as resolution, acquisition time, and reference)
- Receives data from SAR ADC and does pre-processing (average, range detect)
- Uses double buffer to store the result 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:
  - Single-ended or differential mode
  - Input from external pin (only for eight channels in single-ended mode and four channels in differential mode) 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
  - 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
  - Triggered by firmware
  - 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 269 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 action allows sufficient time for the firmware to read the previous scan before the present scan is completed. All input channels are double buffered with 16 registers, except the injection channel. The injection channel is not required to be doubled buffered because it is not normally part of a normal channel scan.

21.3.4.4 Injection Channel

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

The injection channel can only be controlled by the firmware with a firmware trigger (one-shot). This means the injection channel does not support continuous or DSI trigger. It also does not support output of its data or interrupt to the DSI bus. Because the only trigger is one-shot, there is no need for double buffering or an overflow interrupt.

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

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

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

- Maskable end-of-conversion interrupt INJ_EOC_INTR
- Maskable range detect interrupt INJ_RANGE_INTR
- Maskable saturation detect interrupt INJ_SATURATE_INTR
- Maskable collision interrupt INJ_COLLISION_INTR

SAR_INTR, SAR_INTR_MASK, SAR_INTR_MASKED, and SAR_INTR_SET are the corresponding registers.

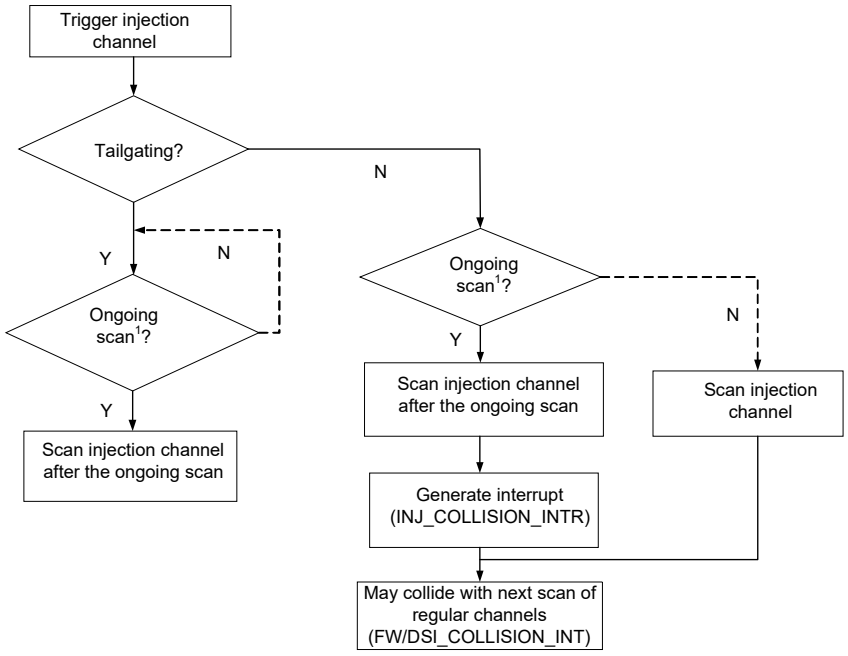
These features are described in detail in Global SARSEQ Configuration on page 275, Channel Configurations on page 275, and Interrupt on page 268.

Tailgating

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

If tailgating is not selected, setting the INJ_START_EN bit immediately starts the conversion of the injection channel provided there is no ongoing scan or SAR ADC is idle. If a scan of the regular channels is ongoing, then the injection channel will be scanned at the end of the ongoing scan, but it will cause a collision and generate a collision interrupt (INJ_COLLISION_INTR). Another potential problem without tailgating is that it can cause the next scan of the regular channels to collide with the injection channel conversion (FW/DSI_COLLISION_INTR is raised). As a result, the next scan of regular channels is postponed until the injection scan is finished, thus causing jitter on a regular scan. Note that continuous trigger and DSI trigger level mode will never trigger a collision interrupt.

Figure 21-12. Injection Channel Flow Chart



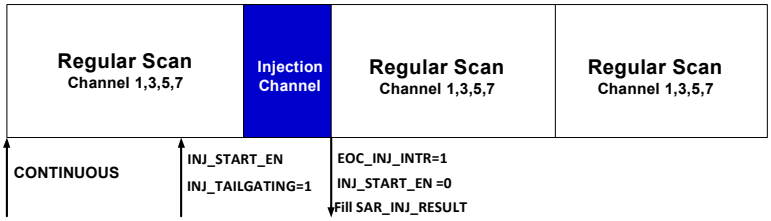
[1] *scan here means scan of ALL the regular channels*

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

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

Figure 21-13 is an example when injection channel is enabled during a continuous scan (channel 1, 3, 5, and 7 are enabled), and tailgating is enabled. Note that the INJ_START_EN bit is immediately cleared when the SAR is disabled (but only if it was enabled before).

Figure 21-13. Injection Channel Enabled with Tailgating



21.3.5 Interrupt

An interrupt can be generated on different events:

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

This section describes each interrupt in detail. These interrupts have an interrupt mask in the 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 register provides a convenient way for the firmware to determine the source of the interrupt.

For verification and debug purposes, a set bit (such as EOS_SET in the SAR_INTR_SET register) is used to trigger each interrupt. This action 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 three collision interrupts: for the firmware trigger (FW_COLLISION_INTR), for the DSI trigger (DSI_COLLISION_INTR), and for the injection channel (INJ_COLLISION_INTR). These interrupts allow 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 maximum value for the specific resolution and sets a maskable interrupt flag for the corresponding channel. This action allows the firmware to take action, such as discarding the result, when the SAR ADC saturates. The sample value is tested 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 three possible ways to trigger a scan are:

- A firmware or one-shot trigger is generated when the firmware writes to the FW_TRIGGER bit of the SAR_START_CTRL register. After the scan is completed, the SARSEQ clears the FW_TRIGGER bit and goes back to idle mode waiting for the next trigger. The FW_TRIGGER bit is cleared immediately after the SAR is disabled.
- A periodic trigger comes in over the DSI connections (dsi_trigger). This trigger is connected to the output of a TCPWM; however, it can also be connected to any GPIO pin or a UDB. The UDB can implement a state machine looking for a certain sequence of events.
- A continuous trigger is activated by setting the CONTINUOUS bit in SAR_SAMPLE_CTRL register. In this mode, after completing a scan the SARSEQ starts the next scan immediately; therefore, the SARSEQ is always BUSY. As a result, all other triggers are essentially ignored. Note that FW_TRIGGER will still get cleared by hardware on the next completion.

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

For firmware 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 72 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 Specification	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).

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

Table 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	9	3	529
2	133	18	4	1000
3	25	4.5	2	281

In addition to controlling the power of the SAR ADC core, the power consumed by VREF buffer (if used) can also be configured. Note that for full VDDA range (1.7 V to 5.5 V) operation without external bypass capacitor, the VREF buffer should be operated in 2x power mode. However, the maximum sample rate supported without external bypass capacitor remains at 100 ksps. For a 1-Msps sample rate, an external bypass capacitor and an 18-MHz clock are required. See Table 21-8 for details.

Table 21-8. SAR VREF Power Options

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

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

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. Perform 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 21-9. DSI mode can be enabled by setting DSI_MODE bit (SAR_CTRL [29]).

Table 21-9. Difference between Control Modes

Control Mode	Register	DSI
DSI_MODE	0	1
SARMUX control	Sequencer control registers: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL SAR_CTRL Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL	DSI signal control signals: dsi_out, dsi_oe,dsi_swctrl, dsi_sw_negvref Firmware control registers: SAR_MUX_SWITCH0, SAR_MUX_HW_SWITCH_CTRL, SAR_CTRL
Global configuration	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND	Global configure registers: SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, SAR_RANGE_COND
Channel configuration	Channel configure registers: CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG	By DSI signal: dsi_cfg_st_sel, dsi_cfg_average, dsi_cfg_resolution, dsi_cfg_differential (CHAN_CONFIG, CHAN_EN, INJ_CHAN_CONFIG are ignored)
Trigger	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])	All Apply Firmware trigger (SAR_START_CTRL[0]) DSI trigger (dsi_trigger) Continuous trigger (SAR_SAMPLE_CTRL [0])

Table 21-9. Difference between Control Modes

Control Mode	Register	DSI
Interrupt	All Apply	All Apply (only EOS_INTR, RANGE_INTR, SATURATE_INTR output on DSI signal)
DSI output	Support	Support
Result data	8 channel result registers 1 injection channel result register	Only channel0 result register is available
Injection	Support	Not supported
Average	Support average on one PIN/signal	Support average on different PIN/signal

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 the PSoC 4200L Registers 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 21-10 shows the PORT_ADDR and PIN_ADDR setup with corresponding SARMUX selection. The unused port/pins are reserved for other products in the PSoC 4 series.

Table 21-10. PORT_ADDR and PIN_ADDR

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

a. sarbus0 and sarbus1 connect to the output of the CTBm block, which contains opamp0/1. See the Continuous Time Block mini (CTBm) chapter on page 291 for more information. When PORT_ADDR=1, sarbus0 connects to positive terminal of SAR ADC regardless of the value of PIN_ADDR; sarbus1 can only connect to the negative terminal of SAR ADC when differential mode is enabled and PORT_ADDR=1.

For differential conversion, the negative terminal connection is dependent on the positive terminal connection, which is defined by PORT_ADDR and PIN_ADDR. By setting DIFFERENTIAL_EN, the channel will do a differential conversion on the even/odd pin pair specified by the pin address with PIN_ADDR [0] ignored. P2.0/P2.1, P2.2/P2.3, P2.4/P2.5, P2.6/P2.7 are valid differential pairs for sequencer control. More flexible analog can be implemented by firmware or DSI.

For single-ended conversions, NEG_SEL (SAR_CTRL [11:9]) is intended to decide which signal is connected to negative input. In differential mode, these bits are ignored. Negative input choice affects the input voltage range and effective resolution. See Negative Input Selection on page 255 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 255 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 21-11. 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 21-12. 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 21-13. 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 action does not affect the ongoing scan. Note that this is an exception to the rule; all other configurations (global or channel) should not be changed while a scan is in progress.

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.

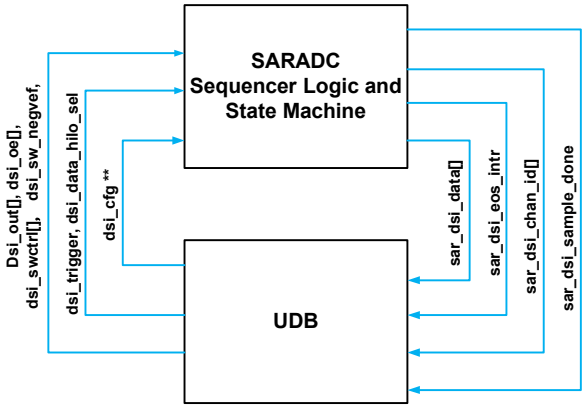
21.3.10.8 Injection Conversions

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 21-14 is a subset of the SAR ADC block diagram (Figure 21-1), which specifies the DSI input and output signals.

Figure 21-14. 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 21-14. DSI Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging, if available) for one channel; the internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0]. If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>.
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_out	8	dsi_out[0]=1, P2.0 connected to ADC dsi_out[1]=1, P2.1 connected to ADC … dsi_out[7]=1, P2.7 connected to ADC **Note** MUX_SWITCH0 configuration determines whether the pin is connected to vplus or vminus.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, opamp0 output connected to ADC dsi_oe[3]=1, opamp1 output connected to ADC **Note** MUX_SWITCH0 configuration determines whether the signal is connected to vplus or vminus.

Table 21-14. DSI Signals

Signal	Width	Description
dsi_swctrl[0]	1	SARMUX analog switch control, connect vssa_kelvin to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temp_sens to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input
dsi_cfg_st_sel	2	Configuration control for DSI control mode: select 1 of 4 global sample times
dsi_cfg_average	1	Configuration control for DSI control mode: enable averaging
dsi_cfg_resolution	1	Configuration control for DSI control mode: 0=12-bit resolution 1=use globally 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 21-15 shows the DSI signals.

Table 21-15. DSI Analog Routing

Signal	Width	Description
dsi_out	8	dsi_out[0]=1, P2.0 connected to ADC dsi_out[1]=1, P2.1 connected to ADC … dsi_out[7]=1, P2.7 connected to ADC **Note** Whether the pin is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_oe	4	dsi_oe[0]=1, AMUXBUSA connected to ADC dsi_oe[1]=1, AMUXBUSB connected to ADC dsi_oe[2]=1, sarbus0 output connected to ADC dsi_oe[3]=1, sarbus1 output connected to ADC **Note** Whether the signal is connected to vplus or vminus is determined by MUX_SWITCH0 configuration.
dsi_swctrl[0]	1	SARMUX analog switch control, connect V_{SSA} to vminus
dsi_swctrl[1]	1	SARMUX analog switch control, connect temperature sensor to vplus
dsi_sw_negvref	1	SAR ADC internal switch control, connect V_{REF} input to NEG input

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 21-16. CHAN_EN and channel configurations in CHAN_CONFIG and INJ_CHAN_CONFIG are ignored.

The dsi_cfg_* signals can optionally be synchronized to the SAR clock domain (actually clk_hf) by setting DSI_SYNC_CONFIG. Bypassing synchronization may be required when running the SAR at a low frequency.

Table 21-16. 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 276. 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 21-17 lists the interrupts that are sent via DSI signals.

Table 21-17. DSI Signal Interrupts

Signal	Width	Description
sar_dsi_chan_id	4	Register mode: Channel ID (ID of the channel that is currently being converted) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_eos_intr	1	End-of-scan interrupt to indicate that the SARSEQ has finished a scan of all enabled channels

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 270 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 21-18. DSI Output Signals

Signal	Width	Description
sar_dsi_sample_done	1	Pulse to indicate that SAR ADC sampling is done. Switches can be changed to the next signal that need to be converted (identical to SAR ADC next output)
sar_dsi_chan_id_valid	1	Valid signal for channel ID
sar_dsi_chan_id	4	Regular mode: Channel ID, ID of the channel that is currently being converted (early) DSI control mode: [0]=saturation detect interrupt [1]=range detect interrupt (valid together with data output)
sar_dsi_data_valid	1	Valid signal for data value
sar_dsi_data	12	Result of converting (and averaging if there is) for one channel. The internal averaging result is 16-bit wide. If dsi_data_hilo_sel=0 then sar_dsi_data[11:0]= sar_data[11:0] If dsi_data_hilo_sel=1 then sar_dsi_data[7:0]= sar_data[15:8] and sar_dsi_data[11:8]=<undefined>
sar_dsi_eos_intr	1	End-Of-Scan interrupt to indicate that SARSEQ just finished a scan of all enabled channels
dsi_data_hilo_sel	1	Selects between high and low byte output for sar_dsi_data[7:0]. This signal is fully asynchronous (affects sar_dsi_data without any clock involved)

21.3.12 Analog Routing Configuration Example

Table 21-19 shows some examples of pin and signal selection for sequencer control, firmware control, and DSI control.

