



**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 4100S Max**

# PSoC 4 Architecture Technical Reference Manual (TRM)

Document No. 002-32818 Rev. \*\*

July 16, 2021

Cypress Semiconductor  
An Infineon Technologies Company  
198 Champion Court  
San Jose, CA 95134-1709  
[www.cypress.com](http://www.cypress.com)  
[www.infineon.com](http://www.infineon.com)

## Copyrights

© Cypress Semiconductor Corporation, 2021. This document is the property of Cypress Semiconductor Corporation, an Infineon Technologies company, and its affiliates ("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, including its affiliates, and its directors, officers, employees, agents, 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, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, Traveo, WICED, and ModusToolbox are trademarks or registered trademarks of Cypress or a subsidiary of Cypress in the United States or in other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Their names and brands may be claimed as property of their respective owners.

# Content Overview



|   |            |
|---|------------|
| <b>Section A: Overview</b>                          | <b>17</b>  |
| 1. Introduction .....                               | 18         |
| 2. Getting Started .....                            | 24         |
| 3. Document Construction .....                      | 26         |
| <b>Section B: CPU System</b>                        | <b>30</b>  |
| 4. Cortex-M0+ CPU .....                             | 31         |
| 5. DMA Controller Modes .....                       | 37         |
| 6. Interrupts .....                                 | 52         |
| 7. Device Security .....                            | 63         |
| <b>Section C: System Resources Subsystem (SRSS)</b> | <b>65</b>  |
| 8. I/O System .....                                 | 66         |
| 9. Clocking System .....                            | 88         |
| 10. Power Supply and Monitoring .....               | 99         |
| 11. Chip Operational Modes .....                    | 103        |
| 12. Power Modes .....                               | 104        |
| 13. Watchdog Timer .....                            | 108        |
| 14. Trigger Multiplexer Block .....                 | 113        |
| 15. Reset System .....                              | 117        |
| <b>Section D: Digital System</b>                    | <b>119</b> |
| 16. Serial Communications Block (SCB) .....         | 120        |
| 17. CAN FD Controller .....                         | 179        |
| 18. Timer, Counter, and PWM (TCPWM) .....           | 249        |
| 19. LCD Direct Drive .....                          | 285        |
| 20. Inter-IC Sound Bus .....                        | 294        |
| 21. CRYPTO .....                                    | 301        |
| <b>Section E: Analog System</b>                     | <b>314</b> |
| 22. SAR ADC .....                                   | 315        |
| 23. Low-Power Comparator .....                      | 339        |
| 24. Continuous Time Block mini (CTBm) .....         | 344        |
| 25. CapSense .....                                  | 353        |
| 26. Temperature Sensor .....                        | 354        |
| 27. Analog Routing .....                            | 357        |

**Section F: Program and Debug 365**

|     |                                      |     |
|-----|--------------------------------------|-----|
| 28. | Program and Debug Interface .....    | 366 |
| 29. | Nonvolatile Memory Programming ..... | 374 |

# Contents



|  |           |
|--|-----------|
| <b>Section A: Overview</b>                     | <b>17</b> |
| <b>1. Introduction</b>                         | <b>18</b> |
| 1.1 Top Level Architecture .....               | 18        |
| 1.2 Features.....                              | 19        |
| 1.3 CPU System .....                           | 19        |
| 1.3.1 Processor.....                           | 19        |
| 1.3.2 Interrupt Controller .....               | 19        |
| 1.3.3 Direct Memory Access .....               | 19        |
| 1.3.4 Cryptographic Accelerator (CRYPTO) ..... | 19        |
| 1.3.5 Memory .....                             | 20        |
| 1.3.5.1 Flash.....                             | 20        |
| 1.3.5.2 SRAM .....                             | 20        |
| 1.4 System-Wide Resources .....                | 20        |
| 1.4.1 Clocking System .....                    | 20        |
| 1.4.2 Power System.....                        | 20        |
| 1.4.3 GPIO .....                               | 20        |
| 1.4.4 Watchdog Timers .....                    | 21        |
| 1.5 Fixed-Function Digital .....               | 21        |
| 1.5.1 Timer/Counter/PWM Block.....             | 21        |
| 1.5.2 Serial Communication Blocks .....        | 21        |
| 1.5.3 Controller Area Network (CAN).....       | 21        |
| 1.5.4 LCD Segment Drive .....                  | 21        |
| 1.5.5 I2S Master.....                          | 21        |
| 1.6 Analog System.....                         | 22        |
| 1.6.1 SAR ADC .....                            | 22        |
| 1.6.2 Continuous Time Block mini (CTBm).....   | 22        |
| 1.6.3 Low-Power Comparators .....              | 22        |
| 1.7 Special Function Peripherals .....         | 22        |
| 1.7.1 CapSense .....                           | 22        |
| 1.8 Program and Debug .....                    | 22        |
| 1.9 Device Feature Summary .....               | 23        |
| <b>2. Getting Started</b>                      | <b>24</b> |
| 2.1 Support .....                              | 24        |
| 2.2 Development Ecosystem .....                | 24        |
| 2.2.1 PSoC 4 MCU Resources .....               | 24        |
| 2.2.2 ModusToolbox™ Software .....             | 25        |
| <b>3. Document Construction</b>                | <b>26</b> |
| 3.1 Major Sections .....                       | 26        |
| 3.2 Documentation Conventions.....             | 26        |
| 3.2.1 Register Conventions.....                | 26        |
| 3.2.2 Numeric Naming .....                     | 26        |

|                              |  |           |
|------------------------------|--|-----------|
| 3.2.3                        | Units of Measure .....                               | 27        |
| 3.2.4                        | Acronyms .....                                       | 28        |
| <b>Section B: CPU System</b> |  | <b>30</b> |
| <b>4.</b>                    | <b>Cortex-M0+ CPU</b>                                | <b>31</b> |
| 4.1                          | Features .....                                       | 31        |
| 4.2                          | Block Diagram .....                                  | 32        |
| 4.3                          | How It Works .....                                   | 32        |
| 4.4                          | Address Map .....                                    | 32        |
| 4.5                          | Registers .....                                      | 33        |
| 4.6                          | Operating Modes .....                                | 34        |
| 4.7                          | Instruction Set .....                                | 35        |
| 4.7.1                        | Address Alignment .....                              | 36        |
| 4.7.2                        | Memory Endianness .....                              | 36        |
| 4.8                          | Systick Timer .....                                  | 36        |
| 4.9                          | Debug .....  | 36        |
| <b>5.</b>                    | <b>DMA Controller Modes</b>                          | <b>37</b> |
| 5.1                          | Block Diagram Description .....                      | 37        |
| 5.1.1                        | Trigger Sources and Multiplexing .....               | 38        |
| 5.1.1.1                      | Trigger Multiplexer .....                            | 38        |
| 5.1.1.2                      | Creating Software Triggers .....                     | 41        |
| 5.1.2                        | Pending Triggers .....                               | 41        |
| 5.1.3                        | Output Triggers .....                                | 41        |
| 5.1.4                        | Channel Prioritization .....                         | 41        |
| 5.1.5                        | Data Transfer Engine .....                           | 41        |
| 5.2                          | Descriptors .....                                    | 42        |
| 5.2.1                        | Address Configuration .....                          | 42        |
| 5.2.2                        | Transfer Size .....                                  | 44        |
| 5.2.3                        | Descriptor Chaining .....                            | 44        |
| 5.2.4                        | Transfer Mode .....                                  | 45        |
| 5.2.4.1                      | Single Data Element Per Trigger (OPCODE 0) .....     | 45        |
| 5.2.4.2                      | Entire Descriptor Per Trigger (OPCODE 1) .....       | 46        |
| 5.2.4.3                      | Entire Descriptor Chain Per Trigger (OPCODE 2) ..... | 47        |
| 5.3                          | Operation and Timing .....                           | 48        |
| 5.4                          | Arbitration .....                                    | 50        |
| 5.5                          | Register Lists .....                                 | 51        |
| <b>6.</b>                    | <b>Interrupts</b>                                    | <b>52</b> |
| 6.1                          | Features .....                                       | 52        |
| 6.2                          | How It Works .....                                   | 52        |
| 6.3                          | Interrupts and Exceptions - Operation .....          | 53        |
| 6.3.1                        | Interrupt/Exception Handling .....                   | 53        |
| 6.3.2                        | Level and Pulse Interrupts .....                     | 54        |
| 6.3.3                        | Exception Vector Table .....                         | 55        |
| 6.4                          | Exception Sources .....                              | 56        |
| 6.4.1                        | Reset Exception .....                                | 56        |
| 6.4.2                        | Non-Maskable Interrupt (NMI) Exception .....         | 56        |
| 6.4.3                        | HardFault Exception .....                            | 56        |
| 6.4.4                        | Supervisor Call (SVCALL) Exception .....             | 56        |
| 6.4.5                        | PendSV Exception .....                               | 57        |
| 6.4.6                        | SysTick Exception .....                              | 57        |
| 6.5                          | Interrupt Sources .....                              | 57        |