Table 21-19. Analog Routing Configuration Example

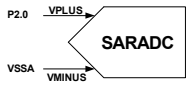
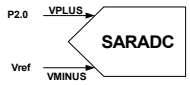
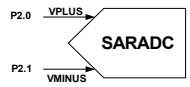
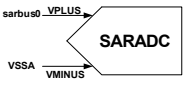
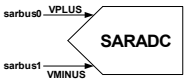
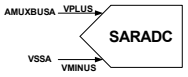
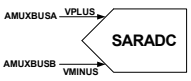
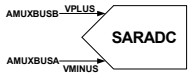
	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_out [0] =1 dsi_swctrl[0]=1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 (CHANx_CONFIG[2:0]) NEG_SEL = 7 (CTRL [11:9]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0]=1 HW_CTRL_NEGVREF =1 (CTRL[13])	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] =0 NEG_SEL = 7 (CTRL [11:9]) HW_CTRL_NEGVREF =0 (CTRL[13])	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 dsi_out [0] =1 dsi_sw_negvref =1 HW_CTRL_NEGVREF =1 (CTRL[13])
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 0 (CHANx_CONFIG[6:4]) PIN_ADDR = 0 or PIN_ADDR = 1 (CHANx_CONFIG[2:0]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH_HW_CTRL[1] = 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[0] = 1 MUX_SWITCH0[9] = 1 MUX_SWITCH_HW_CTRL[0] = 0 MUX_SWITCH_HW_CTRL[1] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_out [0] =1 dsi_out [1] =1 MUX_SWITCH0[0] = 1 MUX_SWITCH_HW_CTRL[0] = 1 MUX_SWITCH0 [9] = 1 MUX_SWITCH_HW_CTRL[1]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 0 (CTRL[30]) PORT_ADDR = 1 (CHANx_CONFIG[6:4]) NEG_SEL = 0 (CTRL [11:9]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] =1 MUX_SWITCH_HW_CTRL[16] =1 **Note** Connecting sarbus1 to VPLUS is not supported for Port/Pin control	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[16] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [2] =1 dsi_swctrl[0]=1 MUX_SWITCH0 [16] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH_HW_CTRL[22] =1

Table 21-19. Analog Routing Configuration Example<Italic> (continued)

	Sequencer Control	Firmware Control	DSI Control
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 1 **(**CHANx_CONFIG[6:4]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) SWITCH_DISABLE = 1 (CTRL[30]) MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22] = 0 MUX_SWITCH_HW_CTRL[23] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [2] = 1 dsi_oe [3] = 1 MUX_SWITCH0[22] = 1 MUX_SWITCH0[25] = 1 MUX_SWITCH_HW_CTRL[22]=1 MUX_SWITCH_HW_CTRL[23]=1
	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** **NEG_SEL = 0** (CTRL [11:9]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]= 1	DIFFERENTIAL_EN = 0 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[16] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[16]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 0 dsi_oe [0] = 1 dsi_swctrl[0]=1 MUX_SWITCH0[18] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[16]=1 MUX_SWITCH0 [16] = 1
	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 0** (CTRL[30]) PORT_ADDR = 7 **(**CHANx_CONFIG[6:4]) PIN_ADDR = 2 **(**CHANx_CONFIG[2:0]**)** MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 0 MUX_SWITCH_HW_CTRL[19]= 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[18] = 1 MUX_SWITCH0[21] = 1 MUX_SWITCH_HW_CTRL[18]= 1 MUX_SWITCH_HW_CTRL[19]= 1
	Not supported. The differential pair is fixed for Port/Pin control	DIFFERENTIAL_EN = 1 (CHANx_CONFIG[8]) **SWITCH_DISABLE = 1** (CTRL[30]) MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =0 MUX_SWITCH_HW_CTRL[19] = 0	DSI_MODE = 1 (CTRL[29]) dsi_cfg_differential = 1 dsi_oe [0] = 1 dsi_oe [1] = 1 MUX_SWITCH0[19] = 1 MUX_SWITCH0[20] = 1 MUX_SWITCH_HW_CTRL[18] =1 MUX_SWITCH_HW_CTRL[19] = 1

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 21-20 lists the methods to route temperature sensors to SAR ADC. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit disables temperature sensor by cutting its bias current.

Table 21-20. Route Temperature to SAR ADC

Control Methods	Setup
Sequencer	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4]) PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0]) SWITCH_DISABLE = 0 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
Firmware	DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8]) VREF_SEL = 0 (SAR_CTRL[6:4]) SWITCH_DISABLE = 1 (SAR_CTRL[30]) SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 0 SAR_MUX_SWITCH_HW_CTRL[17]= 0 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]
DSI	SWITCH_DISABLE = 1 (SAR_CTRL[30]) VREF_SEL = 0 (SAR_CTRL[6:4]) Set DSI Signals: dsi_cfg_differential=1 dsi_swctrl[1]=1 dsi_swctrl[0]=1 SAR_MUX_SWITCH0[16] = 1 SAR_MUX_SWITCH0[17] = 1 SAR_MUX_SWITCH_HW_CTRL[16]= 1 SAR_MUX_SWITCH_HW_CTRL[17]= 1 NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a]

a. For temperature sensor, override NEL_SEG (SAR_CTRL [11:9]) to '0'.

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. Low-Power Comparator

PSoC® 4 devices have two low-power comparators. These comparators can perform fast analog signal comparison in all system power modes except the Stop mode. Refer to the Power Modes chapter on page 91 for details on various device power modes. The positive and negative inputs can be connected to dedicated GPIO pins or to AMUXBUS-A/AMUXBUS-B. The comparator output can be read by the CPU through a status register, used as an interrupt or wakeup source, fed to the DSI for processing or routed to a GPIO.

22.1 Features

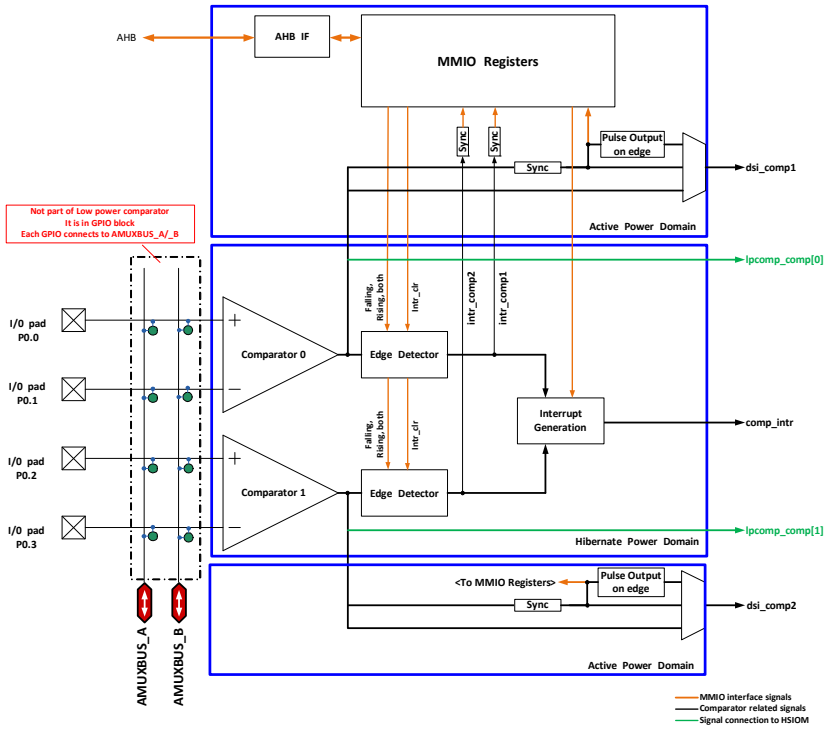
PSoC 4 comparators have the following features:

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

22.2 Block Diagram

Figure 22-1 shows the block diagram for the low-power comparator.

Figure 22-1. Low-Power Comparator Block Diagram



22.3 How It Works

The following sections describe the operation of the PSoC 4 low-power comparator, including input configuration, power and speed mode, output and interrupt configuration, hysteresis, wake up from low-power modes, comparator clock, and offset trim.

22.3.1 Input Configuration

Inputs to the comparators can be as follows:

- Both positive and negative inputs from dedicated input pins.
- Both positive and negative inputs from any pin through AMUXBUS (not available in Deep-Sleep mode).
- One input from an external pin and another input from an internally-generated signal. Both inputs can be connected to either positive or negative inputs of the comparator. The internally-generated signal is connected to the comparator input through the analog AMUXBUS.
- Both positive and negative inputs from internally-generated signals. The internally-generated signals are connected to the comparator input through AMUXBUS-A/ AMUXBUS-B.

From Figure 22-1, note that P0.0 and P0.1 connect to positive and negative inputs of Comparator 0; P0.2 and P0.3 connect to the inputs of Comparator 1. Also, note that the AMUXBUS nets do not have a direct connection to the comparator inputs. Therefore, the comparator connection is routed to the AMUXBUS nets through the corresponding input pin. These input pins will not be available for other purposes when using AMUXBUS for comparator connections. They should be left open in designs that use AMUXBUS for comparator input connection. Note that AMUXBUS connections are not available in Deep-Sleep and Hibernate modes. If Deep-Sleep or Hibernate operation is required, the low-power comparator must be connected to the dedicated pins. This restriction also includes routing of any internally-generated signal, which uses the AMUXBUS for the connection. See the I/O System chapter on page 55 for more details on connecting the GPIO to AMUXBUS A/B or setting up the GPIO for comparator input.

22.3.2 Output and Interrupt Configuration

The output of Comparator0 and Comparator1 are available in the OUT1 bit [6] and OUT2 bit [14], respectively, in the LPCOMP_CONFIG register (Table 22-1). The comparator outputs are synchronized to SYSCLK before latching them to the OUTx bits in the LPCOMP_CONFIG register. The out-

put of each comparator is connected to a corresponding edge detector block. This block determines the edge that triggers the interrupt. The edge selection and interrupt enable is configured using the INTTYPE1 bits [5:4] and INTTYPE2 bits [13:12] in the LPCOMP_CONFIG register. Using the INTTYPEx bits, the interrupt type can be selected to disabled, rising edge, falling edge, or both edges, as described in Table 22-1.

Each comparator's output can be routed directly to a GPIO pin through the HSIOM. The comparator outputs are available as Deep-Sleep source 2 connection in the HSIOM. See High-Speed I/O Matrix on page 66 for details on HSIOM. For details on the pins that support the low-power comparator output, refer to the device datasheet. The output on these pins are direct output from the comparator and are not synchronized. Because they act as Deep-Sleep source for the pins, the comparator output is available in Deep-Sleep power mode as well. Note that the HSIOM is not available in Hibernate power mode. Hence, the comparator output on the supported pins will not be available in Hibernate mode.

In addition, the comparator outputs can be individually routed to a pin or other blocks through DSI (dsi_comp1/dsi_comp2 signals in Figure 22-1). The comparator output to the DSI can be configured to be asynchronous, synchronized to SYSCLK, or a synchronized pulse output on the comparator output's rising edge. The DSI_BYPASS1 bit [16] and DSI_LEVEL1 bit [17] of the LPCOMP_CONFIG register are used to configure Comparator0's output to DSI. Similarly, DSI_BYPASS2 bit [20] and DSI_LEVEL2 bit [21] of the LPCOMP_CONFIG register are used for Comparator1 DSI output. See Table 22-1 for details.

During an edge event, the comparator will trigger an interrupt (intr_comp1/intr_comp2 signals in Figure 22-1). The interrupt request is registered in the COMP1 bit [0] and COMP2 bit [1] of the LPCOMP_INTR register for Comparator0 and Comparator1, respectively. Both Comparator0 and Comparator1 share a common interrupt (comp_intr signal in Figure 22-1), which is a logical OR of the two interrupts and mapped as the low-power comparator block's interrupt in the CPU NVIC. Refer to the Interrupts chapter on page 45 for details. If both the comparators are used in a design, the COMP1 and/or COMP2 bits of the LPCOMP_INTR register need to be read in the interrupt service routine to know which one triggered the interrupt. Alternatively, COMP1_MASK bit [0] and COMP2_MASK bit [1] of the LPCOMP_INTR_MASK register can be used to mask the Comparator0 and Comparator1 interrupts to the CPU. Only the masked interrupts will be serviced by the CPU. After the interrupt is processed, the interrupt should be cleared by writing a '1' to the COMP1 and COMP2 bits of the LPCOMP_INTR register in firmware. If the interrupt is not cleared, the next compare event will not trigger an interrupt and the CPU will not be able to process the event. In Active and Sleep modes, the dsi_comp1/dsi_comp2 outputs can be routed to UDB mapped interrupts for processing each comparator's trigger separately. However, the UDB/DSI routing is not available in Deep-Sleep and Hibernate modes.

The LPCOMP interrupt (comp1_intr/comp2_intr) is synchronous with SYSCLK. Clearing dsi_comp1/dsi_comp2 and comp1_intr/comp2_intr are all synchronous.

In Active and Sleep modes, dsi_comp1/dsi_comp2 can be routed to GPIO or other blocks through DSI routing in UDB with or without synchronization; there is an additional synchronizer on UDB DSI output. See the Universal Digital Blocks (UDB) chapter on page 148 for details on the DSI signal synchronization. In Deep-Sleep and Hibernate modes, this routing is unavailable because the UDBs are powered off. In addition, if the dsi_comp1/dsi_comp2 is routed to the UDB for further processing, the timing depends on the user's algorithm and synchronizer choice.

LPCOMP_INTR_SET register bits [1:0] can be used to assert an interrupt for software debugging.

In Deep-Sleep and Hibernate mode, the wakeup interrupt controller (WIC) can be activated by a comparator edge event, which then wakes up the CPU. Thus, the LPCOMP has the capability to monitor a specified signal in low-power modes.

Table 22-1. Output and Interrupt Configuration in LPCOMP_CONFIG Register

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[6]	OUT1	Current/Instantaneous output value of Comparator0
LPCOMP_CONFIG[14]	OUT2	Current/Instantaneous output value of Comparator1
LPCOMP_CONFIG[5:4]	INTTYPE1	Sets on which edge Comparator0 will trigger an IRQ 00: Disabled 01: Rising Edge 10: Falling Edge 11: Both rising and falling edges
LPCOMP_CONFIG[13:12]	INTTYPE2	Sets on which edge Comparator1 will trigger an IRQ 00: Disabled 01: Rising Edge 10: Falling Edge 11: Both rising and falling edges
LPCOMP_CONFIG[16]	DSI_BYPASS1	Comparator0 bypass output synchronization for DSI output 0: Output synchronized to SYSCLK 1: Bypass/output asynchronous
LPCOMP_CONFIG[17]	DSI_LEVEL1	Comparator0 DSI output level 0: Pulse output - Generates a pulse of width two SYSCLK cycles on a rising edge 1: Level
LPCOMP_CONFIG[20]	DSI_BYPASS2	Comparator1 bypass output synchronization for DSI output 0: Output synchronized to SYSCLK 1: Bypass/output asynchronous
LPCOMP_CONFIG[21]	DSI_LEVEL2	Comparator1 DSI output level 0: Pulse output - Generates a pulse of width two SYSCLK cycles on a rising edge 1: Level
LPCOMP_INTR[0]	COMP1	Comparator0 Interrupt: hardware sets this interrupt when Comparator0 triggers. Write a '1' to clear the interrupt
LPCOMP_INTR[1]	COMP2	Comparator2 Interrupt: hardware sets this interrupt when Comparator1 triggers. Write a '1' to clear the interrupt
LPCOMP_INTR_SET[0]	COMP1	Write a '1' to trigger the software interrupt for Comparator0
LPCOMP_INTR_SET[1]	COMP2	Write a 1 to trigger the software interrupt for Comparator1

22.3.3 Power Mode and Speed Configuration

The low-power comparators can operate in three power modes:

- Fast
- Slow
- Ultra low-power

The power or speed setting for Comparator0 is configured using MODE1 bits [1:0] in the LPCOMP_CONFIG register. The power or speed setting for Comparator1 is configured using MODE2 bits [9:8] in the same register. The power consumption and response time vary depending on the selected power mode; power consumption is highest in fast mode and lowest in ultra-low-power mode, response time is fastest in fast mode and slowest in ultra-low-power mode. Refer to the device data-sheet for specifications for the response time and power consumption for various power settings.