|   |   |           |
|---|---|-----------|
| 6.6   | Exception Priority .....                                      | 58        |
| 6.7   | Enabling and Disabling Interrupts .....                       | 59        |
| 6.8   | Exception States .....  | 59        |
| 6.8.1   | Pending Exceptions .....                                      | 60        |
| 6.9   | Stack Usage for Exceptions .....                              | 60        |
| 6.10  | Interrupts and Low-Power Modes .....                          | 61        |
| 6.11  | Exceptions – Initialization and Configuration .....           | 61        |
| 6.12  | Registers .....   | 61        |
| 6.13  | Associated Documents .....                                    | 62        |
| <b>7.</b>   | <b>Device Security</b> .....                                  | <b>63</b> |
| 7.1   | Features .....  | 63        |
| 7.2   | How It Works .....  | 63        |
| 7.2.1   | Device Security .....   | 63        |
| 7.2.2   | Flash Security .....  | 64        |
| <b>Section C: System Resources Subsystem (SRSS)</b> ..... |   | <b>65</b> |
| <b>8.</b>   | <b>I/O System</b> .....                                       | <b>66</b> |
| 8.1   | Features .....  | 66        |
| 8.2   | GPIO Interface Overview .....                                 | 67        |
| 8.3   | I/O Cell Architecture .....                                   | 68        |
| 8.3.1   | Digital Input Buffer .....                                    | 69        |
| 8.3.2   | Digital Output Driver .....                                   | 69        |
| 8.3.2.1   | Drive Modes .....   | 69        |
| 8.3.2.2   | Slew Rate Control .....                                       | 71        |
| 8.4   | High-Speed I/O Matrix .....                                   | 72        |
| 8.5   | Smart I/O .....   | 73        |
| 8.5.1   | Overview .....  | 73        |
| 8.5.2   | Block Components .....  | 74        |
| 8.5.2.1   | Clock and Reset .....   | 74        |
| 8.5.2.2   | Synchronizer .....  | 75        |
| 8.5.2.3   | Lookup Table (LUT3) .....                                     | 76        |
| 8.5.2.4   | Data Unit .....   | 79        |
| 8.5.3   | Routing .....   | 82        |
| 8.5.4   | Operation .....   | 83        |
| 8.6   | I/O State on Power Up .....                                   | 84        |
| 8.7   | Behavior in Low-Power Modes .....                             | 84        |
| 8.8   | Interrupt .....   | 84        |
| 8.9   | Peripheral Connections .....                                  | 86        |
| 8.9.1   | Firmware Controlled GPIO .....                                | 86        |
| 8.9.2   | Analog I/O .....  | 86        |
| 8.9.3   | LCD Drive .....   | 86        |
| 8.9.4   | CapSense .....  | 86        |
| 8.9.5   | Serial Communication Block (SCB) .....                        | 87        |
| 8.9.6   | Timer, Counter, and Pulse Width Modulator (TCPWM) Block ..... | 87        |
| 8.10  | Registers .....   | 87        |
| <b>9.</b>   | <b>Clocking System</b> .....                                  | <b>88</b> |
| 9.1   | Block Diagram .....   | 88        |
| 9.2   | Clock Sources .....   | 89        |
| 9.2.1   | Internal Main Oscillator (IMO) .....                          | 89        |
| 9.2.1.1   | Startup Behavior .....  | 90        |
| 9.2.1.2   | Programming Clock (36-MHz) .....                              | 90        |

|            |   |            |
|------------|---|------------|
| 9.2.2      | Internal Low-speed Oscillator (ILO) .....           | 90         |
| 9.2.3      | External Clock (EXTCLK) .....                       | 91         |
| 9.2.4      | External Crystal Oscillator (ECO) .....             | 91         |
| 9.2.4.1    | ECO Trimming .....                                  | 91         |
| 9.2.5      | Phase-Locked Loop (PLL) .....                       | 92         |
| 9.2.6      | Watch Crystal Oscillator (WCO) .....                | 92         |
| 9.3        | Clock Supervision .....                             | 93         |
| 9.3.1      | Clock Supervision Block (CSV) .....                 | 93         |
| 9.3.2      | Clock Supervision Interrupt (CSI) .....             | 93         |
| 9.3.3      | Programmable Delay Block (PGM_DELAY) .....          | 93         |
| 9.4        | Clock Distribution .....                            | 93         |
| 9.4.1      | HFCLK and PLL Input Selection .....                 | 94         |
| 9.4.2      | High-Frequency Clock (HFCLK) Input Selection .....  | 95         |
| 9.4.3      | Low-Frequency (LFCLK) Input Selection .....         | 95         |
| 9.4.4      | System Clock (SYSCLK) Prescaler Configuration ..... | 95         |
| 9.4.5      | Peripheral Clock Divider Configuration .....        | 95         |
| 9.5        | Low-Power Mode Operation .....                      | 98         |
| 9.6        | Register List .....                                 | 98         |
| <b>10.</b> | <b>Power Supply and Monitoring</b> .....            | <b>99</b>  |
| 10.1       | Block Diagram .....                                 | 99         |
| 10.2       | Power Supply Scenarios .....                        | 100        |
| 10.2.1     | Single 1.8 V to 5.5 V Unregulated Supply .....      | 100        |
| 10.2.2     | Direct 1.71 V to 1.89 V Regulated Supply .....      | 101        |
| 10.3       | How It Works .....                                  | 101        |
| 10.3.1     | Regulator Summary .....                             | 101        |
| 10.3.1.1   | Active Digital Regulator .....                      | 101        |
| 10.3.1.2   | Deep-Sleep Regulator .....                          | 101        |
| 10.4       | Voltage Monitoring .....                            | 102        |
| 10.4.1     | Power-on-Reset (POR) .....                          | 102        |
| 10.4.1.1   | Brownout-Detect (BOD) .....                         | 102        |
| 10.5       | Register List .....                                 | 102        |
| <b>11.</b> | <b>Chip Operational Modes</b> .....                 | <b>103</b> |
| 11.1       | Boot .....  | 103        |
| 11.2       | User .....  | 103        |
| 11.3       | Privileged .....                                    | 103        |
| 11.4       | Debug .....   | 103        |
| <b>12.</b> | <b>Power Modes</b> .....                            | <b>104</b> |
| 12.1       | Active Mode .....                                   | 105        |
| 12.2       | Sleep Mode .....                                    | 105        |
| 12.3       | Deep-Sleep Mode .....                               | 105        |
| 12.4       | Power Mode Summary .....                            | 106        |
| 12.5       | Low-Power Mode Entry and Exit .....                 | 107        |
| 12.6       | Register List .....                                 | 107        |
| <b>13.</b> | <b>Watchdog Timer</b> .....                         | <b>108</b> |
| 13.1       | Features .....                                      | 108        |
| 13.2       | Block Diagram .....                                 | 108        |
| 13.3       | How It Works .....                                  | 109        |
| 13.3.1     | Enabling and Disabling WDT .....                    | 109        |
| 13.3.2     | WDT Interrupts and Low-Power Modes .....            | 110        |
| 13.3.3     | WDT Reset Mode .....                                | 110        |

|                   |  |            |
|-------------------|--|------------|
| 13.4              | Additional Timers .....                        | 110        |
| 13.4.1            | WDT0 and WDT1 .....                            | 111        |
| 13.4.2            | WDT2 .....                                     | 111        |
| 13.4.3            | Cascading .....                                | 111        |
| 13.5              | Register List .....                            | 112        |
| <b>14.</b>        | <b>Trigger Multiplexer Block</b> .....         | <b>113</b> |
| 14.1              | Features .....                                 | 113        |
| 14.2              | Architecture .....                             | 113        |
| 14.2.1            | Trigger Multiplexer Group .....                | 114        |
| 14.2.2            | Software Triggers .....                        | 116        |
| 14.3              | Register List .....                            | 116        |
| <b>15.</b>        | <b>Reset System</b> .....                      | <b>117</b> |
| 15.1              | Reset Sources .....                            | 117        |
| 15.1.1            | Power-on Reset .....                           | 117        |
| 15.1.2            | Brownout Reset .....                           | 117        |
| 15.1.3            | Watchdog Reset .....                           | 117        |
| 15.1.4            | Software Initiated Reset .....                 | 117        |
| 15.1.5            | External Reset .....                           | 118        |
| 15.1.6            | Protection Fault Reset .....                   | 118        |
| 15.2              | Identifying Reset Sources .....                | 118        |
| 15.3              | Register List .....                            | 118        |
| <b>Section D:</b> | <b>Digital System</b> .....                    | <b>119</b> |
| <b>16.</b>        | <b>Serial Communications Block (SCB)</b> ..... | <b>120</b> |
| 16.1              | Features .....                                 | 120        |
| 16.2              | Architecture .....                             | 120        |
| 16.2.1            | Buffer Modes .....                             | 120        |
| 16.2.1.1          | FIFO Mode .....                                | 120        |
| 16.2.1.2          | EZ Mode .....                                  | 121        |
| 16.2.2            | Clocking Modes .....                           | 121        |
| 16.3              | Serial Peripheral Interface (SPI) .....        | 122        |
| 16.3.1            | Features .....                                 | 122        |
| 16.3.2            | General Description .....                      | 123        |
| 16.3.3            | SPI Modes of Operation .....                   | 124        |
| 16.3.3.1          | Motorola SPI .....                             | 124        |
| 16.3.3.2          | Texas Instruments SPI .....                    | 126        |
| 16.3.3.3          | National Semiconductors SPI .....              | 128        |
| 16.3.4            | SPI Buffer Modes .....                         | 129        |
| 16.3.4.1          | FIFO Mode .....                                | 129        |
| 16.3.4.2          | EZSPI Mode .....                               | 130        |
| 16.3.5            | Clocking and Oversampling .....                | 132        |
| 16.3.5.1          | Clock Modes .....                              | 132        |
| 16.3.5.2          | Using SPI Master to Clock Slave .....          | 134        |
| 16.3.5.3          | Oversampling and Bit Rate .....                | 134        |
| 16.3.6            | Enabling and Initializing SPI .....            | 135        |
| 16.3.7            | I/O Pad Connection .....                       | 136        |
| 16.3.7.1          | SPI Master .....                               | 136        |
| 16.3.7.2          | SPI Slave .....                                | 137        |
| 16.3.7.3          | Glitch Avoidance at System Reset .....         | 138        |
| 16.3.8            | SPI Registers .....                            | 138        |
| 16.4              | UART .....                                     | 139        |