The comparators are enabled/disabled using ENABLE1 bit [7] and ENABLE2 bit [15] in the LPCOMP_CONFIG register, as described in Table 22-2.

Note The output of the comparator may glitch when the power mode is changed while comparator is enabled. To avoid this, disable the comparator before changing the power mode.

Table 22-2. Comparator Power Mode Selection Bits MODE1 and MODE2

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[1:0]	MODE1	Compartor0 power mode selection 00: Slow operating mode (uses less power) 01: Fast operating mode (uses more power) 10: Ultra low-power operating mode (uses lowest possible power)
LPCOMP_CONFIG[9:8]	MODE2	Compartor1 power mode selection 00: Slow operating mode (uses less power) 01: Fast operating mode (uses more power) 10: Ultra low-power operating mode (uses lowest possible power)
LPCOMP_CONFIG[7]	ENABLE1	Comparator0 enable bit 0: Disables Comparator0 1: Enables Comparator0
LPCOMP_CONFIG[15]	ENABLE2	Comparator1 enable bit 0: Disables Comparator1 1: Enables Comparator1

22.3.4 Hysteresis

For applications that compare signals close to each other and slow changing signals, hysteresis helps to avoid oscillations at the comparator output when the signals are noisy. For such applications, a fixed 10-mV hysteresis may be enabled in the comparator block.

The 10-mV hysteresis level is enabled/disabled by using the HYST1 bit [2] and HYST2 bit [10] in the LPCOMP_CONFIG register, as described in Table 22-3.

Table 22-3. Hysteresis Control Bits HYST1 and HYST2

Register[Bit_Pos]	Bit_Name	Description
LPCOMP_CONFIG[2]	HYST1	Enable/Disable 10 mV hysteresis to Comparator0 - 0: Enable Hysteresis - 1: Disable Hysteresis
LPCOMP_CONFIG[10]	HYST2	Enable/Disable 10 mV hysteresis to Comparator1 - 0: Enable Hysteresis - 1: Disable Hysteresis

22.3.5 Wakeup from Low-Power Modes

The comparator is operational in the device's low-power modes, including Sleep, Deep-Sleep, and Hibernate modes. The comparator output interrupt can wake the device from Sleep, Deep-Sleep, and Hibernate modes. The comparator should be enabled in the LPCOMP_CONFIG register, the INTTYPEx bits in the LPCOMP_CONFIG register should not be set to disabled, and the INTR_MASKx bit should be set in the LPCOMP_INTR_MASK register for the corresponding comparator to wake the device from low-power modes. The features that are not available during the Deep-Sleep and Hibernate modes include:

- Comparisons involving AMUXBUS connections
- Routing comparator output through DSI

In the Deep-Sleep or Hibernate power mode, a compare event on either Comparator0 or Comparator1 output will generate a wakeup interrupt. The INTTYPEx bits in the LPCOMP_CONFIG register should be configured, as required, for the corresponding comparator to wake the device from low-power modes. The mask bits in the LPCOMP_INTR_MASK register is used to select whether one or both of the comparator's interrupt is serviced by the CPU.

22.3.6 Comparator Clock

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

22.3.7 Offset Trim

The comparator offset is trimmed at the factory to less than 4.0 mV. The trim is a two-step process, trimmed first at common mode voltage equal to 0.1 V, then at common mode

voltage equal to V_{DD}–0.1 V. Offset voltage is guaranteed to be less than 10.0 mV over the input voltage range of 0.1 V to V_{DD}–0.1 V. For normal operation, further adjustment of trim values is not recommended.

If a tighter trim is required at a specific input common mode voltage, a trim may be performed at the desired input common mode voltage. The comparator offset trim is performed using the LPCOMP_TRIM1/2/3/4 registers. LPCOMP_TRIM1 and LPCOMP_TRIM2 are used to trim comparator 0. LPCOMP_TRIM3 and LPCOMP_TRIM4 are used to trim comparator 1. The bit fields that change the trim values are TRIMA bits [4:0] in LPCOMP_TRIM1 and LPCOMP_TRIM3, and TRIMB bits [3:0] in LPCOMP_TRIM2 and LPCOMP_TRIM4. TRIMA bits are used to coarse tune the offset; TRIMB bits are used to fine tune. The use of TRIMB bits for offset correction is restricted to slow mode of comparator operation.

Any standard comparator offset trim procedure can be used to perform the trimming. The following method can be used to improve the offset at a given reference/common mode voltage input.

1. Short the comparator inputs externally and connect the voltage reference, V_{ref}, to the input.
2. Set up the comparator for comparison, turn off hysteresis, and check the output.
3. If the output is high, the offset is positive. Otherwise, the offset is negative. Follow these steps to tune the offset:
   a. Tune the TRIMA bits[4:0] until the output switches direction. TRIMA bits[3:0] control the amount of offset and TRIMA bit[4] controls the polarity of offset ('1' indicates positive offset and '0' indicates negative offset).
   b. When the tuning of TRIMA bits is complete, tune the TRIMB bits[3:0] until the output switches direction again. The TRIMB bit tuning is valid only for slow mode of comparator operation. TRIMB bit[3] controls the polarity of offset. Increasing TRIMB bits [2:0] reduces the offset.
   c. After completing step 3-b, the values available in the TRIMA and TRIMB bits will be the closest possible trim value for that particular V_{ref}.

22.4 Register Summary

Table 22-4. Low-Power Comparator Register Summary

Register	Function
LPCOMP_ID	Includes the information of LPCOMP controller ID and revision number
LPCOMP_CONFIG	LPCOMP configuration register
LPCOMP_INTR	LPCOMP interrupt register
LPCOMP_INTR_SET	LPCOMP interrupt set register
LPCOMP_INTR_MASK	LPCOMP interrupt request mask register
LPCOMP_INTR_MASKED	LPCOMP masked interrupt output register
LPCOMP_TRIM1	Trim fields for comparator 0
LPCOMP_TRIM2	Trim fields for comparator 0
LPCOMP_TRIM3	Trim fields for comparator 1
LPCOMP_TRIM4	Trim fields for comparator 1

23. Continuous Time Block mini (CTBm)

The Continuous Time Block mini (CTBm) provides discrete operational amplifiers (opamps) inside the chip for use in continuous-time signal chains. Each CTBm block includes a switch matrix for input/output configuration, two identical opamps, which are also configurable as two comparators, a charge pump inside each opamp, and a digital interface for comparator output routing, switch controls, and interrupts. The PSoC 4200L family has two CTBm blocks - four discrete opamps. Additionally, the CTBm blocks can also operate in Deep-Sleep power mode.

23.1 Features

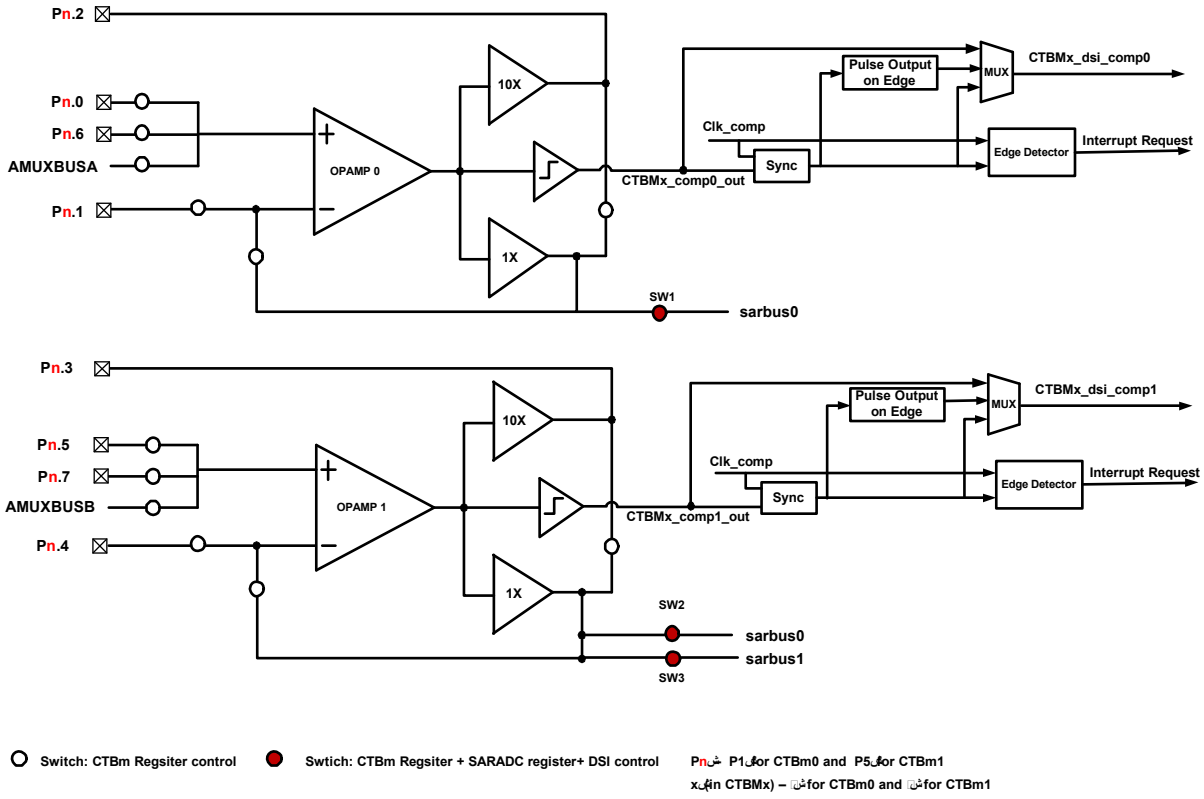
The opamps in the PSoC 4 CTBm block have the following features:

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

23.2 Block Diagram

Figure 23-1 shows the block diagram for the CTBmx - CTBm0 and CTBm1 block available in PSoC 4 devices.

Figure 23-1. CTBm Block Diagram



23.3 How It Works

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

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

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

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

Follow these steps to set up a comparator:

1. Configure the power mode
2. Configure the input switch

3. Configure the comparator output circuitry, as required - interrupt generation, output, and so on
4. Configure hysteresis and enable the comparator

23.3.1 Power Mode Configuration

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

Note: 'y' denotes Opamp1/0 (y = 1 for Opamp1 and 0 for Opamp0) in CTBMx.

23.3.2 Output Strength Configuration

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

Table 23-1. Output Driver versus Power Mode

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

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

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

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

Register[Bit_Pos]	Bit_Name	Description
CTBMx_CTB_CTRL[31]	ENABLE	CTBM power mode selection 0: CTBM is disabled 1: CTBM is enabled
CTBMx_OA_RES0_CTRL [11]	OA0_PUMP_EN	Opamp0 pump enable bit 0: Opamp0 pump is disabled in CTBMx 1: Opamp0 pump is enabled in CTBMx
CTBMx_OA_RES1_CTRL [11]	OA1_PUMP_EN	Opamp1 pump enable bit 0: Opamp1 pump is disabled in CTBMx 1: Opamp1 pump is enabled in CTBMx

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

Register[Bit_Pos]	Bit_Name	Description
CTBMx_OA_RES0_CTRL [1:0]	OA0_PWR_MODE	Opamp0 power mode select bits 00: Opamp0 is OFF in CTBMx 01: Opamp0 is in low power mode in CTBMx 10: Opamp0 is in medium power mode in CTBMx 11: Opamp0 is in high power mode in CTBMx
CTBMx_OA_RES1_CTRL [1:0]	OA1_PWR_MODE	Opamp1 power mode select bits 00: Opamp1 is OFF in CTBMx 01: Opamp1 is in low power mode in CTBMx 10: Opamp1 is in medium power mode in CTBMx 11: Opamp1 is in high power mode in CTBMx
CTBMx_OA_RES0_CTRL [2]	OA0_DRIVE_STR_SEL	Opamp0 output drive strength select bits 0: Opamp0 output drive strength is 1X in CTBMx 1: Opamp0 output drive strength is 10X in CTBMx
CTBMx_OA_RES1_CTRL [2]	OA1_DRIVE_STR_SEL	Opamp1 output drive strength select bits 0: Opamp1 output drive strength is 1X in CTBMx 1: Opamp1 output drive strength is 10X in CTBMx

23.3.3 Compensation

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

Table 23-3. Opampy (Opamp0 or Opamp1) Compensation Bits in CTBm

Register[Bit_Pos]	Bit_Name	Description
CTBMx_OAy_COMP_TRIM[1:0]	OAy_COMP_TRIM	Opampy compensation trim bits 00: No compensation 01: Minimum compensation, high speed, and low stability in CTBMx 10: Medium compensation, balanced speed, and stability in CTBMx 11: Maximum compensation, low speed, and high stability in CTBMx

23.3.4 Switch Control

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

Switches can be closed by setting the corresponding bit in register CTBMx_OAy_SW; clearing them will cause the corresponding switches to open. To open the switch, write '1' to CTBMx_OAy_SW_CLEAR, which clears the corresponding bit in CTBMx_OAy_SW. See the PSoC 4200L Registers TRM for details on the switches and the connections they enable.

23.3.4.1 Input Configuration

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

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

- Positive input: Both opamp0 and opamp1 of each CTBm have three positive input options through analog switches: two external pins and one AMUXBUS line. See Table 23-4 for details.

Table 23-4. Positive Input Selection

	Positive Input	Switch Control Bit	Description
Opamp0	AMUXBUSA	CTBMx_OA0_SW [0]	0: open 1: close switch
	Pn.0[a]	CTBMx_OA0_SW [2]	0: open 1: close switch
	Pn.6	CTBMx_OA0_SW [3]	0: open 1: close switch
Opamp1	AMUXBUSB	CTBMx_OA1_SW [0]	0: open 1: close switch
	Pn. 5	CTBMx_OA1_SW [1]	0: open 1: close switch
	Pn.7	CTBMx_OA1_SW [4]	0: open 1: close switch

a. Pn = P1 for CTBm0 and P5 for CTBm1

- Negative input: Both opamp0 and opamp1 of each CTBm have two negative input options through analog switches: one external pin or output feedback, which is controlled by the CTBMx_OAy_SW register. Table 23-5 shows the control bits.

Table 23-5. Negative Input Selection

	Negative Input	Switch Control Bit	Description
Opamp0	Pn.1[a]	CTBMx_OA0_SW [8]	0: open 1: close switch
	Opamp0 output feedback through 1X output driver	CTBMx_OA0_SW [14]	0: open 1: close switch
Opamp1	Pn.4	CTBMx_OA1_SW [8]	0: open 1: close switch
	Opamp1 output feedback through 1X output driver	CTBMx_OA1_SW [14]	0: open 1: close switch

a. Pn = P1 for CTBm0 and P5 for CTBm1

23.3.4.2 Output Configuration

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

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

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

Table 23-6. Truth Table of SW1 Control Logic

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

Table 23-7. Truth Table of SW2 Control Logic

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

Table 23-8. Truth Table of SW3 Control Logic

DIFFERENTIAL_EN	PORT_ADDR	PIN_ADDR	CTBMx_SW_HW_CTRL[3]	dsi_out[3]	CTBMx_OA0_SW[18]	SW3
X	X	X	X	X	0	0
X	X	0	1	0	1	0
X	0	X	1	0	1	0
0	X	X	X	0	1	0
X	X	X	0	X	1	1
X	X	X	X	X	1	1
1	1	2	X	X	1	1

23.3.4.3 Comparator Mode

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

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

23.3.4.4 Comparator Configuration

The hysteresis of 10 mV ±5 percent can be enabled in one direction (low to high). Input hysteresis can be enabled by setting CTBMx_OA_RESy_CTRL[5]. The two comparators in each CTBm block also have three power modes: low, medium, and high, controlled by setting CTBMx_OA_RESy_CTRL [1:0]. Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ultra-low-power mode. Exact specifications for power consumption and response time are provided in the datasheet.

The comparator output is routed to the DSI with optional synchronization. The synchronization with comparator clock (system AHB clock) can be configured in CTBMx_CTBM_OA_RESxy_CTRL[6].

The output state of comparator0 and comparator1 are stored in CTBMx_COMP_STAT[0] and CTBMx_COMP_STAT[16], respectively.

Table 23-9 summarizes various bits used to configure the comparator mode in the CTBM block.

Table 23-9. Comparator Mode and Configuration Register Settings

Register[Bit_Pos]	Bit_Name	Description
CTBMx_OA_RESyy_CTRL[4]	OAy_COMP_EN	Opampy comparator enable bit 0: Comparator mode is disabled in opampy in CTBMx 1: Comparator mode is enabled in opampy in CTBMx
CTBMx_OA_RESy_CTRL[5]	OAy_HYST_EN	Opampy Comparator hysteresis enable bit 0: Hysteresis is disabled in opampy in CTBMx 1: Hysteresis is enabled in opampy in CTBMx
CTBMx_OA_RESy_CTRL[6]	OAy_BYPASS_DSI_SYNC	Opampy bypass comparator output synchronization for DSI (trigger) output 0: Synchronize (level or pulse) 1: Bypass
CTBMx_OA_RESy_CTRL[7]	OAy_ DSI_LEVEL	Opampy comparator DSI (trigger) output synchronization level 0: Pulse 1: Level

23.3.4.5 Comparator Interrupt

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

Each comparator has a separate IRQ. CTBMx_INTR [0] is for comparator0 IRQ, CTBMx_INTR [1] is for comparator1 IRQ. Though each comparator of each CTBM have different IRQ bits, they all share a single CTBM ISR mapped in the CPU NVIC. See the Interrupts chapter on page 45 for details. You can check which CTBM's comparator(s) triggered the ISR by polling the CTBMx_INTR bits.

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

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

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

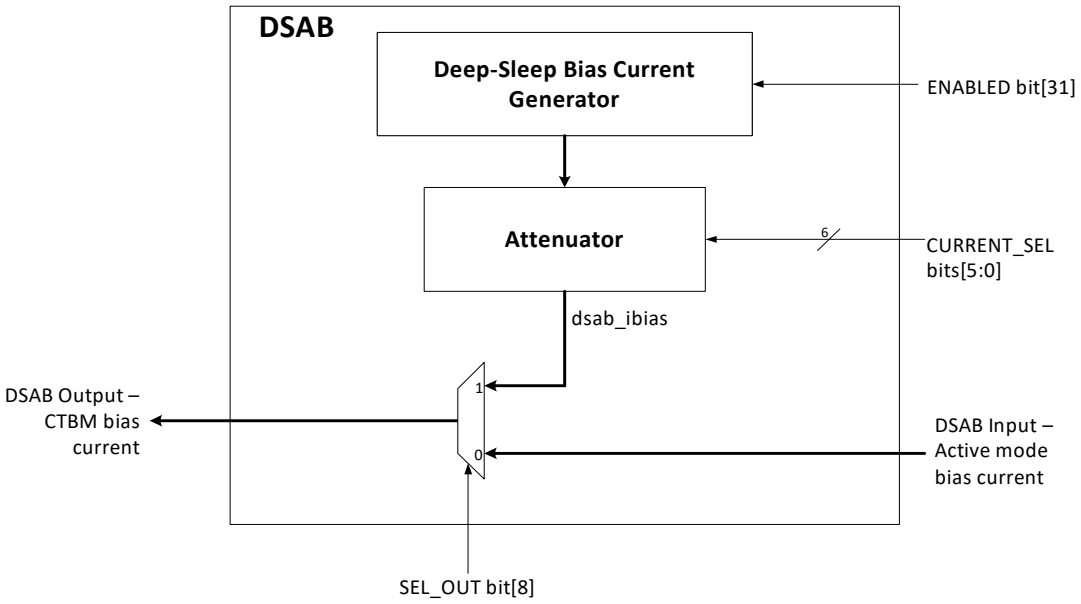
For verification and debug purposes, a set bit is provided for each interrupt in the CTBMx_INTR_SET register. This bit allows the firmware to raise the interrupt without a real comparator switch event.

23.3.4.6 Deep-Sleep Mode Operation

In Deep-Sleep mode, the block that provides the bias current, reference voltage, and IMO clock is turned off. As a result, the CTBm functionality, which relies on the bias current and IMO clock for its operation is not available. See the Power Modes chapter on page 91 for details on various power modes and blocks available in each mode. To support the functionality of the CTBm during deep sleep, an alternate bias current is generated by a special block called Deep-Sleep Amplifier Bias (DSAB) block. This current allows the opamps in the CTBm to be functional in Deep-Sleep mode.

Figure 23-2 shows the architecture of the DSAB block. This block receives the Active mode bias current as input. It outputs the bias current that is fed to the opamp bias circuitry. In Active mode, the DSAB block acts similar to a pass-through block and routes the bias current from the input to the output. In Deep-Sleep mode, if enabled, the DSAB generates the alternate bias current, attenuates the output to a user-selected value, and provides the bias current for the CTBm at its output. If the DSAB block is disabled, the output is always connected to the input bias current and the alternate bias current is not generated during deep sleep. The opamps will not be functional in Deep-Sleep mode, if the DSAB block is disabled. The ENABLED bit [31] of the PASS_DSAB_DSAB_CTRL register enables/disables the block; the CURRENT_SEL bits [5:0] selects the output bias current value. The value selected is CURRENT_SEL × 0.075 uA (±5 percent). The SEL_OUT bit[8] is used to control the selection between the two bias currents, which can be routed to the CTBm bias. Table 23-10 summarizes the bit configuration settings of the PASS_DSAB_DSAB_CTRL register.

Figure 23-2. Deep-Sleep Amplifier Bias Block Diagram



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

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

Table 23-10. DSAB and CTBM Deep-Sleep Configuration Register Settings

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

23.4 Register Summary

Table 23-11. Register Summary

Name	Description
CTBMx_CTRL	Global CTBm block enable
CTBMx_OA_RES0_CTRL	Opamp0 control register
CTBMx_OA_RES1_CTRL	Opamp1 control register
CTBMx_COMP_STAT	Comparator status
CTBMx_INTR	Interrupt request register
CTBMx_INTR_SET	Interrupt request set register
CTBMx_INTR_MASK	Interrupt request mask
CTBMx_INTR_MASKED	Interrupt request masked
CTBMx_OA0_SW	Opamp0 switch control
CTBMx_OA0_SW_CLEAR	Opamp0 switch control clear
CTBMx_OA1_SW	Opamp1 switch control
CTBMx_OA1_SW_CLEAR	Opamp1 switch control clear
CTBMx_SW_HW_CTRL	CTBm hardware control enable
CTBMx_SW_STATUS	CTBm bus switch control status
CTBMx_OA0_OFFSET_TRIM	Opamp0 trim control
CTBMx_OA0_SLOPE_OFFSET_TRIM	Opamp0 trim control
CTBMx_OA0_COMP_TRIM	Opamp0 trim control
CTBMx_OA1_OFFSET_TRIM	Opamp1 trim control
CTBMx_OA1_SLOPE_OFFSET_TRIM	Opamp1 trim control
CTBMx_OA1_COMP_TRIM	Opamp1 trim control
PASS_DSAB_DSAB_CTRL	DSAB control register
PASS_DSAB_TRIM	IBIAS trim register

24. LCD Direct Drive

The PSoC® 4 Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PSoC device to directly drive STN and TN segment LCDs.

24.1 Features

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

- Supports up to 56 segments and eight commons
- Supports Type A (standard) and Type B (low-power) drive waveforms
- Any GPIO can be configured as a common or segment
- Supports five drive methods:
  - Digital correlation
  - PWM at 1/2 bias
  - PWM at 1/3 bias
  - PWM at 1/4 bias
  - PWM at 1/5 bias
- Ability to drive 3-V displays from 1.8 V V_{DD} in Digital Correlation mode
- Operates in active, sleep, and deep-sleep modes
- Digital contrast control

24.2 LCD Segment Drive Overview

A segmented LCD panel has the liquid crystal material between two sets of electrodes and various polarization and reflector layers. The two electrodes of an individual segment are called commons (COM) or backplanes and segment electrodes (SEG). From an electrical perspective, an LCD segment can be considered as a capacitive load; the COM/SEG electrodes can be considered as the rows and columns in a matrix of segments. The opacity of an LCD segment is controlled by varying the root-mean-square (RMS) voltage across the corresponding COM/SEG pair.

The following terms/voltages are used in this chapter to describe LCD drive:

- **V_{RMSOFF}**: The voltage that the LCD driver can realize on segments that are intended to be off.
- **V_{RMSON}**: The voltage that the LCD driver can realize on segments that are intended to be on.
- **Discrimination Ratio (D)**: The ratio of V_{RMSON} and V_{RMSOFF} that the LCD driver can realize. This depends on the type of waveforms applied to the LCD panel. Higher discrimination ratio results in higher contrast.

Liquid crystal material does not tolerate long term exposure to DC voltage. Therefore, any waveforms applied to the panel must produce a 0-V DC component on every segment (on or off). Typically, LCD drivers apply waveforms to the COM and SEG electrodes that are generated by switching between multiple voltages. The following terms are used to define these waveforms:

- **Duty**: A driver is said to operate in 1/M duty when it drives 'M' number of COM electrodes. Each COM electrode is effectively driven 1/M of the time.
- **Bias**: A driver is said to use 1/B bias when its waveforms use voltage steps of (1/B) × VDRV. VDRV is the highest drive voltage in the system (equals to V_{DD} in PSoC 4). PSoC 4 supports 1/2, 1/3, 1/4, and 1/5 biases in PWM drive modes.
- **Frame**: A frame is the length of time required to drive all the segments. During a frame, the driver cycles through the commons in sequence. All segments receive 0-V DC (but non-zero RMS voltage) when measured over the entire frame.

PSoC 4 supports two different types of drive waveforms in all drive modes. These are:

- **Type-A Waveform**: In this type of waveform, the driver structures a frame into M sub-frames. 'M' is the number of COM electrodes. Each COM is addressed only once during a frame. For example, COM[i] is addressed in sub-frame i.
- **Type-B Waveform**: The driver structures a frame into 2M sub-frames. The two sub-frames are inverses of each other. Each COM is addressed twice during a frame. For example, COM[i] is addressed in sub-frames i and M+i. Type-B waveforms are slightly more power efficient because it contains fewer transitions per frame.

24.2.1 Drive Modes

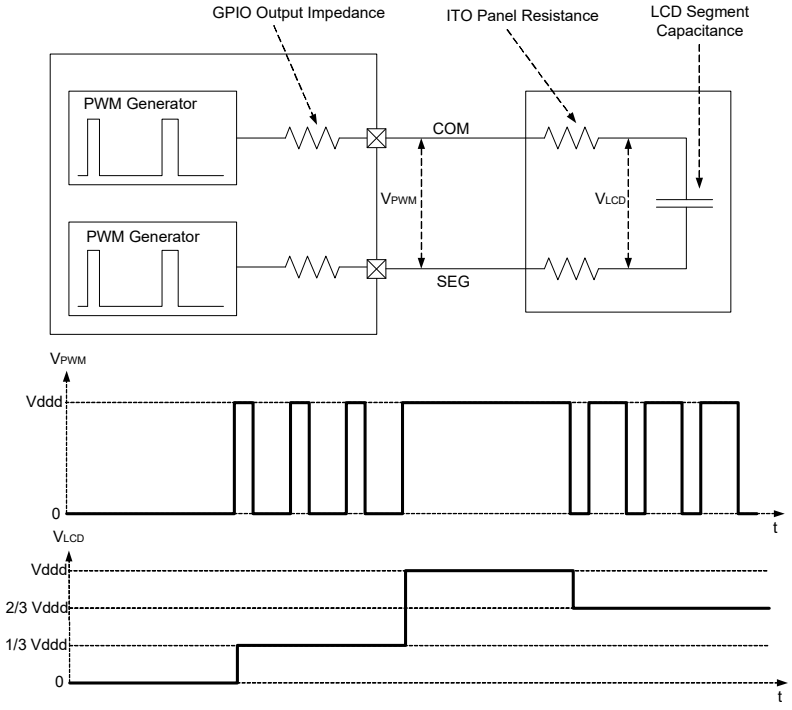
PSoC 4 supports the following drive modes.

- PWM drive at 1/2 bias
- PWM drive at 1/3 bias
- PWM drive at 1/4 bias with high-frequency clock input
- PWM drive at 1/5 bias with high-frequency clock input
- Digital correlation

24.2.1.1 PWM Drive

In PWM drive mode, multi-voltage drive signals are generated using a PWM output signal together with the intrinsic resistance and capacitance of the LCD. Figure 24-1 illustrates this.

Figure 24-1. PWM Drive (at 1/3 Bias)



The output waveform of the drive electronics is a PWM waveform. With the Indium Tin Oxide (ITO) panel resistance and the segment capacitance to filter the PWM, the voltage across the LCD segment is an analog voltage, as shown in Figure 24-1. This figure illustrates the generation of a 1/3 bias waveform (four commons and voltage steps of $V_{DD}/3$).

The PWM is derived from either ILO (32 kHz, low-speed operation) or IMO (high-speed operation). The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 24-2 and Figure 24-3 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/2 bias and 1/4 duty. Only COM0/COM1 and SEG0/SEG1 are drawn for demonstration purpose. Similarly, Figure 24-4 and Figure 24-5 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/3 bias and 1/4 duty.