|          |  |     |
|----------|--|-----|
| 16.4.1   | Features .....                                   | 139 |
| 16.4.2   | General Description .....                        | 140 |
| 16.4.3   | UART Modes of Operation .....                    | 140 |
| 16.4.3.1 | Standard Protocol .....                          | 140 |
| 16.4.3.2 | UART Local Interconnect Network (LIN) Mode ..... | 145 |
| 16.4.3.3 | SmartCard (ISO7816) .....                        | 149 |
| 16.4.3.4 | Infrared Data Association (IrDA) .....           | 150 |
| 16.4.4   | Clocking and Oversampling .....                  | 150 |
| 16.4.5   | Enabling and Initializing the UART .....         | 151 |
| 16.4.6   | I/O Pad Connection .....                         | 151 |
| 16.4.6.1 | Standard UART Mode .....                         | 151 |
| 16.4.6.2 | SmartCard Mode .....                             | 152 |
| 16.4.6.3 | LIN Mode .....                                   | 152 |
| 16.4.6.4 | IrDA Mode .....                                  | 153 |
| 16.4.7   | UART Registers .....                             | 153 |
| 16.5     | Inter Integrated Circuit (I2C) .....             | 154 |
| 16.5.1   | Features .....                                   | 154 |
| 16.5.2   | General Description .....                        | 154 |
| 16.5.3   | External Electrical Connections .....            | 155 |
| 16.5.4   | Terms and Definitions .....                      | 156 |
| 16.5.4.1 | Clock Stretching .....                           | 156 |
| 16.5.4.2 | Bus Arbitration .....                            | 156 |
| 16.5.5   | I2C Modes of Operation .....                     | 157 |
| 16.5.5.1 | Write Transfer .....                             | 157 |
| 16.5.5.2 | Read Transfer .....                              | 158 |
| 16.5.6   | I2C Buffer Modes .....                           | 159 |
| 16.5.6.1 | FIFO Mode .....                                  | 159 |
| 16.5.6.2 | EZI2C Mode .....                                 | 160 |
| 16.5.7   | Clocking and Oversampling .....                  | 162 |
| 16.5.7.1 | Glitch Filtering .....                           | 163 |
| 16.5.7.2 | Oversampling and Bit Rate .....                  | 164 |
| 16.5.8   | Enabling and Initializing the I2C .....          | 166 |
| 16.5.8.1 | Configuring for I2C FIFO Mode .....              | 166 |
| 16.5.8.2 | Configuring for EZ Mode .....                    | 166 |
| 16.5.9   | I/O Pad Connections .....                        | 167 |
| 16.5.10  | I2C Registers .....                              | 168 |
| 16.6     | SCB Interrupts .....                             | 169 |
| 16.6.1   | SPI Interrupts .....                             | 170 |
| 16.6.2   | UART Interrupts .....                            | 174 |
| 16.6.3   | I2C Interrupts .....                             | 177 |

## **17. CAN FD Controller 179**

|          |                               |     |
|----------|-------------------------------|-----|
| 17.1     | Overview .....                | 179 |
| 17.1.1   | Features .....                | 179 |
| 17.1.2   | Features Not Supported .....  | 180 |
| 17.2     | Configuration .....           | 180 |
| 17.2.1   | Block Diagram .....           | 180 |
| 17.2.2   | Clock Sources .....           | 180 |
| 17.2.3   | Interrupt Lines .....         | 180 |
| 17.3     | Functional Description .....  | 182 |
| 17.3.1   | Operation Modes .....         | 182 |
| 17.3.1.1 | Software Initialization ..... | 182 |
| 17.3.1.2 | Normal Operation .....        | 183 |

|           |   |     |
|-----------|---|-----|
| 17.3.1.3  | CAN FD Operation .....  | 183 |
| 17.3.1.4  | Transmitter Delay Compensation .....  | 184 |
| 17.3.1.5  | Restricted Operation mode .....   | 185 |
| 17.3.1.6  | Bus Monitoring Mode .....   | 186 |
| 17.3.1.7  | Disable Automatic Retransmission .....  | 186 |
| 17.3.1.8  | Power Down (Sleep Mode) .....   | 187 |
| 17.3.1.9  | Test Mode .....   | 187 |
| 17.3.1.10 | Application Watchdog .....  | 188 |
| 17.3.2    | Timestamp Generation .....  | 188 |
| 17.3.3    | Timeout Counter .....   | 189 |
| 17.3.4    | RX Handling .....   | 189 |
| 17.3.4.1  | Acceptance Filtering .....  | 189 |
| 17.3.4.2  | RX FIFOs .....  | 193 |
| 17.3.4.3  | Dedicated RX Buffers .....  | 195 |
| 17.3.4.4  | Debug on CAN Support .....  | 196 |
| 17.3.5    | TX Handling .....   | 197 |
| 17.3.5.1  | Transmit Pause .....  | 198 |
| 17.3.5.2  | Dedicated TX Buffers .....  | 198 |
| 17.3.5.3  | TX FIFO .....   | 199 |
| 17.3.5.4  | TX Queue .....  | 199 |
| 17.3.5.5  | Mixed Dedicated TX Buffers/TX FIFO .....                                      | 200 |
| 17.3.5.6  | Mixed Dedicated TX Buffers/TX Queue .....                                     | 200 |
| 17.3.5.7  | Transmit Cancellation .....   | 201 |
| 17.3.5.8  | TX Event Handling .....   | 201 |
| 17.3.6    | FIFO Acknowledge Handling .....   | 201 |
| 17.3.7    | Configuring the CAN Bit Timing .....  | 202 |
| 17.3.7.1  | CAN Bit Timing .....  | 202 |
| 17.3.7.2  | CAN Bit Rates .....   | 204 |
| 17.4      | Message RAM .....   | 205 |
| 17.4.1    | Message RAM Configuration .....   | 205 |
| 17.4.2    | RX Buffer and FIFO Element .....  | 206 |
| 17.4.3    | TX Buffer Element .....   | 208 |
| 17.4.4    | TX Event FIFO Element .....   | 210 |
| 17.4.5    | Standard Message ID Filter Element .....                                      | 212 |
| 17.4.6    | Extended Message ID Filter Element .....                                      | 213 |
| 17.4.7    | Trigger Memory Element .....  | 215 |
| 17.4.8    | Message RAM OFF .....   | 217 |
| 17.4.9    | RAM Watchdog (RWD) .....  | 217 |
| 17.4.10   | Address Error .....   | 217 |
| 17.5      | TTCAN Operation .....   | 218 |
| 17.5.1    | Reference Message .....   | 218 |
| 17.5.1.1  | Level 1 .....   | 218 |
| 17.5.1.2  | Level 2 .....   | 218 |
| 17.5.1.3  | Level 0 .....   | 219 |
| 17.5.2    | TTCAN Configuration .....   | 219 |
| 17.5.2.1  | TTCAN Timing .....  | 219 |
| 17.5.2.2  | Message Scheduling .....  | 220 |
| 17.5.2.3  | Trigger Memory .....  | 221 |
| 17.5.2.4  | TTCAN Schedule Initialization .....   | 225 |
| 17.5.3    | TTCAN Gap Control .....   | 226 |
| 17.5.4    | Stop Watch .....  | 226 |
| 17.5.5    | Local Time, Cycle Time, Global Time, and External Clock Synchronization ..... | 227 |
| 17.5.6    | Synchronization Triggers .....  | 229 |

|           |   |     |
|-----------|---|-----|
| 17.5.7    | TTCAN Error Level .....                         | 229 |
| 17.5.8    | TTCAN Message Handling .....                    | 230 |
| 17.5.8.1  | Reference Message .....                         | 230 |
| 17.5.8.2  | Message Reception .....                         | 231 |
| 17.5.8.3  | Message Transmission .....                      | 231 |
| 17.5.9    | TTCAN Interrupt and Error Handling .....        | 233 |
| 17.5.10   | Level 0 .....                                   | 233 |
| 17.5.10.1 | Synchronizing .....                             | 234 |
| 17.5.10.2 | Handling Error Levels .....                     | 234 |
| 17.5.10.3 | Master Slave Relation .....                     | 235 |
| 17.5.11   | Synchronization to External Time Schedule ..... | 235 |
| 17.6      | Setup Procedures .....                          | 236 |
| 17.6.1    | General Program Flow .....                      | 236 |
| 17.6.2    | Clock Stop Request .....                        | 236 |
| 17.6.3    | Message RAM OFF Operation .....                 | 237 |
| 17.6.4    | Message RAM ON Operation .....                  | 237 |
| 17.6.5    | Consolidated Interrupt Handling .....           | 237 |
| 17.6.6    | Procedures Specific to M_TTCAN Channel .....    | 238 |
| 17.6.6.1  | CAN Bus Configuration .....                     | 239 |
| 17.6.6.2  | Message RAM Configuration .....                 | 239 |
| 17.6.6.3  | Interrupt Configuration .....                   | 242 |
| 17.6.6.4  | Transmit Frame Configuration .....              | 243 |
| 17.6.6.5  | Interrupt Handling .....                        | 244 |

## **18. Timer, Counter, and PWM (TCPWM) 249**

|          |  |     |
|----------|--|-----|
| 18.1     | Features .....   | 249 |
| 18.2     | Architecture .....                                       | 250 |
| 18.2.1   | Enabling and Disabling Counters in a TCPWM Block .....   | 250 |
| 18.2.2   | Clocking .....   | 250 |
| 18.2.2.1 | Clock Prescaling .....                                   | 250 |
| 18.2.2.2 | Count Input .....  | 251 |
| 18.2.3   | Trigger Inputs .....                                     | 251 |
| 18.2.4   | Trigger Outputs .....                                    | 254 |
| 18.2.5   | Interrupts .....   | 254 |
| 18.2.6   | PWM Outputs .....  | 254 |
| 18.2.7   | Power Modes .....  | 255 |
| 18.3     | Operation Modes .....                                    | 255 |
| 18.3.1   | Timer Mode .....   | 256 |
| 18.3.1.1 | Configuring Counter for Timer Mode .....                 | 262 |
| 18.3.2   | Capture Mode .....                                       | 262 |
| 18.3.2.1 | Configuring Counter for Capture Mode .....               | 265 |
| 18.3.3   | Quadrature Decoder Mode .....                            | 265 |
| 18.3.3.1 | Configuring Counter for Quadrature Mode .....            | 269 |
| 18.3.4   | Pulse Width Modulation Mode .....                        | 269 |
| 18.3.4.1 | Asymmetric PWM .....                                     | 277 |
| 18.3.4.2 | Configuring Counter for PWM Mode .....                   | 278 |
| 18.3.5   | Pulse Width Modulation with Dead Time Mode .....         | 279 |
| 18.3.5.1 | Configuring Counter for PWM with Dead Time Mode .....    | 281 |
| 18.3.6   | Pulse Width Modulation Pseudo-Random Mode (PWM_PR) ..... | 281 |
| 18.3.6.1 | Configuring Counter for Pseudo-Random PWM Mode .....     | 284 |
| 18.4     | TCPWM Registers .....                                    | 284 |

|  |            |
|--|------------|
| <b>19. LCD Direct Drive</b>                        | <b>285</b> |
| 19.1 Features.....                                 | 285        |
| 19.2 LCD Segment Drive Overview.....               | 285        |
| 19.2.1 Drive Modes.....                            | 286        |
| 19.2.1.1 PWM Drive.....                            | 286        |
| 19.2.2 Recommended Bias Settings.....              | 291        |
| 19.2.3 Digital Contrast Control.....               | 291        |
| 19.3 Block Diagram.....                            | 292        |
| 19.3.1 How it Works.....                           | 292        |
| 19.3.2 High-Speed Master Generator.....            | 292        |
| 19.3.3 LCD Pin Logic.....                          | 293        |
| 19.3.4 Display Data Registers.....                 | 293        |
| 19.4 Register List.....                            | 293        |
| <b>20. Inter-IC Sound Bus</b>                      | <b>294</b> |
| 20.1 Features.....                                 | 294        |
| 20.2 Architecture.....                             | 294        |
| 20.3 Digital Audio Interface Formats.....          | 295        |
| 20.3.1 Standard I2S Format.....                    | 295        |
| 20.3.2 Left Justified (LJ) Format.....             | 297        |
| 20.3.3 Time Division Multiplexed (TDM) Format..... | 297        |
| 20.4 Clocking Features.....                        | 298        |
| 20.5 FIFO Buffer.....                              | 299        |
| 20.6 Interrupt Support.....                        | 300        |
| <b>21. CRYPTO</b>                                  | <b>301</b> |
| 21.1 Features.....                                 | 301        |
| 21.2 Overview.....                                 | 301        |
| 21.3 Trigger Interface.....                        | 302        |
| 21.4 Memory Buffer.....                            | 302        |
| 21.4.1 Access Control.....                         | 302        |
| 21.5 Pseudo Random Number Generator.....           | 303        |
| 21.5.1 Configuring PRNG.....                       | 304        |
| 21.6 True Random Number Generator (TRNG).....      | 304        |
| 21.7 Advance Encryption Standard (AES).....        | 308        |
| 21.7.1 Modes of Operation.....                     | 309        |
| 21.7.2 Configuring AES.....                        | 310        |
| 21.8 SHA.....                                      | 310        |
| 21.8.1 Configuring SHA.....                        | 311        |
| 21.9 CRC.....                                      | 312        |
| 21.9.1 Configuring CRC.....                        | 313        |
| 21.10 Power Modes.....                             | 313        |
| <b>Section E: Analog System</b>                    | <b>314</b> |
| <b>22. SAR ADC</b>                                 | <b>315</b> |
| 22.1 Features.....                                 | 315        |
| 22.2 Block Diagram.....                            | 316        |
| 22.3 How it Works.....                             | 317        |
| 22.3.1 SAR ADC Core.....                           | 317        |
| 22.3.1.1 Single-ended and Differential Mode.....   | 317        |
| 22.3.1.2 Input Range.....                          | 317        |
| 22.3.1.3 Result Data Format.....                   | 318        |
| 22.3.1.4 Negative Input Selection.....             | 318        |

|            |   |            |
|------------|---|------------|
| 22.3.1.5   | Resolution .....  | 319        |
| 22.3.1.6   | Acquisition Time .....                                    | 319        |
| 22.3.1.7   | SAR ADC Clock .....                                       | 320        |
| 22.3.1.8   | SAR ADC Timing.....                                       | 321        |
| 22.3.2     | SARMUX.....   | 321        |
| 22.3.3     | SARREF .....  | 323        |
| 22.3.3.1   | Reference Options .....                                   | 323        |
| 22.3.3.2   | Bypass Capacitors .....                                   | 323        |
| 22.3.3.3   | Input Range versus Reference.....                         | 324        |
| 22.3.4     | SARSEQ.....   | 324        |
| 22.3.4.1   | Averaging .....   | 325        |
| 22.3.4.2   | Range Detection.....                                      | 325        |
| 22.3.4.3   | Double Buffer .....                                       | 325        |
| 22.3.4.4   | Injection Channel.....                                    | 326        |
| 22.3.5     | SAR Interrupts .....                                      | 328        |
| 22.3.5.1   | End-of-Scan Interrupt (EOS_INTR).....                     | 328        |
| 22.3.5.2   | Overflow Interrupt.....                                   | 328        |
| 22.3.5.3   | Collision Interrupt .....                                 | 328        |
| 22.3.5.4   | Injection End-of-Conversion Interrupt (INJ_EOC_INTR)..... | 329        |
| 22.3.5.5   | Range Detection Interrupts .....                          | 329        |
| 22.3.5.6   | Saturate Detection Interrupts .....                       | 329        |
| 22.3.5.7   | Interrupt Cause Overview.....                             | 329        |
| 22.3.6     | Trigger.....  | 330        |
| 22.3.6.1   | External Trigger Configuration .....                      | 331        |
| 22.3.7     | SAR ADC Status .....                                      | 332        |
| 22.3.8     | Low-Power Mode .....                                      | 332        |
| 22.3.9     | System Operation .....                                    | 333        |
| 22.3.9.1   | SARMUX Analog Routing .....                               | 333        |
| 22.3.9.2   | Global SARSEQ Configuration.....                          | 334        |
| 22.3.9.3   | Channel Configurations.....                               | 335        |
| 22.3.9.4   | Channel Enables .....                                     | 335        |
| 22.3.9.5   | Interrupt Masks.....                                      | 336        |
| 22.3.9.6   | Trigger .....   | 336        |
| 22.3.9.7   | Retrieve Data after Each Interrupt.....                   | 336        |
| 22.3.9.8   | Injection Conversions .....                               | 336        |
| 22.3.10    | Temperature Sensor Configuration .....                    | 337        |
| 22.4       | Registers.....  | 338        |
| <b>23.</b> | <b>Low-Power Comparator</b>                               | <b>339</b> |
| 23.1       | Features.....   | 339        |
| 23.2       | Block Diagram .....                                       | 339        |
| 23.3       | How It Works .....  | 340        |
| 23.3.1     | Input Configuration.....                                  | 340        |
| 23.3.2     | Output and Interrupt Configuration .....                  | 340        |
| 23.3.3     | Power Mode and Speed Configuration .....                  | 341        |
| 23.3.4     | Hysteresis .....  | 342        |
| 23.3.5     | Wakeup from Low-Power Modes .....                         | 342        |
| 23.3.6     | Comparator Clock .....                                    | 343        |
| 23.3.7     | Offset Trim .....   | 343        |
| 23.4       | Register Summary .....                                    | 343        |
| <b>24.</b> | <b>Continuous Time Block mini (CTBm)</b>                  | <b>344</b> |
| 24.1       | Features.....   | 344        |

|                   |   |            |
|-------------------|---|------------|
| 24.2              | Block Diagram .....                             | 345        |
| 24.3              | How It Works .....                              | 346        |
| 24.3.1            | Power Mode Configuration .....                  | 346        |
| 24.3.2            | Output Strength Configuration .....             | 346        |
| 24.3.3            | Compensation .....                              | 347        |
| 24.3.4            | Switch Control .....                            | 348        |
| 24.3.4.1          | Input Configuration .....                       | 348        |
| 24.3.4.2          | Output Configuration .....                      | 348        |
| 24.3.4.3          | Comparator Mode .....                           | 349        |
| 24.3.4.4          | Comparator Configuration .....                  | 349        |
| 24.3.4.5          | Comparator Interrupt .....                      | 350        |
| 24.3.4.6          | Deep-Sleep Mode Operation .....                 | 350        |
| 24.4              | Register Summary .....                          | 352        |
| <b>25.</b>        | <b>CapSense</b> .....                           | <b>353</b> |
| <b>26.</b>        | <b>Temperature Sensor</b> .....                 | <b>354</b> |
| 26.1              | Features .....                                  | 354        |
| 26.2              | How it Works .....                              | 354        |
| 26.3              | Temperature Sensor Configuration .....          | 356        |
| 26.4              | Algorithm .....                                 | 356        |
| 26.5              | Registers .....                                 | 356        |
| <b>27.</b>        | <b>Analog Routing</b> .....                     | <b>357</b> |
| 27.1              | Features .....                                  | 357        |
| 27.2              | Architecture .....                              | 357        |
| 27.2.1            | Analog Interconnection .....                    | 359        |
| 27.2.2            | AMUXBUS Splitting .....                         | 364        |
| 27.3              | Register List .....                             | 364        |
| <b>Section F:</b> | <b>Program and Debug</b> .....                  | <b>365</b> |
| <b>28.</b>        | <b>Program and Debug Interface</b> .....        | <b>366</b> |
| 28.1              | Features .....                                  | 366        |
| 28.2              | Functional Description .....                    | 366        |
| 28.3              | Serial Wire Debug (SWD) Interface .....         | 367        |
| 28.3.1            | SWD Timing Details .....                        | 368        |
| 28.3.2            | ACK Details .....                               | 368        |
| 28.3.3            | Turnaround (Trn) Period Details .....           | 369        |
| 28.4              | Cortex-M0+ Debug and Access Port (DAP) .....    | 369        |
| 28.4.1            | Debug Port (DP) Registers .....                 | 369        |
| 28.4.2            | Access Port (AP) Registers .....                | 370        |
| 28.5              | Programming the PSoC 4 Device .....             | 371        |
| 28.5.1            | SWD Port Acquisition .....                      | 371        |
| 28.5.1.1          | SWD Port Acquire Sequence .....                 | 371        |
| 28.5.2            | SWD Programming Mode Entry .....                | 371        |
| 28.5.3            | SWD Programming Routines Executions .....       | 371        |
| 28.6              | PSoC 4 SWD Debug Interface .....                | 372        |
| 28.6.1            | Debug Control and Configuration Registers ..... | 372        |
| 28.6.2            | Breakpoint Unit (BPU) .....                     | 372        |
| 28.6.3            | Data Watchpoint (DWT) .....                     | 372        |
| 28.6.4            | Debugging the PSoC 4 Device .....               | 373        |
| 28.7              | Registers .....                                 | 373        |

|  |            |
|--|------------|
| <b>29. Nonvolatile Memory Programming</b>        | <b>374</b> |
| 29.1 Features.....                               | 374        |
| 29.2 Functional Description .....                | 374        |
| 29.3 System Call Implementation .....            | 375        |
| 29.4 Blocking and Non-Blocking System Calls..... | 375        |
| 29.4.1 Performing a System Call .....            | 376        |
| 29.5 System Calls.....                           | 377        |
| 29.5.1 Silicon ID.....                           | 378        |
| 29.5.2 Configure Clock .....                     | 378        |
| 29.5.3 Load Flash Bytes .....                    | 379        |
| 29.5.4 Write Row .....                           | 380        |
| 29.5.5 Program Row .....                         | 381        |
| 29.5.6 Erase All.....                            | 382        |
| 29.5.7 Checksum .....                            | 383        |
| 29.5.8 Write Protection .....                    | 384        |
| 29.5.9 Non-Blocking Write Row .....              | 385        |
| 29.5.10 Non-Blocking Program Row.....            | 386        |
| 29.5.11 Resume Non-Blocking .....                | 387        |
| 29.6 System Call Status .....                    | 388        |
| 29.7 Non-Blocking System Call Pseudo Code .....  | 389        |