Figure 24-2. PWM1/2 Type-A Waveform Example

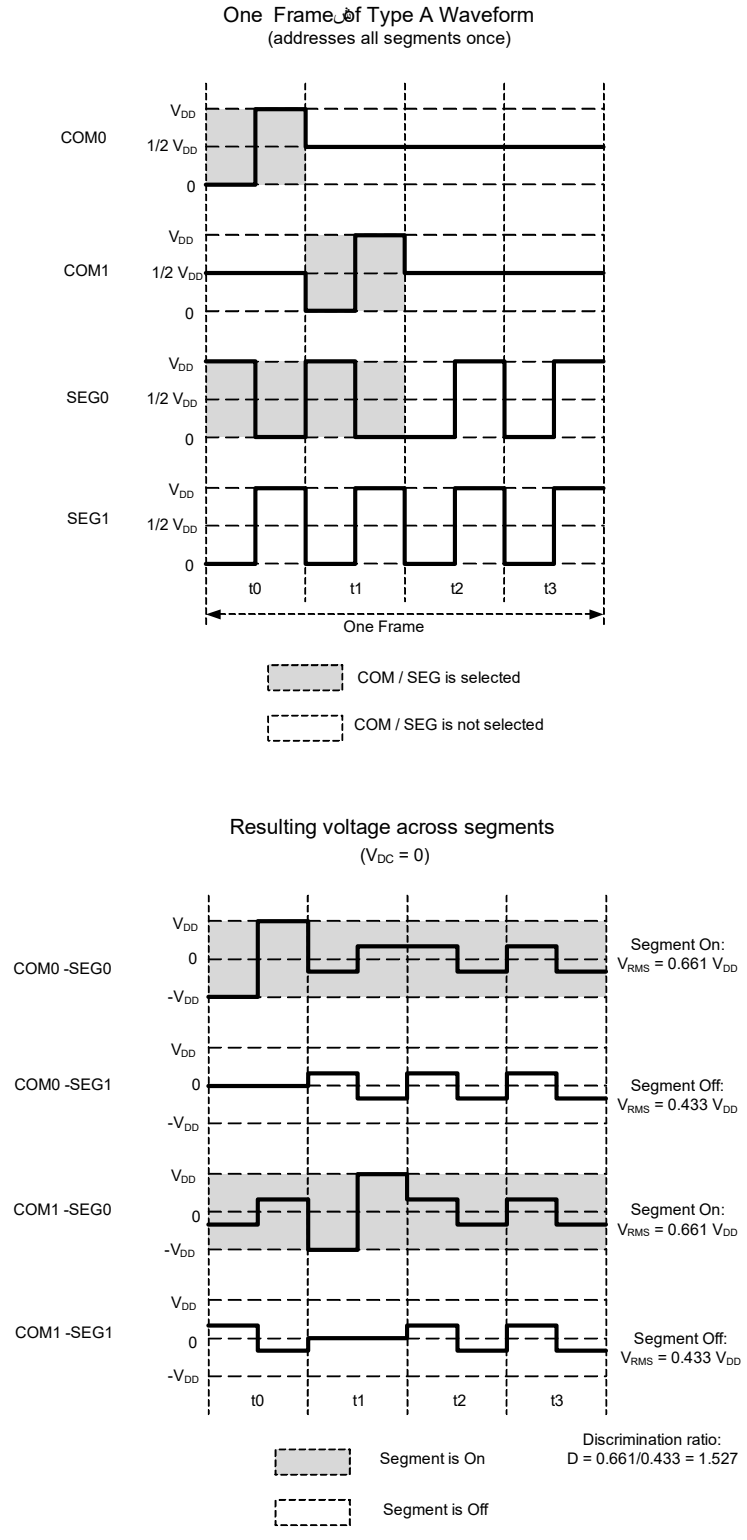


Figure 24-3. PWM1/2 Type-B Waveform Example

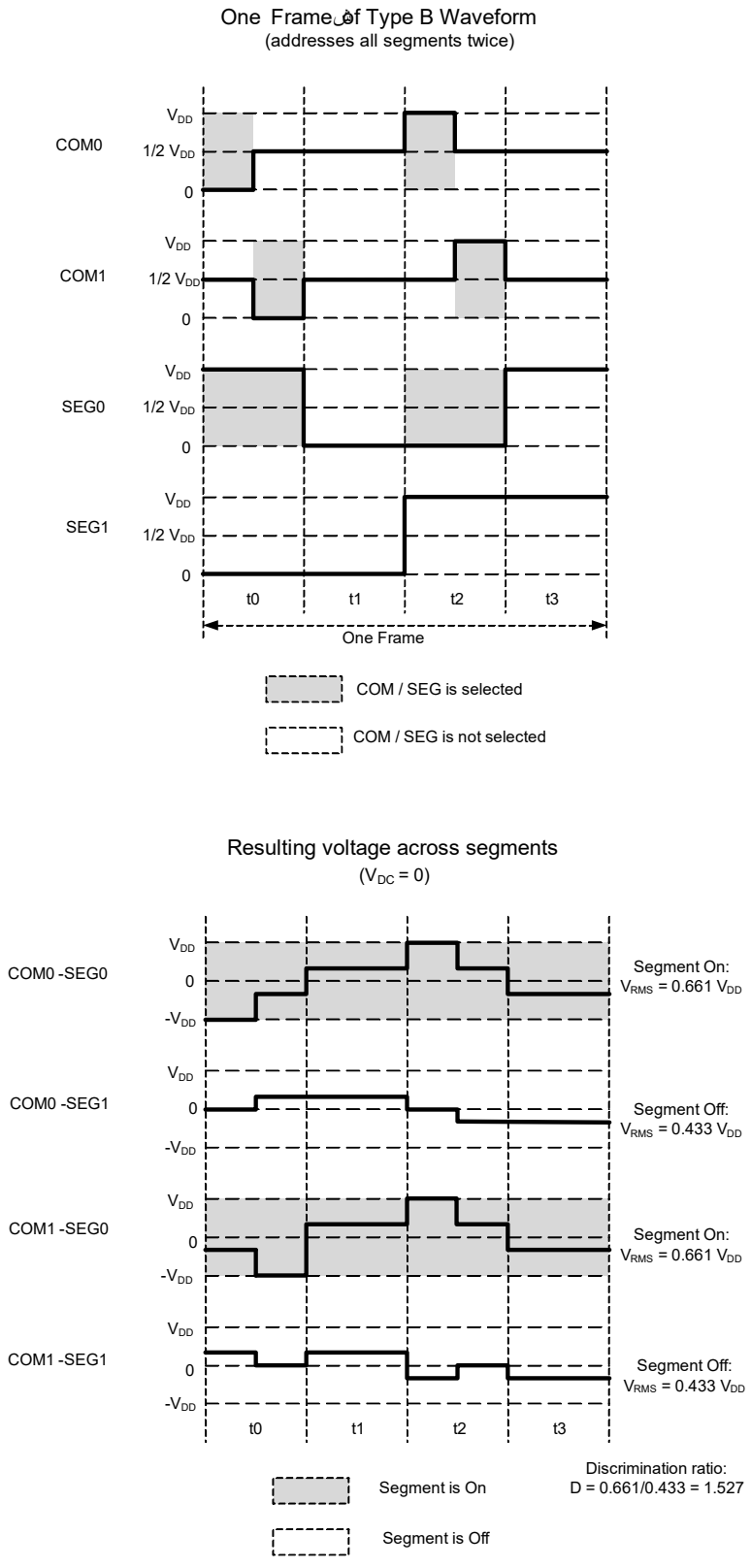


Figure 24-4. PWM1/3 Type-A Waveform Example

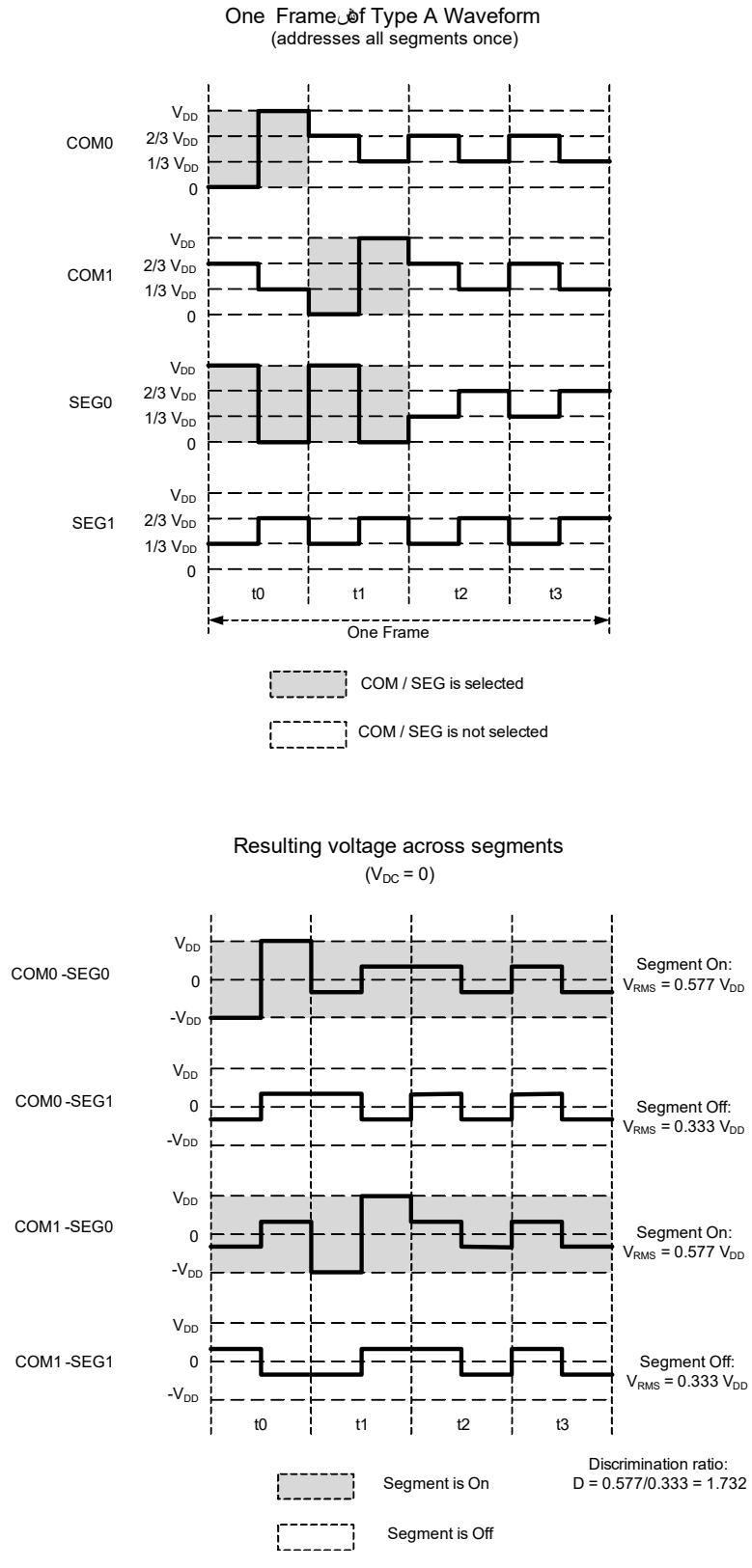
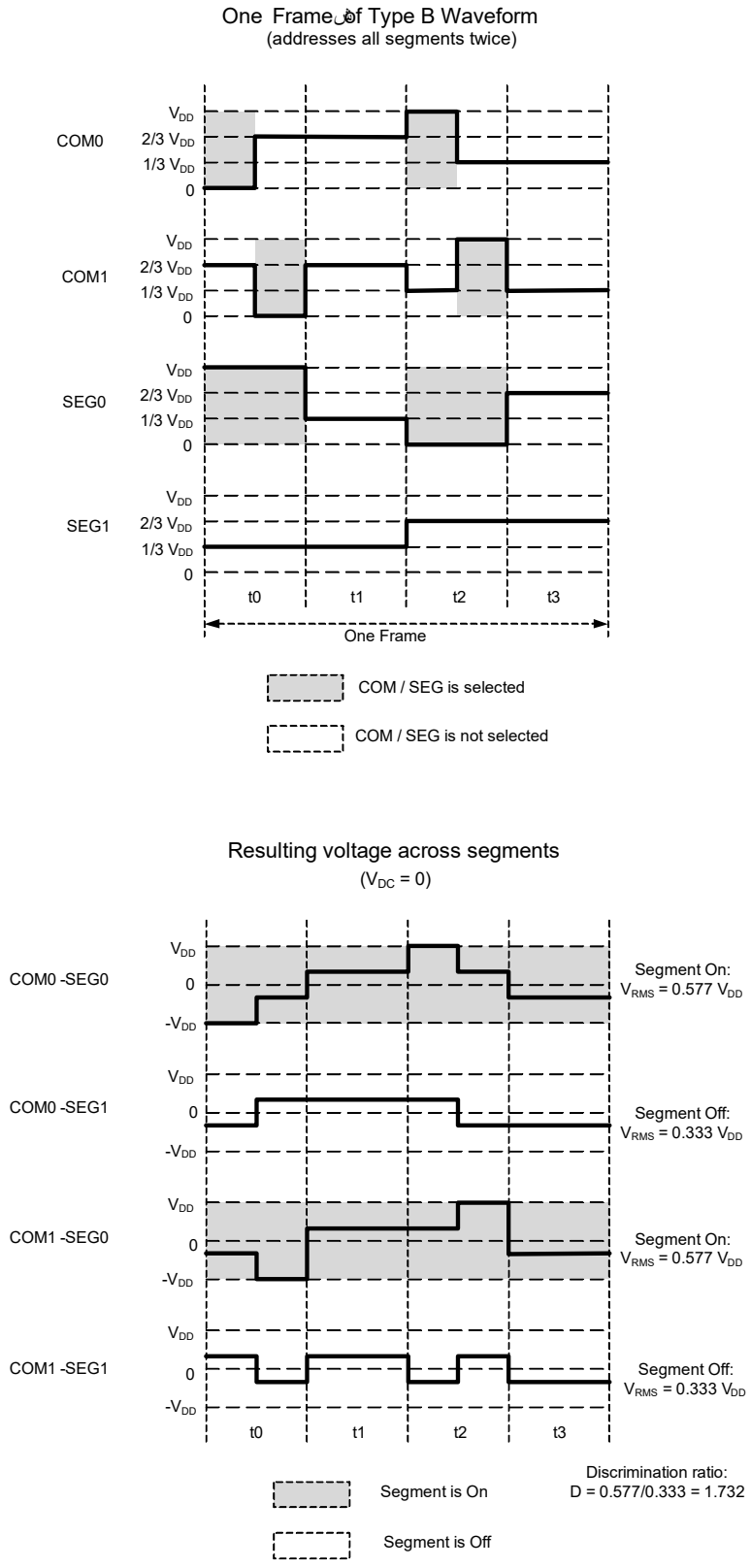


Figure 24-5. PWM1/3 Type-B Waveform Example



The effective RMS voltage for ON and OFF segments can be calculated easily using these equations:

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

Equation 24-1

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

Equation 24-2

Where B is the bias and M is the duty (number of COMs).

For example, if the number of COMs is four, the resulting discrimination ratios (D) for 1/2 and 1/3 biases are 1.528 and 1.732, respectively. 1/3 bias offers better discrimination ratio in two and three COM drives also. Therefore, 1/3 bias offers better contrast than 1/2 bias and is recommended for most applications. 1/4 and 1/5 biases are available only in high-speed operation of the LCD. They offer better discrimination ratio especially when used with high COM designs (more than four COMs).

When the low-speed operation of LCD is used, the PWM signal is derived from the 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 24-2 and Figure 24-3.

24.2.1.2 Digital Correlation

The digital correlation mode, instead of generating bias voltages between the rails, takes advantage of the characteristic of LCDs that the contrast of LCD segments is determined by the RMS voltage across the segments. In this approach, the correlation coefficient between any given pair of COM and SEG signals determines whether the corresponding LCD segment is on or off. Thus, by doubling the base drive frequency of the COM signals in their inactive sub-frame intervals, the phase relationship of the COM and SEG drive signals can be varied to turn segments on and off. This is different from varying the DC levels of the signals as in the PWM drive approach. Figure 24-8 and Figure 24-9 are example waveforms that illustrate the principles of operation.