# Section A: Overview



This section encompasses the following chapters:

- [Introduction chapter on page 18](#)
- [Getting Started chapter on page 24](#)
- [Document Construction chapter on page 26](#)

## Document Revision History

| Revision | Issue Date    | Description of Change                  |
|----------|---------------|--|
| **       | July 16, 2021 | Initial version of PSoC 4100S Max TRM. |

# 1. Introduction



PSoC<sup>®</sup> 4 is a programmable embedded system controller with an Arm<sup>®</sup> Cortex<sup>®</sup>-M0+ CPU. PSoC 4100S Max device is an enhanced version of the PSoC 4000 family and is upward-compatible with larger members of PSoC 4.

PSoC 4 devices have these characteristics:

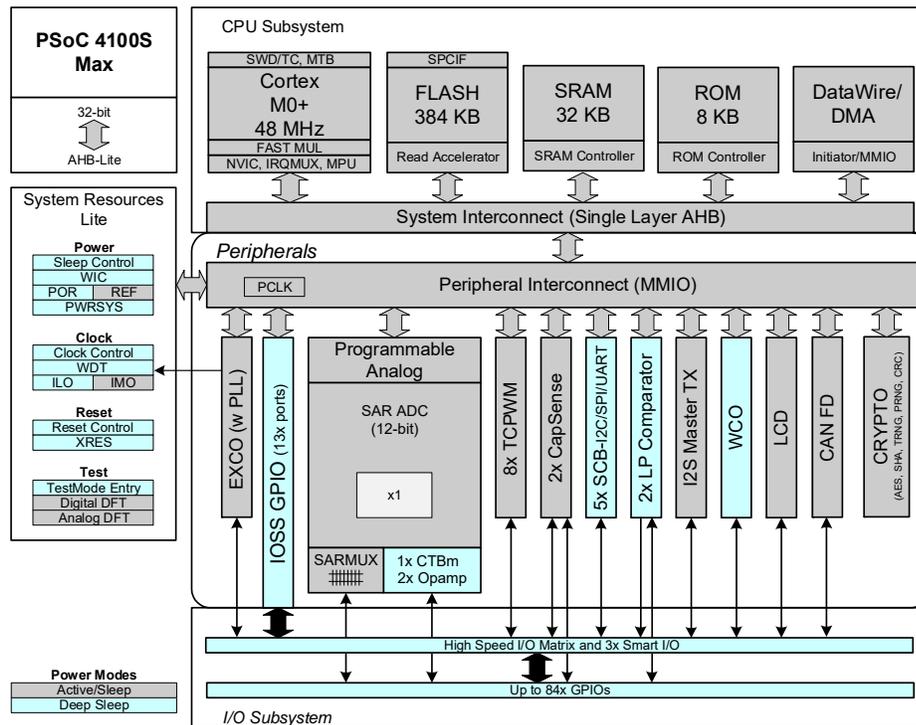
- High-performance, 32-bit single-cycle Cortex-M0+ CPU core
- High-performance analog system
- Self and Mutual Capacitive touch sensing (CapSense<sup>®</sup>)
- Configurable Timer/Counter/PWM block
- Configurable analog blocks for analog signal conditioning
- Configurable communication block with I<sup>2</sup>C, SPI, and UART operating modes
- Low-power operating modes – Sleep and Deep-Sleep

This document describes each functional block of the PSoC 4100S Max 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 4100S Max architecture.

Figure 1-1. PSoC 4100S Max Block Diagram



## 1.2 Features

The PSoC 4100S Max device has these major components:

- 32-bit Cortex-M0+ CPU with single-cycle multiply, delivering up to 0.9 DMIPS/MHz
- Up to 384 KB flash and 32 KB SRAM
- Direct memory access (DMA)
- Cryptographic Accelerator with symmetric cryptography (AES), Hash (SHA), True Random Number Generation and CRC capabilities.
- Up to eight center-aligned pulse-width modulator (PWM) with complementary, dead-band programmable outputs
- One watchdog timer and three general-purpose timers with interrupt capability
- Twelve-bit SAR ADC (with a sampling rate of 1 Msps) with hardware sequencing for multiple channels
- Up to two opamps that can be used for analog signal conditioning and as a comparator. Opamps can operate in Deep Sleep low-power mode.
- Two low-power comparators
- Up to five serial communication blocks (SCB) that can work as SPI, UART, I<sup>2</sup>C, and local interconnect network (LIN) slave serial communication channels
- One controller area network (CAN FD) block
- Three Smart I/O blocks, which provides the ability to perform Boolean functions in the I/O signal path
- Two CapSense (Fifth-Generation) blocks with synchronized sampling.
- Segment LCD direct drive
- Low-power operating modes: Sleep and Deep-Sleep
- Programming and debugging system through serial wire debug (SWD)
- Fully supported by ModusToolbox IDE tool
- One I2S TX Mode Master block

## 1.3 CPU System

### 1.3.1 Processor

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

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

### 1.3.2 Interrupt Controller

The CPU subsystem includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode.

### 1.3.3 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.

### 1.3.4 Cryptographic Accelerator (CRYPTO)

The CRYPTO Accelerator block will support 128-bit AES, all SHA modes, True Random Number and Pseudo Random Number Generator function, and a CRC function. It will incorporate a 512 byte instruction and operand storage buffer.

## 1.3.5 Memory

### 1.3.5.1 Flash

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

### 1.3.5.2 SRAM

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

## 1.4 System-Wide Resources

### 1.4.1 Clocking System

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

The IMO with an accuracy of  $\pm 2$  percent is the primary source of internal clocking in the device. The default IMO frequency is 24 MHz and can be adjusted between 24 MHz and 48 MHz in steps of 4 MHz. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a low-power, less accurate oscillator and is used as a source for LFCLK, to generate clocks for peripheral operation in Deep-Sleep mode. Its clock frequency is 40 kHz with  $\pm 60$  percent accuracy.

An external clock source ranging from 1 MHz to 48 MHz can be used to generate the clock derivatives for the functional blocks instead of the IMO.

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

The WCO is a 32-kHz watch crystal oscillator. It is used to dynamically trim the IMO to an accuracy of  $\pm 1$  percent to enable precision timing applications.

### 1.4.2 Power System

The device operates with a single external supply in the range 1.71 V to 5.5 V. It provides multiple power supply domains –  $V_{DD}$  to power the digital section, and  $V_{DDA}$  for noise isolation of the analog section.  $V_{DD}$  and  $V_{DDA}$  should be shorted externally.

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

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

### 1.4.3 GPIO

Every GPIO 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

In addition, the device has up to three Smart I/O blocks that provides the ability to perform Boolean functions on the port I/Os. The Smart I/O block is available in all device power modes, including low-power modes.

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

#### 1.4.4 Watchdog Timers

The PSoC 4 device has one 16-bit watchdog timer, which is capable of automatically resetting the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality.

In addition to this, two 16-bit and one 32-bit up-counting timers are available for general-purpose use.

## 1.5 Fixed-Function Digital

### 1.5.1 Timer/Counter/PWM Block

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

### 1.5.2 Serial Communication Blocks

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

The features of each SCB include:

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

### 1.5.3 Controller Area Network (CAN)

One CAN Flexible Data-rate (CAN FD) block is provided in the PSoC 4100S Max device, which will be certifiable to be Bosch CAN standard compliant and will operate at 5 Mbps. It will incorporate a 4KB receive and transmit buffer.

### 1.5.4 LCD Segment Drive

The PSoC 4100S Max 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.

### 1.5.5 I2S Master

The PSoC 4100S Max device has an I2S TX Master block, which is both electrically and protocol compliant to standard I2S Specification. Also supports the left justified audio format.

## 1.6 Analog System

### 1.6.1 SAR ADC

The PSoC 4100S Max device 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.

### 1.6.2 Continuous Time Block mini (CTBm)

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. The PSoC 4100S Max device has one such CTBm block.

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

### 1.6.3 Low-Power Comparators

The PSoC 4100S Max device 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 AMUXBUS.

## 1.7 Special Function Peripherals

### 1.7.1 CapSense

The PSoC 4100S Max device has two Fifth-Generation CapSense blocks with synchronous scanning. CapSense functionality is supported on all GPIO pins, in self-capacitance and mutual-capacitance modes. Also includes dedicated IO's up to 16 IO per block to improve the performance and enable autonomous operation.

## 1.8 Program and Debug

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

## 1.9 Device Feature Summary

Table 1-1 shows the PSoC 4100S Max devices summary.

Table 1-1. PSoC 4100S Max Device Summary

| Features                          | PSoC 4100S Max                |
|-----------------------------------|-------------------------------|
| Maximum CPU Frequency             | 48 MHz                        |
| Flash                             | 384 KB                        |
| SRAM                              | 32 KB                         |
| DMA                               | 16 channel                    |
| GPIOs (max)                       | 84                            |
| Smart I/O                         | 3 ports                       |
| CapSense                          | 2 channel                     |
| LCD Driver                        | Available                     |
| Timer, Counter, PWM (TCPWM)       | 8                             |
| 16-bit Timer                      | 2                             |
| 32-bit Timer                      | 1                             |
| Serial Communication Block (SCB)  | 5                             |
| Opamp (CTBm)                      | 2                             |
| Low-Power Comparator (LPCOMP)     | 2                             |
| SAR ADC                           | 12-bit, 1 Msps                |
| Watch Crystal Oscillator (WCO)    | Available                     |
| External Crystal Oscillator (ECO) | Available                     |
| Power Modes                       | Active, Sleep, and Deep-Sleep |
| CAN                               | 5 Mbps                        |
| CRYPTO                            | AES, SHA, TRNG, PRNG, CRC     |
| I2S                               | I2S TX Master                 |

## 2. Getting Started



### 2.1 Support

Free support for PSoC 4 products is available online at [www.cypress.com/psoc4](http://www.cypress.com/psoc4). Resources include training seminars, discussion forums, application notes, PSoC consultants, CRM technical support email, knowledge base, and application support engineers.