Figure 24-6. Digital Correlation Type-A Waveform

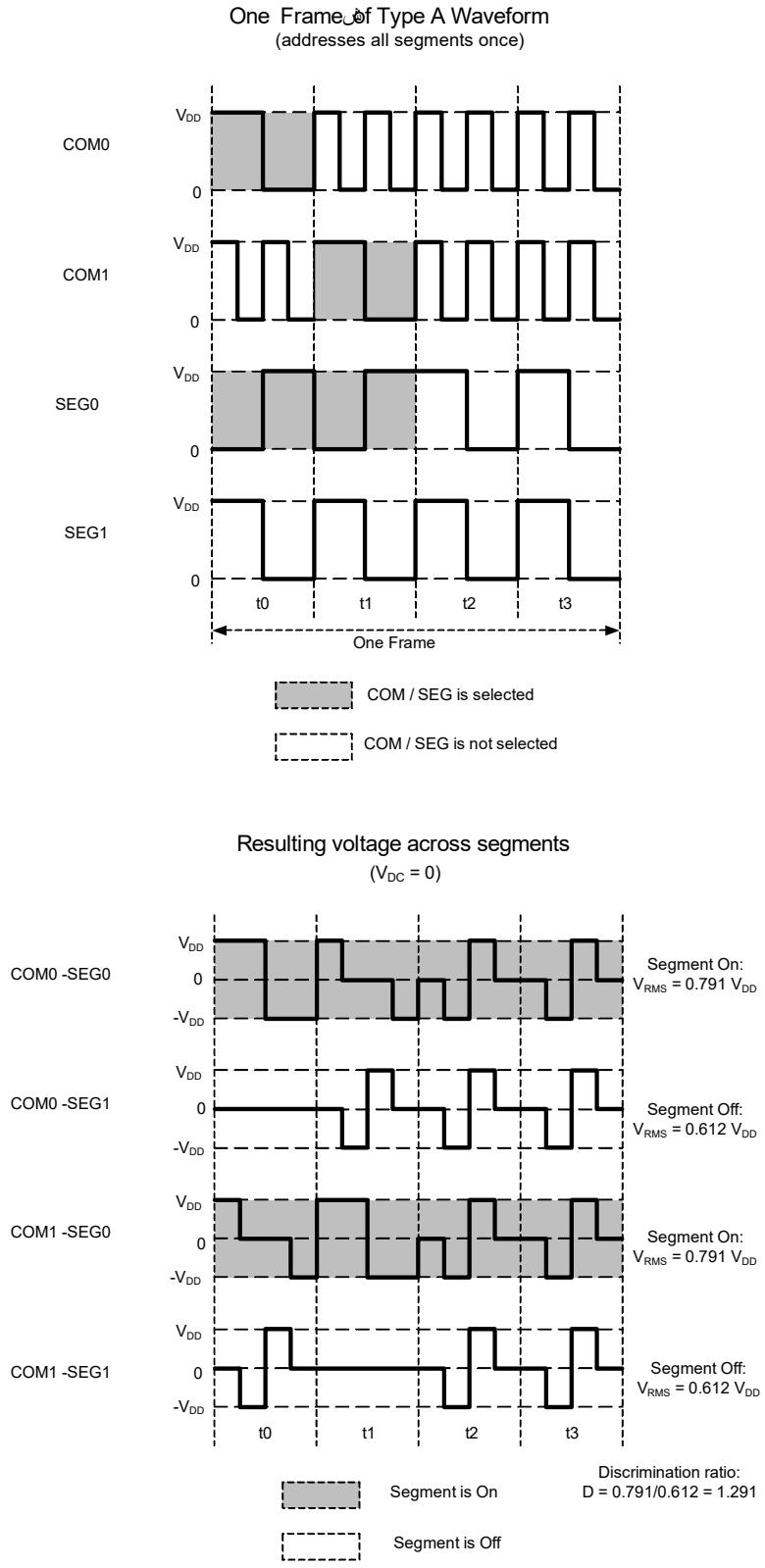
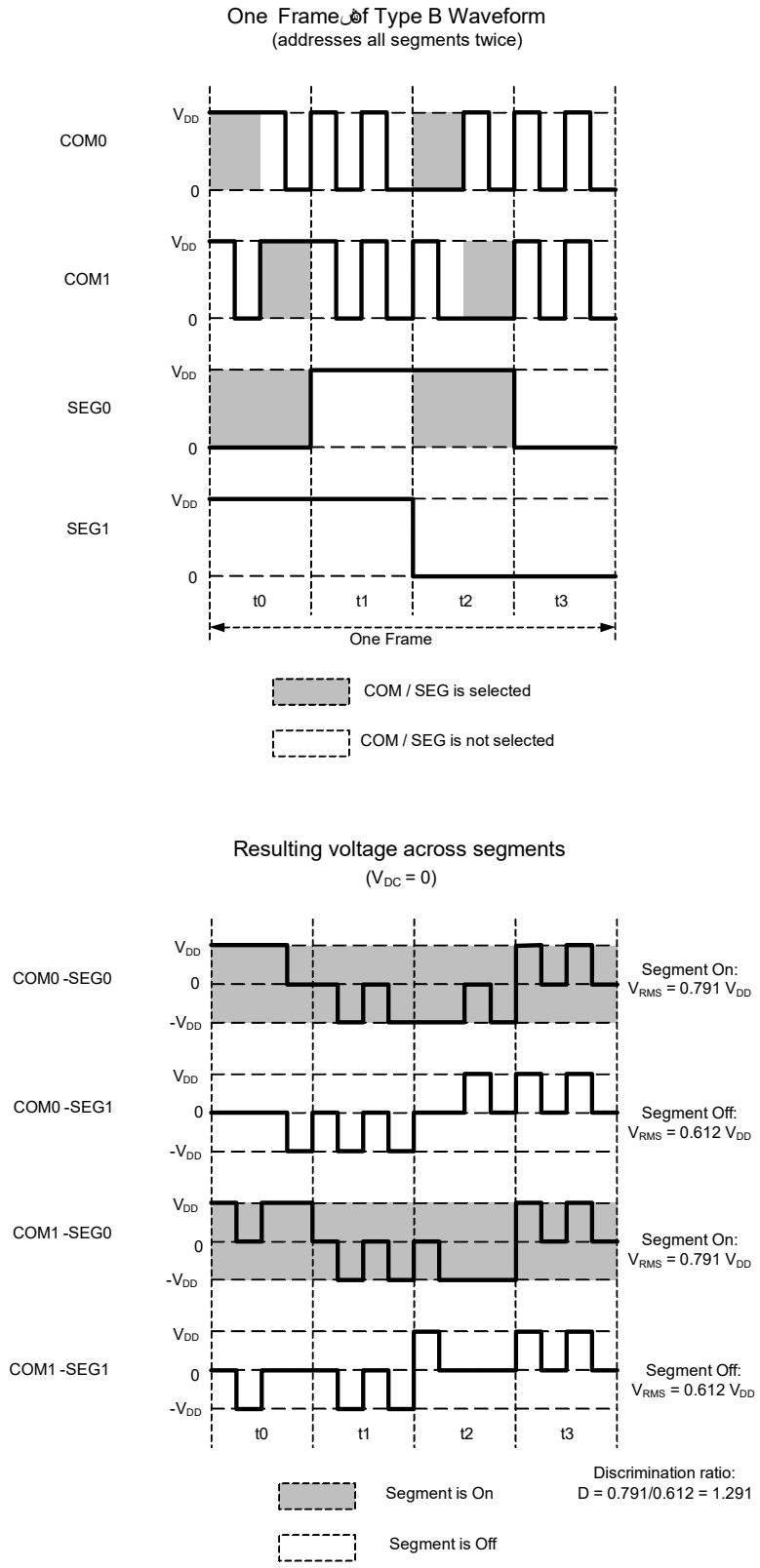


Figure 24-7. Digital Correlation Type-B Waveform



The RMS voltage applied to on and off segments can be calculated as follows:

$$V_{RMS(OFF)} = \sqrt{\frac{(M-1)}{2M}} x(V_{DD})$$

$$V_{RMS(ON)} = \sqrt{\frac{2+(M-1)}{2M}} x(V_{DD})$$

Where B is the bias and M is the duty (number of COMs). This leads to a discrimination ratio (D) of 1.291 for four COMs. Digital correlation mode also has the ability to drive 3-V displays from 1.8-V V_{DD}.

24.2.2 Recommended Usage of Drive Modes

The PWM drive mode has higher discrimination ratios compared to the digital correlation mode, as explained in 24.2.1.1 PWM Drive and 24.2.1.2 Digital Correlation. Therefore, the contrast in digital correlation method is lower than PWM method but digital correlation has lower power consumption because its waveforms toggle at low frequencies.

The digital correlation mode creates reduced, but acceptable contrast on TN displays, but no noticeable difference in contrast or viewing angle on higher contrast STN displays. Because each mode has strengths and weaknesses, recommended usage is as follows.

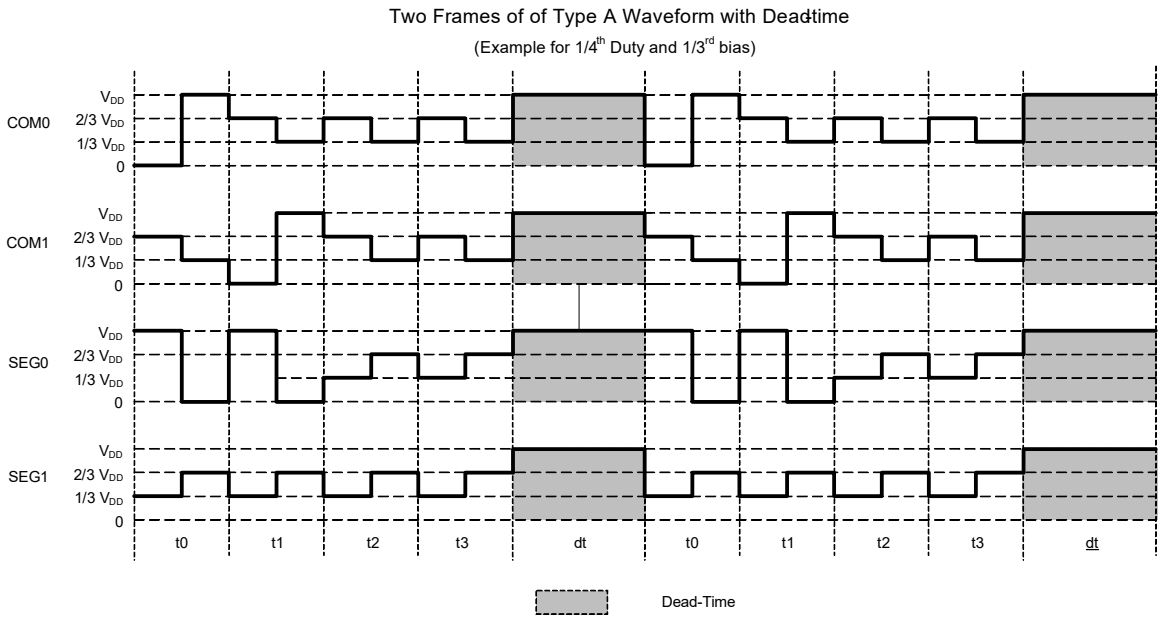
Table 24-1. Recommended Usage of Drive Modes

Display Type	Deep-Sleep Mode	Sleep/Active Mode	Notes
Four COM TN Glass	Digital correlation	PWM 1/3 bias	Firmware must switch between LCD drive modes before going to deep sleep or waking up.
Four COM STN Glass	Digital correlation		No contrast advantage for PWM drive with STN glass.
Eight COM, STN	Not supported	PWM 1/4 bias and 1/5 bias	Supported only in the high-speed LCD mode. The low-speed clock is not fast enough to make the PWM work at high multiplex ratios.

24.2.3 Digital Contrast Control

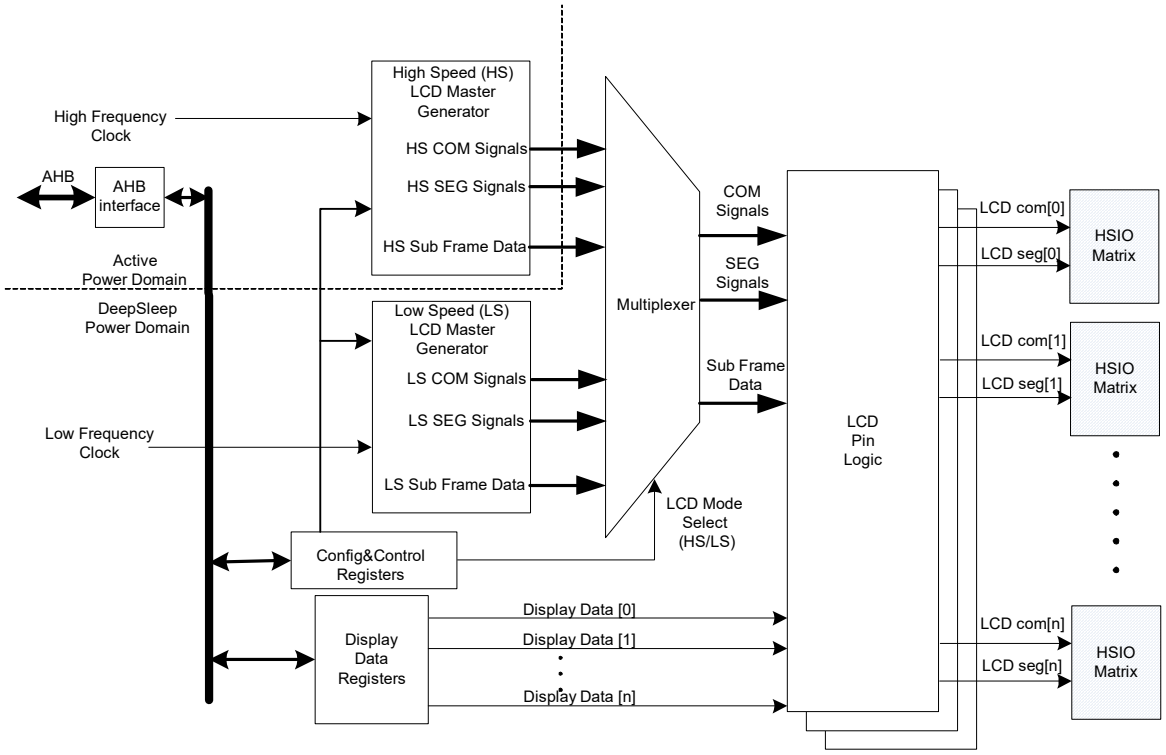
In all drive modes, digital contrast control can be used to change the contrast level of the segments. This method reduces contrast by reducing the driving time of the segments. This is done by inserting a 'Dead-Time' interval after each frame. During dead time, all COM and SEG signals are driven to a logic 1 state. The dead time can be controlled in fine resolution. Figure 24-8 illustrates the dead-time contrast control method for 1/3 bias and 1/4 duty implementation.

Figure 24-8. Dead-Time' Contrast Control



24.3 Block Diagram

Figure 24-9. Block Diagram of LCD Direct Drive System



24.3.1 How it Works

The LCD controller block contains two generators; one with a high-speed clock source HFCLK and the other with a low-speed clock source (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 302.

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 66. These two generators share the same configuration registers. These memory mapped I/O registers are connected to the system bus (AHB) using an AHB interface.