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

### 2.2 Development Ecosystem

#### 2.2.1 PSoC 4 MCU Resources

Cypress provides a wealth of data at [www.cypress.com](http://www.cypress.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](#), [PSoC Roadmap](#)
- **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
  - [AN91184](#): PSoC<sup>®</sup> 4-BLE - Designing Bluetooth<sup>®</sup> Low Energy Applications
  - [AN88619](#): PSoC 4 Hardware Design Considerations
  - [AN73854](#): PSoC - Introduction To Bootloaders
  - [AN89610](#): Arm<sup>®</sup> Cortex<sup>®</sup> Code Optimization
  - [AN86233](#): PSoC 4 MCU Power Reduction Techniques
  - [AN57821](#): Mixed Signal Circuit Board Layout
  - [AN85951](#): PSoC 4 and PSoC 6 MCU CapSense Design Guide
- **Code Examples** demonstrate product features and usage, and are also available on [Cypress 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 nonvolatile memory.
- **Development Tools**
  - [ModusToolbox™ Software](#) enables cross platform code development with a robust suite of tools and software libraries.
  - [CY8CKIT-041S-Max](#) PSoC<sup>®</sup> 4100S Max Pioneer Kit is a low-cost hardware platform that enables design and debug of the PSoC 4100S Max device.
  - [MiniProg4](#) and [MiniProg3](#) 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 4 MCU 101 series](#).
  - [Cypress 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](#).

## 2.2.2 ModusToolbox™ Software

**ModusToolbox Software** is Cypress' comprehensive collection of multi-platform tools and software libraries that enable an immersive development experience for creating converged MCU and wireless systems. It is:

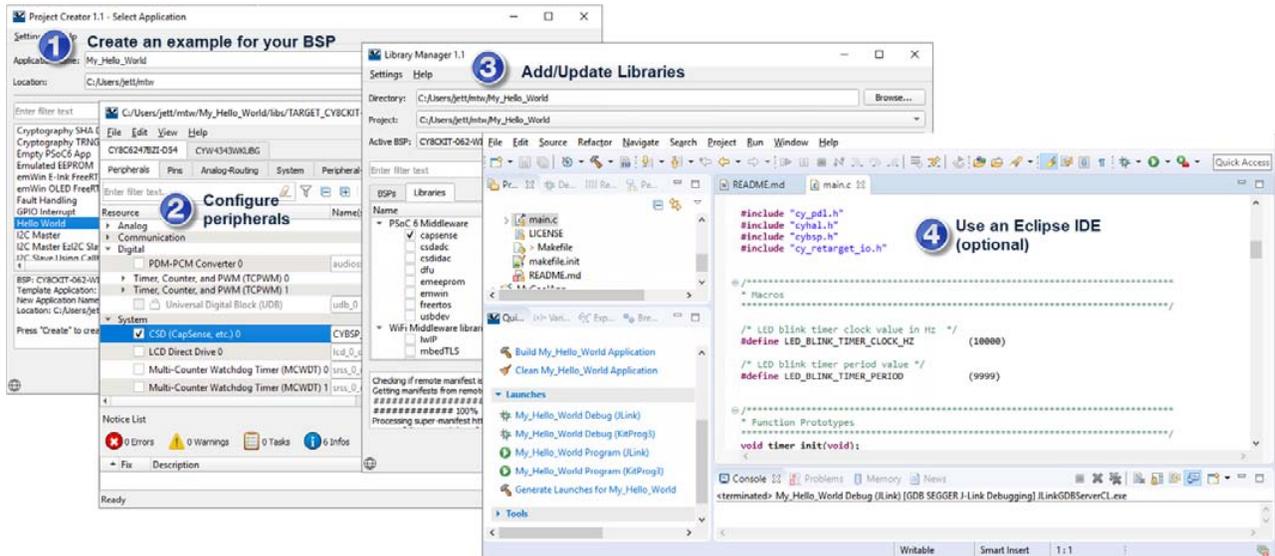
- Comprehensive - it has the resources you need
- Flexible - you can use the resources in your own workflow
- Atomic - you can get just the resources you want

Cypress provides a large collection of code [repositories on GitHub](#), including:

- Board Support Packages (BSPs) aligned with Cypress kits
- Low-level resources, including a peripheral driver library (PDL)
- Middleware enabling industry-leading features such as CapSense
- An extensive set of thoroughly tested [code example applications](#)

ModusToolbox Software is IDE-neutral and easily adaptable to your workflow and preferred development environment. It includes a project creator, peripheral and library configurators, a library manager, as well as the optional Eclipse IDE for ModusToolbox, as [Figure 2-1](#) shows. For information on using Cypress tools, refer to the documentation delivered with ModusToolbox Software, and [AN79953: Getting Started with PSoC 4](#).

Figure 2-1. ModusToolbox Software Tools



# 3. Document Construction



This document includes the following sections:

- [Section B: CPU System on page 30](#)
- [Section C: System Resources Subsystem \(SRSS\) on page 65](#)
- [Section D: Digital System on page 119](#)
- [Section E: Analog System on page 314](#)
- [Section F: Program and Debug on page 365](#)

## 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.
- **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 4100S Max: PSoC 4 Registers TRM](#).

### 3.2.2 Numeric Naming

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

### 3.2.3 Units of Measure

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

Table 3-1. Units of Measure

| Abbreviation | Unit of Measure                                 |
|--------------|---|
| bps          | bits per second                                 |
| °C           | degrees Celsius                                 |
| dB           | decibels  |
| fF           | femtofarads                                     |
| Hz           | Hertz   |
| k            | kilo, 1000                                      |
| K            | kilo, 2 <sup>10</sup>                           |
| 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  |
| AES     | advanced encryption standard   |
| AHB     | AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm® data transfer bus |
| API     | application programming interface  |
| APOR    | analog power-on reset  |
| BC      | broadcast clock  |
| BOD     | brownout detect  |
| BOM     | bill of materials  |
| BR      | bit rate   |
| BRA     | bus request acknowledge  |
| BRQ     | bus request  |
| CAN     | controller area network  |
| CAN FD  | CAN flexible data-rate   |
| CBC     | cipher block chaining  |
| CFB     | cipher feedback  |
| CI      | carry in   |
| CMP     | compare  |
| CO      | carry out  |
| COM     | LCD common signal  |
| CPU     | central processing unit  |
| CRC     | cyclic redundancy check  |
| CRYPTO  | cryptographic accelerator  |
| CSD     | CapSense sigma delta   |
| CT      | continuous time  |
| CTB     | continuous time block  |
| CTBm    | continuous time block mini   |
| CTR     | counter  |
| DAC     | digital-to-analog converter  |
| DAP     | debug access port  |
| DAS     | digitized analog signal  |
| DC      | direct current   |
| DI      | digital or data input  |
| DMA     | direct memory access   |
| DMIPS   | Dhrystone million instructions per second  |
| DO      | digital or data output   |
| DSI     | digital signal interface   |
| DSM     | deep-sleep mode  |
| DW      | data wire  |

Table 3-2. Acronyms (continued)

| Acronym          | Definition  |
|------------------|---|
| ECB              | electronic code book                                |
| 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 <sup>2</sup> C | 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   |
| 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                                 |
| LFSR             | linear feedback shift registers                     |
| 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                                 |
| MPU              | memory protection unit                              |
| MSb              | most significant bit                                |
| MSB              | most significant byte                               |
| MSC              | multi sense converter                               |

Table 3-2. Acronyms (continued)

| Acronym | Definition                            |
|---------|---------------------------------------|
| MSP     | main stack pointer                    |
| NMI     | non-maskable interrupt                |
| NVIC    | nested vectored interrupt controller  |
| OFB     | output feedback                       |
| 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              |
| PRNG    | pseudo random number generator        |
| PRS     | pseudo random sequence                |
| PSoC®   | Programmable System-on-Chip           |
| PSP     | process stack pointer                 |
| 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                      |
| RTC     | real-time clock                       |
| RW      | read/write                            |
| SAR     | successive approximation register     |
| SEG     | LCD segment signal                    |
| SC      | switched capacitor                    |
| SCB     | serial communication block            |
| SIE     | serial interface engine               |
| SIO     | special I/O                           |
| SE0     | single-ended zero                     |
| SHA     | secure hash algorithm                 |
| 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 |

Table 3-2. Acronyms (continued)

| Acronym | Definition                                  |
|---------|---|
| SPIS    | serial peripheral interconnect slave        |
| SRAM    | static random-access memory                 |
| SROM    | supervisory read only memory                |
| SSADC   | single slope ADC                            |
| SSC     | supervisory system call                     |
| SYSCCLK | system clock                                |
| SWD     | single wire debug                           |
| TC      | terminal count                              |
| TCPWM   | timer, counter, PWM                         |
| TD      | transaction descriptors                     |
| TIA     | trans-impedance amplifier                   |
| TRNG    | true random number generator                |
| TX      | Transmitter                                 |
| UART    | universal asynchronous receiver/transmitter |
| UDB     | universal digital block                     |
| USB     | universal serial bus                        |
| USBIO   | USB I/O                                     |
| VTOR    | vector table offset register                |
| WCO     | watch crystal oscillator                    |
| WDT     | watchdog timer                              |
| WDR     | watchdog reset                              |
| XRES    | external reset                              |
| XRES_N  | external reset, active low                  |

# Section B: CPU System

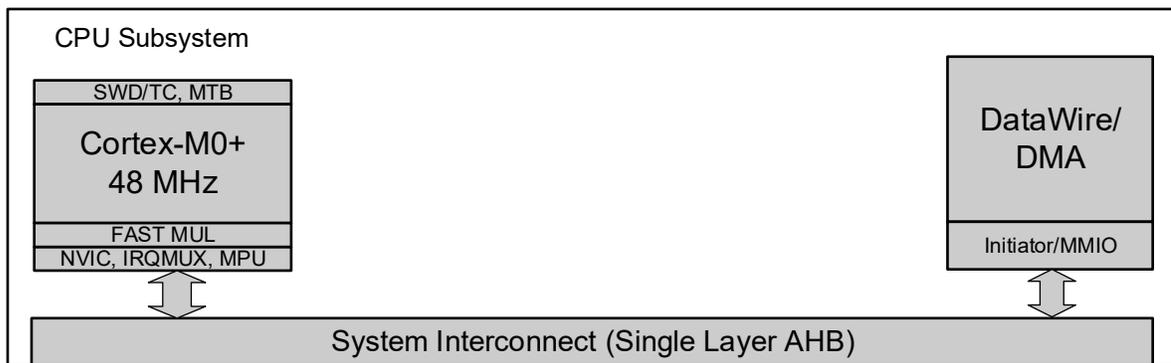


This section encompasses the following chapters:

- [Cortex-M0+ CPU chapter on page 31](#)
- [DMA Controller Modes chapter on page 37](#)
- [Interrupts chapter on page 52](#)
- [Device Security chapter on page 63](#)

## 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 two-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](http://www.arm.com).