The LCD controller works in three device power modes: active, sleep, and deep-sleep. High-speed operation is supported in active and sleep modes. Low-speed operation is supported in active, sleep, and deep-sleep modes. The LCD controller is unpowered in hibernate and stop modes.

24.3.2 High-Speed and Low-Speed Master Generators

The high-speed and low-speed master generators are similar to each other. The only exception is that the high-speed version has larger frequency dividers to generate the frame and sub-frame periods. This is because the clock of the high-speed block (HFCLK) is derived from the IMO, which is typically at 30 to 100 times the frequency of the ILO (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
- PWM 1/4 bias (not supported in low-speed generator)
- PWM 1/5 bias (not supported in low-speed generator)
- Off/disabled. Typically, one of the two generators will be configured to be Off

OP_MODE and BIAS fields in LCD_CONTROL bits select the drive mode.

- A counter to generate the sub-frame timing. The SUBFR_DIV field in the LCD_DIVIDER register determines the duration of each sub-frame. If the divide value written into this counter is C, the sub-frame period is 4 × (C+1). The low-speed generator has an 8-bit counter. This counter generates a maximum half sub-frame period of 8 ms from the 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.

24.3.3 Multiplexer and LCD Pin Logic

The multiplexer selects the output signals of either high-speed or low-speed master generator blocks and feeds it to the LCD pin logic. This selection is controlled by the configuration and control register. The LCD pin logic uses the sub-frame signal from the multiplexer to choose the display data. This pin logic will be replicated for each LCD pin.

24.3.4 Display Data Registers

Each LCD segment pin is part of an LCD port with its own display data register, LCD_DATAnx. The device has eight such LCD ports. Note that these ports are not real pin ports but the ports/connections available in the LCD hardware for mapping the segments to commons. Each LCD segment configured is considered as a pin in these LCD ports. The LCD_DATAnx registers are 32-bit wide and store the ON/OFF data for all SEG-COM combination enabled in the design. LCD_DATA0x holds SEG-COM data for COM0 to COM3 and LCD_DATA1x holds SEG-COM data for COM4 to COM7. The bits [4i+3:4i] (where 'i' is the pin number) of each LCD_DATA0x register represent the ON/OFF data for Pin[i] in Port[x] and COM[3,2,1,0] combinations, as shown in Table 24-2. The LCD_DATAnx register should be programmed according to the display data of each frame. The display data registers are Memory Mapped I/O (MMIO) and accessed through the AHB slave interface.

Table 24-2. SEG-COM Mapping in LCD_DATA0x Registers (each SEG is a pin of the LCD port)

BITS[31:28] = PIN_7[3:0]				BITS[27:24] = PIN_6[3:0]			
PIN_7-COM3	PIN_7-COM2	PIN_7-COM1	PIN_7-COM0	PIN_6-COM3	PIN_6-COM2	PIN_6-COM1	PIN_6-COM0
BITS[23:20] = PIN_5[3:0]				**BITS[19:16] = PIN_4[3:0]**			
PIN_5-COM3	PIN_5-COM2	PIN_5-COM1	PIN_5-COM0	PIN_4-COM3	PIN_4-COM2	PIN_4-COM1	PIN_4-COM0
BITS[15:12] = PIN_3[3:0]				**BITS[11:8] = PIN_2[3:0]**			
PIN_3-COM3	PIN_3-COM2	PIN_3-COM1	PIN_3-COM0	PIN_2-COM3	PIN_2-COM2	PIN_2-COM1	PIN_2-COM0
BITS[7:3] = PIN_1[3:0]				**BITS[3:0] = PIN_0[3:0]**			
PIN_1-COM3	PIN_1-COM2	PIN_1-COM1	PIN_1-COM0	PIN_0-COM3	PIN_0-COM2	PIN_0-COM1	PIN_0-COM0

24.4 Register List

Table 24-3. LCD Direct Drive Register List

Register Name	Description
LCD_ID	This register includes the information of LCD controller' ID and revision number
LCD_DIVIDER	This register controls the sub-frame and dead-time period
LCD_CONTROL	This register is used to configure high-speed and low-speed generators
LCD_DATA0x	LCD port pin data register for COM0 to COM3; x = port number, eight ports are available
LCD_DATA1x	LCD port pin data register for COM4 to COM7; x = port number, eight ports are available

25. CapSense

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

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

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

26. Temperature Sensor

PSoC® 4 has an on-chip temperature sensor that is used to measure the internal die temperature. The sensor consists of a transistor connected in diode configuration.

26.1 Features

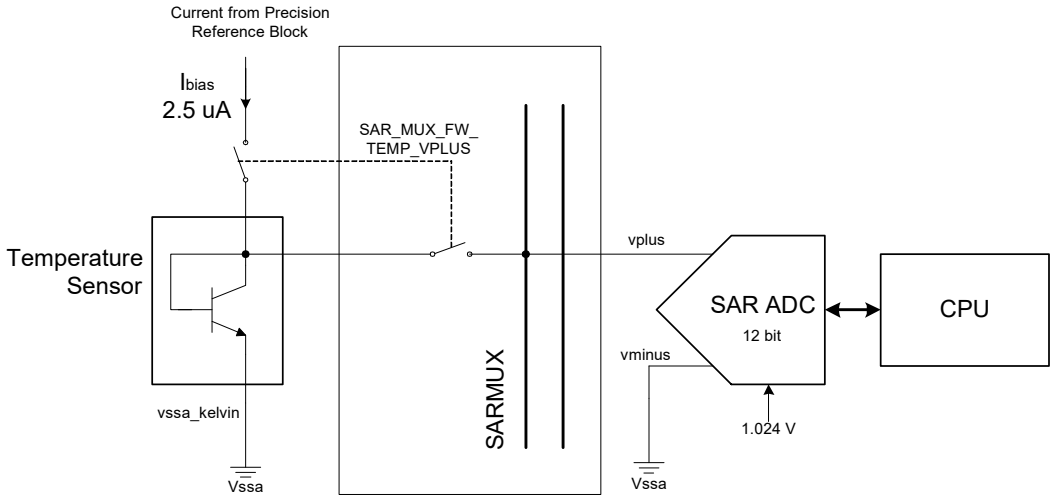
The temperature sensor has the following features:

- ± 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

26.2 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 26-1.

Figure 26-1. 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 26-1

- Temp is the slope compensated temperature in °C represented as Q16.16 fixed point number format.
- 'A' is the 16-bit multiplier constant. The value of A is determined using the PSoC 4 family characterization data of two point slope calculation. It is calculated as given in the following equation.

$$A = (\text{signed int})\left(2^{16}\left(\frac{100°C-(-40°C)}{SAR_{100°C}-SAR_{-40°C}}\right)\right)$$ **Equation 26-2**

Where,

SAR_{100C} = ADC counts at 100°C

SAR_{-40C} = ADC counts at –40°C

Constant 'A' is stored in a register SFLASH_SAR_TEMP_MULTIPLIER.

- 'B' is the 16-bit offset value. The value of B is determined on a per die basis by taking care of all the process variations and the actual bias current (I_{bias}) present in the chip. It is calculated as given in the following equation.

$$B = (\text{unsigned int})\left(2^{6}x100°C-\left(\frac{A\times SAR_{100C}}{2^{10}}\right)\right)$$ **Equation 26-3**

Where,

SAR_{100C} = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH_SAR_TEMP_OFFSET.

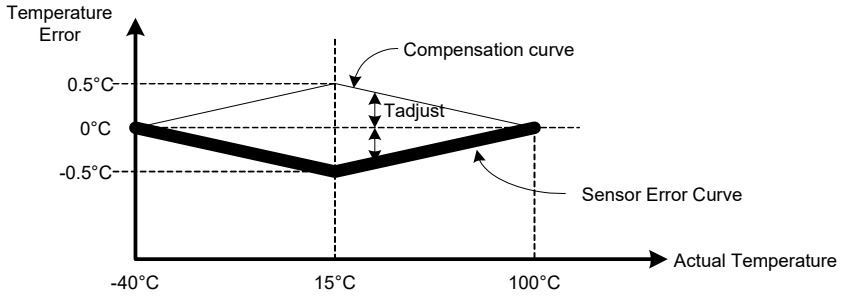
- T_{adjust} is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating $T_{initial} = (A\times SAR_{out}+ 2^{10}\times B)$. If it is greater than 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 26-4**

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 26-5**

Figure 26-2. 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.

26.3 Temperature Sensor Configuration

The temperature sensor output is routed to the positive input of SAR ADC via dedicated switches, which can be controlled by sequencer, firmware, or digital system interconnect (DSI).See the SAR ADC chapter on page 252 for details on how to read the temperature sensor output using the ADC.

26.4 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 26-1).

   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

26.5 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 26-1.
SFLASH_SAR_TEMP_OFFSET	Constant 'B' as defined in Equation 26-1.

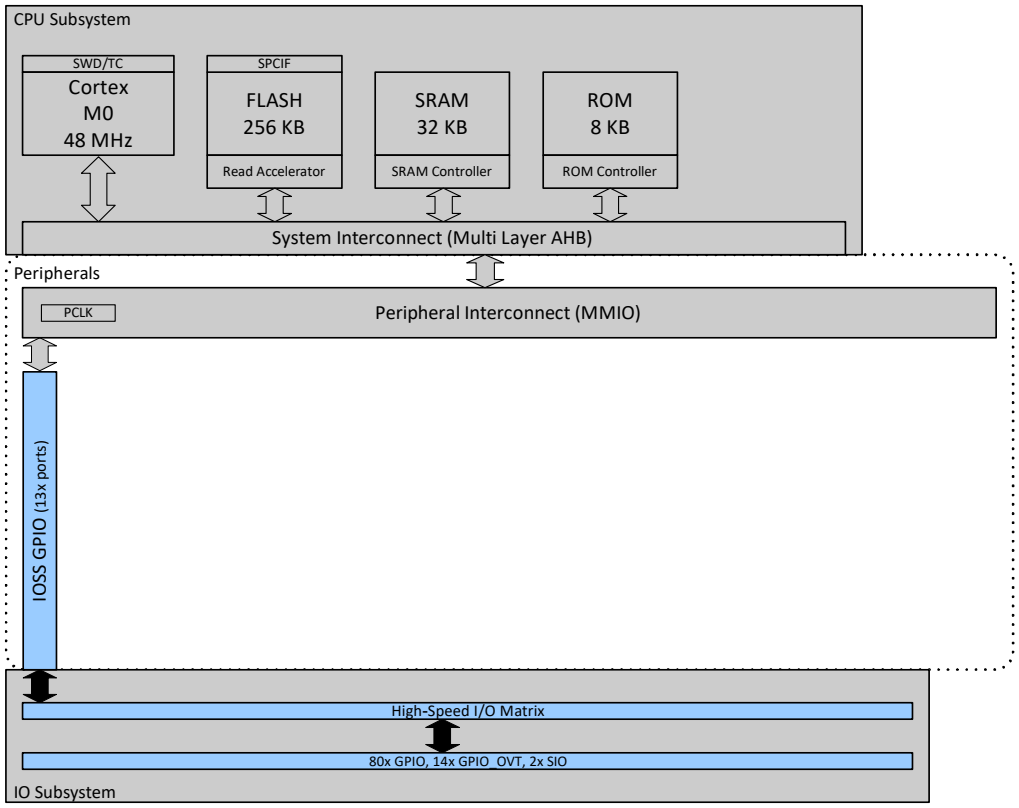
Section F: Program and Debug

This section encompasses the following chapters:

- Program and Debug Interface chapter on page 318
- Nonvolatile Memory Programming chapter on page 325

Top Level Architecture

Program and Debug Block Diagram



27. Program and Debug Interface

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

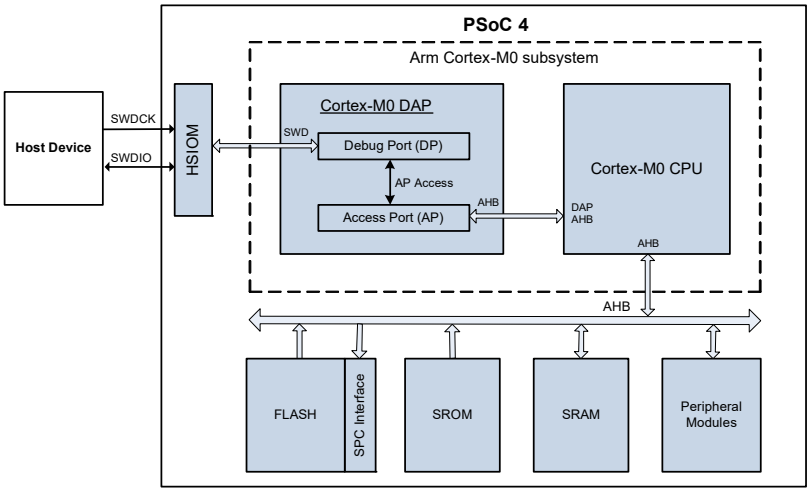
27.1 Features

- 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

27.2 Functional Description

Figure 27-1 shows the block diagram of the program and debug interface in PSoC 4. The Cortex-M0 debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the “host”, communicates with the DAP of the PSoC 4 “target” using the two pins of the SWD interface - the bidirectional data pin (SWDIO) and the host-driven clock pin (SWDCK). The SWD physical port pins (SWDIO and SWDCK) communicate with the DAP through the high-speed I/O matrix (HSIOM). See the I/O System chapter on page 55 for details on HSIOM.

Figure 27-1. Program and Debug Interface



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

27.3 Serial Wire Debug (SWD) Interface

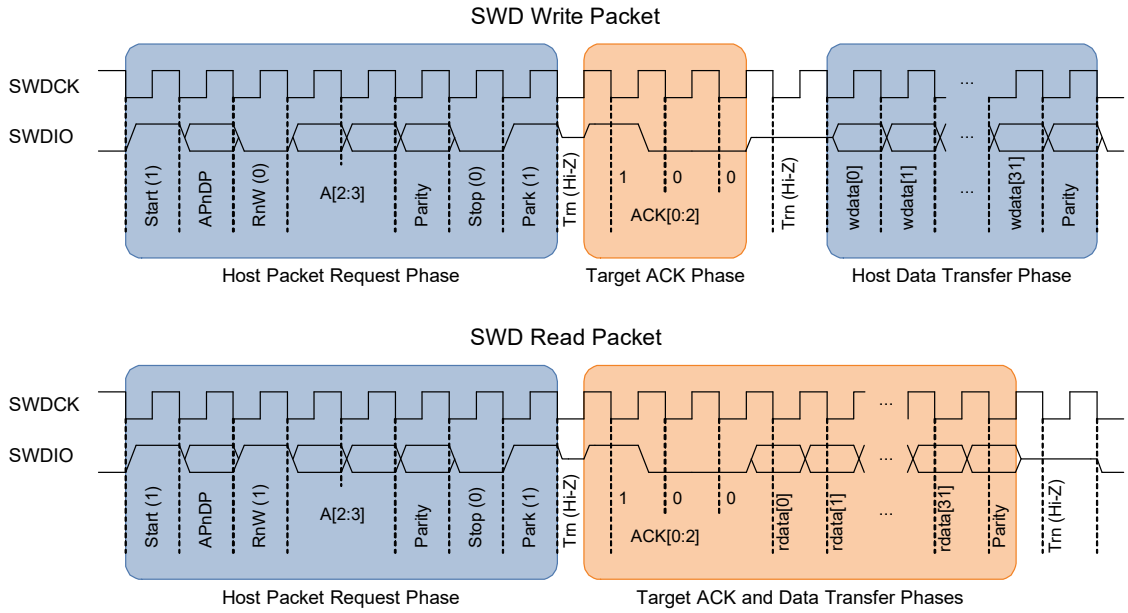
PSoC 4's Cortex-M0 supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete data transfer (one SWD packet) requires 46 clocks and consists of three phases:

- **Host Packet Request Phase** – The host issues a request to the PSoC 4 target.
- **Target Acknowledge Response Phase** – The PSoC 4 target sends an acknowledgement to the host.
- **Data Transfer Phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

Figure 27-2 shows the timing diagrams of read and write SWD packets.