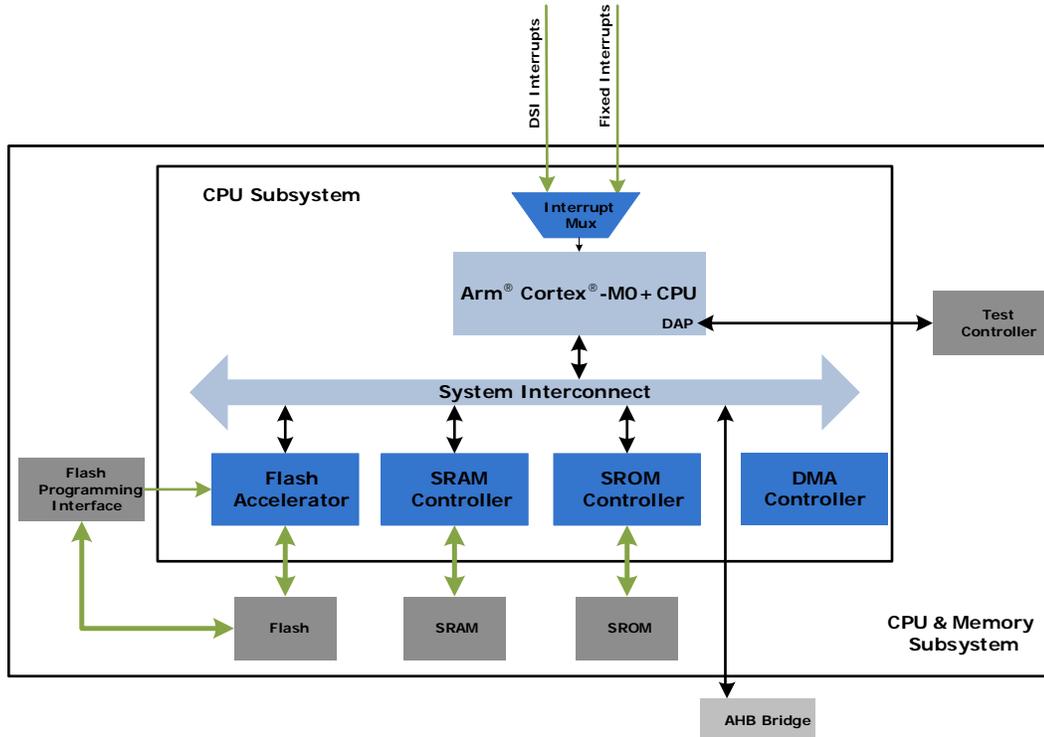
### 4.1 Features

The PSoC 4 Cortex-M0+ has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.95-1.36 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
- Implements design time configurable Memory Protection Unit (MPU)
- Supports unprivileged and privileged mode execution
- Supports optional Vector Table Offset Register (VTOR)
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

## 4.2 Block Diagram

Figure 4-1. CPU Subsystem Block Diagram



## 4.3 How It Works

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 34). 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 - 0x9FFFFFFF | External RAM    | Executable region for data.  |
| 0xA0000000 - 0xDFFFFFFF | External device | External device memory.  |
| 0xE0000000 - 0xE00FFFFF | PPB             | Peripheral registers within the CPU core.  |
| 0xE0100000 - 0xFFFFFFFF | Device          | PSoC 4 implementation-specific.  |

## 4.5 Registers

The Cortex-M0+ has sixteen 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 <sup>a</sup> | Reset Value    | Description  |
|-----------|-------------------|----------------|--|
| R0-R12    | RW                | Undefined      | R0-R12 are 32-bit general-purpose registers for data operations.   |
| MSP (R13) | RW                | [0x00000000]   | The stack pointer (SP) is register R13. In thread mode, bit[1] of the CONTROL register indicates which stack pointer to use:<br>0 = Main stack pointer (MSP). This is the reset value.<br>1 = Process stack pointer (PSP).<br>On Reset, the processor loads the MSP with the value from 0x00000000 and the PSP state is indeterminate. |
| PSP (R13) |                   |                |  |
| LR (R14)  | RW                | Undefined      | The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.  |
| PC (R15)  | RW                | [0x00000004]   | The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.  |
| PSR       | RW                | Undefined      | The program status register (PSR) combines:<br>Application Program Status Register (APSR).<br>Execution Program Status Register (EPSR).<br>Interrupt Program Status Register (IPSR).   |
| APSR      | RW                | Undefined      | The APSR contains the current state of the condition flags from previous instruction executions.   |
| EPSR      | RO                | [0x00000004].0 | On reset, EPSR is loaded with the value bit[0] of the register [0x00000004].   |
| IPSR      | RO                | 0              | The IPSR contains the exception number of the current ISR.   |
| PRIMASK   | RW                | 0              | The PRIMASK register prevents activation of all exceptions with configurable priority.   |
| CONTROL   | RW                | 0              | The CONTROL register controls the stack used when the processor is in thread mode.   |

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

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

Table 4-3. Cortex-M0+ PSR Bit Assignments

| Bit     | PSR Register | Name | Usage  |
|---------|--------------|------|--|
| 31      | APSR         | N    | Negative flag  |
| 30      | APSR         | Z    | Zero flag  |
| 29      | APSR         | C    | Carry or borrow flag   |
| 28      | APSR         | V    | Overflow flag  |
| 27 – 25 | –            | –    | Reserved   |
| 24      | EPSR         | T    | Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.  |
| 23 – 6  | –            | –    | Reserved   |
| 5 – 0   | IPSR         | N/A  | Exception number of current ISR:<br>0 = thread mode<br>1 = reserved<br>2 = NMI<br>3 = HardFault<br>4 – 10 = reserved<br>11 = SVCcall<br>12, 13 = reserved<br>14 = PendSV<br>15 = SysTick<br>16 = IRQ0<br>...<br>35 = IRQ19 |

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the [Interrupts chapter on page 52](#) for a list of exceptions.

## 4.6 Operating Modes

The 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 action ensures that instructions after the ISB execute using the new stack pointer.

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

## 4.7 Instruction Set

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

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

Table 4-4. Thumb Instruction Set

| Mnemonic            | Brief Description                              | Mnemonic            | Brief Description                         |
|---------------------|--|---------------------|---|
| ADCS                | Add with carry                                 | MVNS                | Bit wise NOT                              |
| ADD{S} <sup>a</sup> | Add  | NOP                 | No operation                              |
| ADR                 | PC-relative address to register                | ORRS                | Logical OR                                |
| ANDS                | Bit wise AND                                   | POP                 | Pop registers from stack                  |
| ASRS                | Arithmetic shift right                         | PUSH                | Push registers onto stack                 |
| B{cc}               | Branch {conditionally}                         | REV                 | Byte-reverse word                         |
| BICS                | Bit clear                                      | REV16               | Byte-reverse packed half-words            |
| BKPT                | Breakpoint                                     | REVSH               | Byte-reverse signed half-word             |
| BL                  | Branch with link                               | RORS                | Rotate right                              |
| BLX                 | Branch indirect with link                      | RSBS                | Reverse subtract                          |
| BX                  | Branch indirect                                | SBCS                | Subtract with carry                       |
| CMN                 | Compare negative                               | SEV                 | Send event                                |
| CMP                 | Compare  | STM                 | Store multiple registers, increment after |
| CPSID               | Change processor state, disable interrupts     | STR                 | Store register as word                    |
| CPSIE               | Change processor state, enable interrupts      | STRB                | Store register as byte                    |
| DMB                 | Data memory barrier                            | STRH                | Store register as half-word               |
| DSB                 | Data synchronization barrier                   | SUB{S} <sup>a</sup> | Subtract                                  |
| EORS                | Exclusive OR                                   | SVC                 | Supervisor call                           |
| ISB                 | Instruction synchronization barrier            | SXTB                | Sign extend byte                          |
| LDM                 | Load multiple registers, increment after       | SXTH                | Sign extend half-word                     |
| LDR                 | Load register from PC-relative address         | TST                 | Logical AND-based test                    |
| LDRB                | Load register with word                        | UXTB                | Zero extend a byte                        |
| LDRH                | Load register with half-word                   | UXTH                | Zero extend a half-word                   |
| LDRSB               | Load register with signed byte                 | WFE                 | Wait for event                            |
| LDRSH               | Load register with signed half-word            | WFI                 | Wait for interrupt                        |
| LSLs                | Logical shift left                             |                     |   |
| LSRS                | Logical shift right                            |                     |   |
| MOV{S} <sup>a</sup> | Move   |                     |   |
| MRS                 | Move to general register from special register |                     |   |
| MSR                 | Move to special register from general register |                     |   |
| MULS                | Multiply, 32-bit result                        |                     |   |

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 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 either the Cortex-M0+ internal clock or the low-frequency clock (LFCLK) as the source.