Figure 27-2. 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 27-3 and Table 27-4 for definitions.
   Note Address bits are transmitted with the LSB first.
   e. The parity bit contains the parity of APnDP, RnW, and ADDR bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.
   If the parity bit is not correct, the header is ignored by PSoC 4; there is no ACK response (ACK = 3b111). The programming operation should be aborted and retried again by following a device reset.
   f. The stop bit is always logic 0.
   g. The park bit is always logic 1.
2. Target Acknowledge Response Phase: SWDIO driven by the target
   a. The ACK[2:0] bits represent the target to host response, indicating failure or success, among other results. See Table 27-1 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.

27.3.1 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 27-1 and Figure 27-2 illustrate the timing of SWDIO bit writes and reads.

Table 27-1. 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		

27.3.2 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 27-2 shows the ACK bit-field decoding details.

Table 27-2. SWD Transfer ACK Response Decoding

Response	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	3b111

Details on WAIT and FAULT response behaviors are as follows:

- For a WAIT response, if the transaction is a read, the host should ignore the data read in the data phase. The target does not drive the line and the host must not check the parity bit as well.
- For a WAIT response, if the transaction is a write, the data phase is ignored by the PSoC 4. But, the host must still send the data to be written to complete the packet. The parity bit corresponding to the data should also be sent by the host.
- For a WAIT response, it means that the PSoC 4 is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received. If it fails, then the programming operation should be aborted and retried again.
- For a FAULT response, the programming operation should be aborted and retried again by doing a device reset.

27.3.3 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 27-2. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This action ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor the PSoC 4 should drive the SWDIO line during the Trn period.

27.4 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.

27.4.1 Debug Port (DP) Registers

Table 27-3 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 27-3. Main Debug Port (DP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex-M0 CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In PSoC 4, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

27.4.2 Access Port (AP) Registers

Table 27-4 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 27-4. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the PSoC 4 Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register

27.5 Programming the PSoC 4 Device

PSoC 4 is programmed using the following sequence. Refer to the CY8C4xxx, CYBLxxxx Programming Specifications for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in PSoC 4.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

27.5.1 SWD Port Acquisition

27.5.1.1 SWD Port Acquire Sequence

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

Code 1. SWD Port Acquire Pseudo Code

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

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

}while ((ack != OK) && time_elapsed < 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.

27.5.2 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 CY8C4xxx, CYBLxxxx Programming Specifications document.

27.5.3 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 325. The exact sequence of calling the programming routines is given in the CY8C4xxx, CYBLxxxx Programming Specifications document.

27.6 PSoC 4 SWD Debug Interface

Cortex-M0 DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex-M0 Debug.

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

27.6.1 Debug Control and Configuration Registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the Armv6-M Architecture Reference Manual for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0_DFSR) – This register describes the reason a debug event has occurred and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0_DCRSR) – This register is used to select the general-purpose register in the Cortex-M0 CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0_DCRSR register.
- Debug Exception and Monitor Control Register (CM0_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

27.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0 DAP in PSoC 4 supports up to four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction in the Cortex-M0. The BPU has two types of registers.

- The breakpoint control register (CM0_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in the PSoC 4).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0_BP_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

27.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. Trace is not supported by the Cortex-M0 in PSoC 4. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows.

- The watchpoint compare (CM0_DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint mask (CM0_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0_DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event has occurred.
- The watchpoint comparator PC sample register (CM0_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, non-invasive profiling of the program counter register.

27.6.4 Debugging the PSoC 4 Device

The host debugs the target PSoC 4 by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0 DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target PSoC 4 is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the cor-

rect CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the I/O System chapter on page 55 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 104. 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.

27.7 Registers

Table 27-5. 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

28. Nonvolatile Memory Programming

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

28.1 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

28.2 Functional Description

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

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

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

Note It can take as much as 20 milliseconds to write to flash. During this time, the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources (see the Reset System chapter on page 101) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

Note: The PSoC 4 devices have supervisory flash rows in addition to the users' rows. A part of this supervisory flash is available for the user to store application specific information. Refer to the CY8C4xxx, CYBLxxxx Programming Specifications for more information about the organization of the supervisory flash and the method to program the supervisory flash area.

28.3 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.

28.4 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.

28.4.1 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 28-1 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.

28.5 System Calls

Table 28-1 lists all the system calls supported in PSoC 4 along with the function description and availability in device protection modes. See the Device Security chapter on page 104 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 28-1. List of System Calls

System Call	Description	DAP Access			CPU Access
		Open	Protected	Kill	
Silicon ID	Returns the device Silicon ID, Family ID, and Revision ID	?	?	–	?
Load Flash Bytes	Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 256 bytes	?	–	–	?
Write Row	Erases and then programs a row of flash with data in the page latch buffer	?	–	–	?
Program Row	Programs a row of flash with data in the page latch buffer	?	–	–	?
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	?	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	?	?	–	?
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	?	?	–	
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	?
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	?
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	?

28.5.1 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 CY8C4xxx, CYBLxxxx 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 0x0A0 for PSoC 4200L
Bits [15:12]	Chip Protection	See the Device Security chapter on page 104
Bits [31:16]	0xXXXX	Not used

28.5.2 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.

28.5.3 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 256 bytes. Data is loaded into the page latch buffer starting at the location specified by the “Byte Addr” input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS_SYSARG register. Note that the starting parameter address should be a word-aligned address.

Parameters

Address	Value to be Written	Description
SRAM Address - 32’hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD7	Key2
Bits [23:16]	Byte Addr	Start address of page latch buffer to write data 0x00 – Byte 0 of latch buffer 0xFF – Byte 255 of latch buffer
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1 (Refer to the Cortex-M0 CPU chapter on page 26 for the number of flash macros in the device)

Address	Value to be Written	Description
SRAM Address- 32'hYY + 0x04		
Bits [7:0]	Load Size	Number of bytes to be written to the page latch buffer. 0x00 – 1 byte 0xFF – 256 bytes
Bits [15:8]	0xXX	Don't care parameter
Bits [23:16]	0xXX	Don't care parameter
Bits [31:24]	0xXX	Don't care parameter
SRAM Address- From (32'hYY + 0x08) to (32'hYY + 0x08 + Load Size)		
Byte 0	Data Byte [0]	First data byte to be loaded
.	.	.
.	.	.
Byte (Load size –1)	Data Byte [Load size –1]	Last data byte to be loaded
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0004	Load Flash Bytes opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

28.5.4 Write Row

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

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

Refer to the CLK_IMO_CONFIG register in the PSoC 4200L Registers TRM for more information.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		

Address	Value to be Written	Description
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0005	Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

28.5.5 Program Row

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

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

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD9	Key2
Bits [31:16]	Row ID	Row number to program 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0006	Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

28.5.6 Erase All

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

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

Parameters

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

Return

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

28.5.7 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

28.5.8 Write Protection

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

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

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

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE0	Key2
Bits [23:16]	Device Protection Byte	Parameter applicable only for Flash Macro 0 0x01 – OPEN mode 0x02 – PROTECTED mode 0x04 – KILL mode
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0 0x01 – Flash Macro 1
CPUSS_SYSREQ register		
Bits [15:0]	0x000D	Write Protection opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	0x000000	

28.5.9 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 326.

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

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

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

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

Parameters

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

Return

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

28.5.10 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 326. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

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

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

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDB	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

28.5.11 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 326.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDC	Key2
Bits [31:16]	0xXXXX	Don't care. Not used by SROM
CPUSS_SYSARG register		

Address	Value to be Written	Description
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0009	Resume Non-Blocking opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

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

28.6 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 28-2. 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.

28.7 Non-Blocking System Call Pseudo Code

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

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

#define ROW_SIZE_          ()
#define ROW_SIZE        (ROW_SIZE_)

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

__flash int main(void)
{
  DoUserStuff();

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

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

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

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

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

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

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

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

```
  //Write load size param (128 bytes) to SRAM
  REG( 0x20000004 ) = 0x0000007F;


  for(i = 0; i < ROW_SIZE/4; i += 1)
  {
     REG( 0x20000008 + i*4 ) = 0xDADADADA;
  }

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

  /*Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
  * register and assert the sysreq bit
  */
  CPUSS_SYSREQ_REG = 0x80000004;

  /*Perform Non-Blocking Write Row on Row 200 as an example.
  * Write key1, key2, row id to SRAM row id = 0xC8 -> which is row 200
  */
  REG( 0x20000000 ) = 0x00C8DAB6;

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

  /*Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
  * register and assert the sysreq bit
  */
  CPUSS_SYSREQ_REG = 0x80000007;

  /*Execute user code until iStatusInt equals 3 to signify
  * 3 SPC interrupts have happened. This should be 1 in case
  * of non-blocking program System Call
  */
  while( iStatusInt != 0x03 )
  {
     DoOtherUserStuff();
  }

  /* Get the success or failure status of System Call*/
  syscall_status = CPUSS_SYSARG_REG;
}
```

In the code, the CM0 exception table is configured to be in SRAM by writing 0x01 to the CPUSS_CONFIG register. The SRAM exception table should have the vector address of the SPC interrupt as the address of the *SpcIntHandler()* function, which is also defined to be in SRAM. See the Interrupts chapter on page 45 for details on configuring the CM0 exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PSoC device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.

AND See *Boolean Algebra*.

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

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

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

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

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

B

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

bandwidth

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

bias

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

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

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

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

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

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

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

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

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

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

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

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

byte A digital storage unit consisting of 8 bits.

C

C A high-level programming language.

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

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

D

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

E

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

F

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

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

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

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

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

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

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

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

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

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

G

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

gate

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

ground

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

H

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

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

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

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

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

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

I

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

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

impedance	1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit. 2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.
input	A point that accepts data, in a device, process, or channel.
input/output (I/O)	A device that introduces data into or extracts data from a system.
instruction	An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.
instruction mnemonics	A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.
integrated circuit (IC)	A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.
interface	The means by which two systems or devices are connected and interact with each other.
interrupt	A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.
interrupt service routine (ISR)	A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter	1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams. 2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

L

latency	The time or delay that it takes for a signal to pass through a given circuit or network.
least significant bit (LSb)	The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.
least significant byte (LSB)	The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

Linear Feedback Shift Register (LFSR)	A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.
load	The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).
logic function	A mathematical function that performs a digital operation on digital data and returns a digital value.
lookup table (LUT)	A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.
low time	The amount of time the signal has a value of '0' in one period, for a periodic digital signal.
low-voltage detect (LVD)	A circuit that senses V_{DDD} and provides an interrupt to the system when V_{DDD} falls below a selected threshold.

M

M8CP	An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PSoC device by interfacing to the flash, SRAM, and register space.
macro	A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.
mask	1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference. 2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.
master device	A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.
microcontroller	An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.
mnemonic	A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode	A distinct method of operation for software or hardware. For example, the Digital PSoC block may be in either counter mode or timer mode.
modulation	A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.
Modulator	A device that imposes a signal on a carrier.
MOS	An acronym for metal-oxide semiconductor.
most significant bit (MSb)	The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.
most significant byte (MSB)	The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.
multiplexer (mux)	1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output. 2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND	See *Boolean Algebra*.
negative edge	A transition from a logic 1 to a logic 0. Also known as a falling edge.
net	The routing between devices.
nibble	A group of four bits, which is one-half of a byte.
noise	1. A disturbance that affects a signal and that may distort the information carried by the signal. 2. The random variations of one or more characteristics of any entity such as voltage, current, or data.
NOR	See *Boolean Algebra*.
NOT	See *Boolean Algebra*.

O

OR	See *Boolean Algebra*.
oscillator	A circuit that may be crystal controlled and is used to generate a clock frequency.
output	The electrical signal or signals which are produced by an analog or digital block.

P

parallel	The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.
parameter	Characteristics for a given block that have either been characterized or may be defined by the designer.
parameter block	A location in memory where parameters for the SSC instruction are placed prior to execution.
parity	A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).
path	1. The logical sequence of instructions executed by a computer. 2. The flow of an electrical signal through a circuit.
pending interrupts	An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.
phase	The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).
pin	A terminal on a hardware component. Also called lead.
pinouts	The pin number assignment: the relation between the logical inputs and outputs of the PSoC device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names.
port	A group of pins, usually eight.
positive edge	A transition from a logic 0 to a logic 1. Also known as a rising edge.
posted interrupts	An interrupt that is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.
Power On Reset (POR)	A circuit that forces the PSoC device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*.
program counter	The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.
protocol	A set of rules. Particularly the rules that govern networked communications.
PSoC®	Cypress's Programmable System-on-Chip (PSoC®) devices.
PSoC blocks	*See analog blocks* and *digital blocks*.
PSoC Creator™	The software for Cypress's next generation Programmable System-on-Chip technology.

pulse	A rapid change in some characteristic of a signal (for example, phase or frequency), from a baseline value to a higher or lower value, followed by a rapid return to the baseline value.
pulse width modulator (PWM)	An output in the form of duty cycle which varies as a function of the applied measure.

R

RAM	An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.
register	A storage device with a specific capacity, such as a bit or byte.
reset	A means of bringing a system back to a know state. See *hardware reset* and *software reset*.
resistance	The resistance to the flow of electric current measured in ohms for a conductor.
revision ID	A unique identifier of the PSoC device.
ripple divider	An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to 2^n - 1.
rising edge	*See positive edge*.
ROM	An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.
routine	A block of code, called by another block of code, that may have some general or frequent use.
routing	Physically connecting objects in a design according to design rules set in the reference library.
runt pulses	In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling	The process of converting an analog signal into a series of digital values or reversed.
schematic	A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.
seed value	An initial value loaded into a linear feedback shift register or random number generator.
serial	1. Pertaining to a process in which all events occur one after the other. 2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.

set	To force a bit/register to a value of logic 1.
settling time	The time it takes for an output signal or value to stabilize after the input has changed from one value to another.
shift	The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.
shift register	A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.
sign bit	The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.
signal	A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.
silicon ID	A unique identifier of the PSoC silicon.
skew	The difference in arrival time of bits transmitted at the same time, in parallel transmission.
slave device	A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.
software	A set of computer programs, procedures, and associated documentation about the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed.
software reset	A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not PSoC blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at flash address 0x0000.
SRAM	An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.
SROM	An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash.
stack	A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.
stack pointer	A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.
state machine	The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.

sticky A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The PSoC SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous
1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the PSoC device and reading this manual.

user modules Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital PSoC Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

V_{DDD} A name for a power net meaning "voltage drain." The most positive power supply signal. Usually 5 or 3.3 volts.

volatile Not guaranteed to stay the same value or level when not in scope.

V_{SS} A name for a power net meaning "voltage source." The most negative power supply signal.

W

watchdog timer A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time.

waveform The representation of a signal as a plot of amplitude versus time.

X

XOR See *Boolean Algebra*.