## 4.9 Debug

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

# 5. DMA Controller Modes



The DMA controller, in the PSoC 4100S Max device, provides DataWire (DW) and Direct Memory Access (DMA) functionality. The DMA controller has the following features:

- Supports 16 DMA channels
- Four levels of priority for each channel
- Byte, half-word (2 bytes), and word (4 bytes) transfers
- Three modes of operation supported for each channel
- Configurable interrupt generation
- Output trigger on completion of transfer
- Transfer sizes up to 65,536 data elements

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

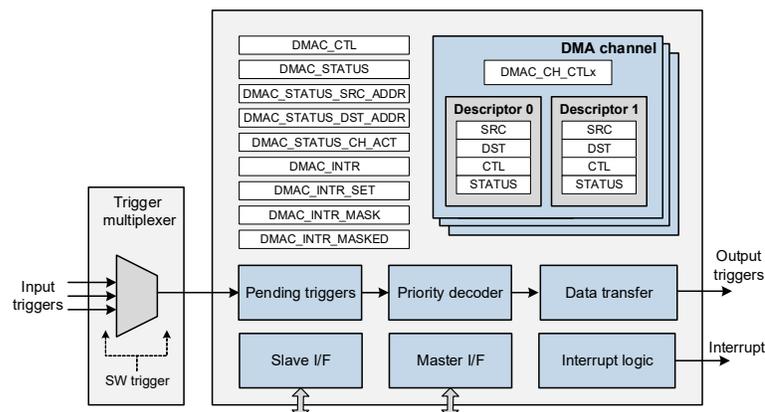
The DMA controller provides Active/Sleep functionality and is not available in the Deep-Sleep power mode.

## 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 provides an overview of the DMA controller at the 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](#).

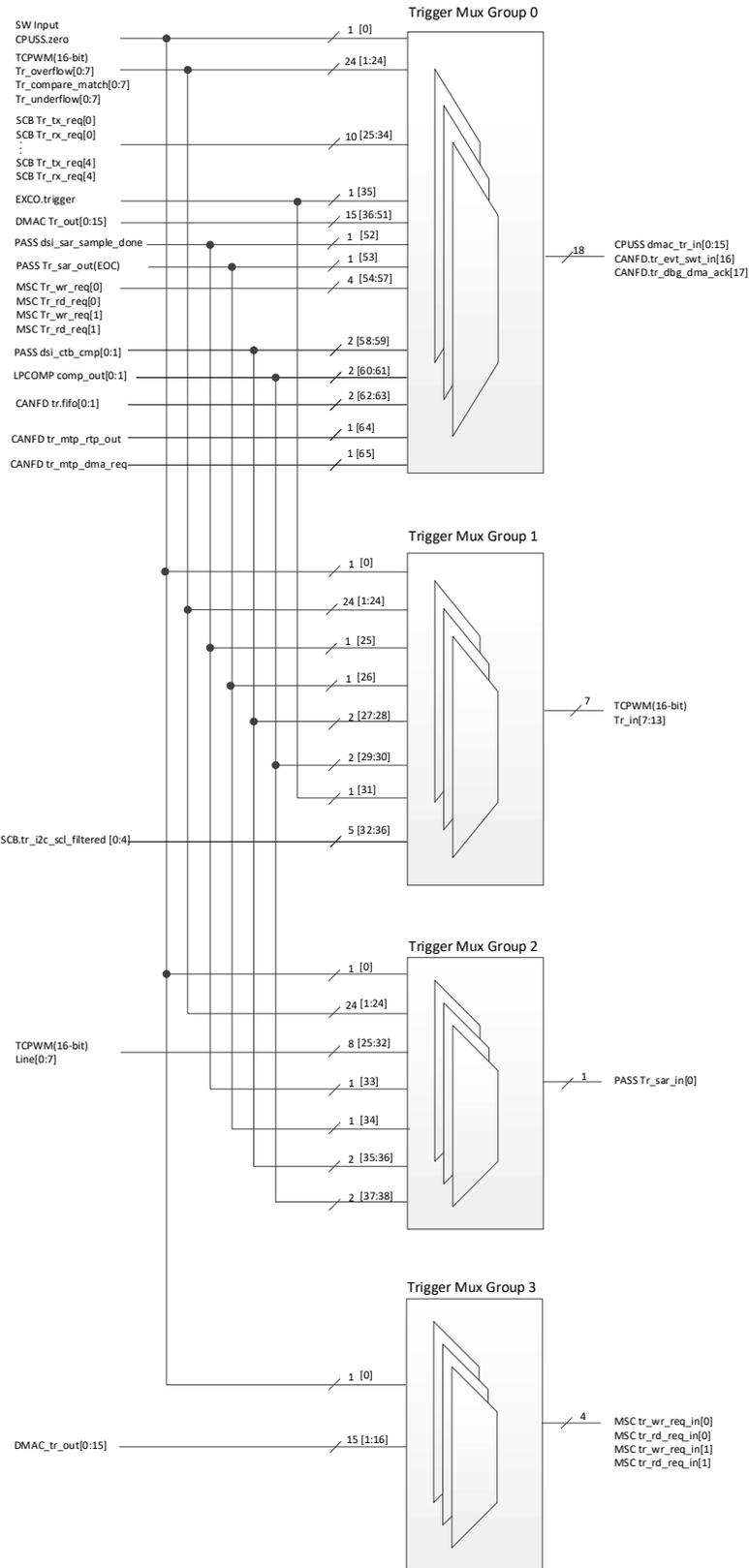
#### 5.1.1.1 Trigger Multiplexer

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

The PSoC 4100S Max device implements a single trigger group (Trigger group 0), which provides trigger inputs to the DMA. The PSoC 4100S Max device trigger input options can come from TCPWM, SCB, EXCO, DMA output triggers, SAR ADC, MSC CapSense, CAN FD, CTBm triggers and LPComp triggers. [Figure 5-2](#) shows the PSoC 4100S Max trigger multiplexer implementation.

Figure 5-2. PSoC 4100S Max Trigger Multiplexer Implementation



The trigger source for individual DMA channels is selected in the PERI\_TR\_GROUP0\_TR\_OUT\_CTLx[6:0] register. Table 5-1 provides the PSoC 4100S Max device trigger multiplexers.

Table 5-1. PSoC 4100S Max Device Trigger Sources

| PERI_TR_GROUP_TR_OUT_CTL x[6:0] | Trigger Source            |
|---------------------------------|---------------------------|
| 0                               | TCPWM 0 overflow          |
| 1                               | TCPWM 1 overflow          |
| 2                               | TCPWM 2 overflow          |
| 3                               | TCPWM 3 overflow          |
| 4                               | TCPWM 4 overflow          |
| 5                               | TCPWM 5 overflow          |
| 6                               | TCPWM 6 overflow          |
| 7                               | TCPWM 7 overflow          |
| 8                               | TCPWM 0 compare match     |
| 9                               | TCPWM 1 compare match     |
| 10                              | TCPWM 2 compare match     |
| 11                              | TCPWM 3 compare match     |
| 12                              | TCPWM 4 compare match     |
| 13                              | TCPWM 5 compare match     |
| 14                              | TCPWM 6 compare match     |
| 15                              | TCPWM 7 compare match     |
| 16                              | TCPWM 0 underflow         |
| 17                              | TCPWM 1 underflow         |
| 18                              | TCPWM 2 underflow         |
| 19                              | TCPWM 3 underflow         |
| 20                              | TCPWM 4 underflow         |
| 21                              | TCPWM 5 underflow         |
| 22                              | TCPWM 6 underflow         |
| 23                              | TCPWM 7 underflow         |
| 24                              | SCB 0 TX request          |
| 25                              | SCB 0 RX request          |
| 26                              | SCB 1 TX request          |
| 27                              | SCB 1 RX request          |
| 28                              | SCB 2 TX request          |
| 29                              | SCB 2 RX request          |
| 30                              | SCB 3 TX request          |
| 31                              | SCB 3 RX request          |
| 32                              | SCB 4 TX request          |
| 33                              | SCB 4 RX request          |
| 34                              | EXCO trigger              |
| 35                              | DMA Channel 0 trigger out |
| 36                              | DMA Channel 1 trigger out |
| 37                              | DMA Channel 2 trigger out |
| 38                              | DMA Channel 3 trigger out |
| 39                              | DMA Channel 4 trigger out |
| 40                              | DMA Channel 5 trigger out |
| 41                              | DMA Channel 6 trigger out |
| 42                              | DMA Channel 7 trigger out |
| 43                              | DMA Channel 8 trigger out |

| PERI_TR_GROUP_TR_OUT_CTL x[6:0] | Trigger Source              |
|---------------------------------|-----------------------------|
| 44                              | DMA Channel 9 trigger out   |
| 45                              | DMA Channel 10 trigger out  |
| 46                              | DMA Channel 11 trigger out  |
| 47                              | DMA Channel 12 trigger out  |
| 48                              | DMA Channel 13 trigger out  |
| 49                              | DMA Channel 14 trigger out  |
| 50                              | DMA Channel 15 trigger out  |
| 51                              | SAR ADC0 sample done        |
| 52                              | SAR ADC0 End of Conversion  |
| 53                              | MSC trigger write request   |
| 54                              | MSC trigger read request    |
| 55                              | MSC trigger write request   |
| 56                              | MSC trigger read request    |
| 57                              | CTBm [1] cmp0               |
| 58                              | CTBm [1] cmp1               |
| 59                              | LPCOMP[0] output            |
| 60                              | LPCOMP[1] output            |
| 61                              | CANFD receive FIFO0 trigger |
| 62                              | CANFD receive FIFO1 trigger |
| 63                              | CANFD trigger output        |

### 5.1.1.2 Creating Software Triggers

Every DMA channel has a trigger input and output trigger associated with it. This trigger input can come from any trigger group, as described in “[Trigger Multiplexer](#)” on page 38. 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 [6:0] is zero, the DMA trigger is configured for a software trigger. The DMA channel is then triggered using the PERI\_TR\_CTL register.

### 5.1.2 Pending Triggers

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

The pending triggers are registered in the status register (DMAC\_STATUS\_CH\_ACT).

### 5.1.3 Output Triggers

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

### 5.1.4 Channel Prioritization

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

### 5.1.5 Data Transfer Engine

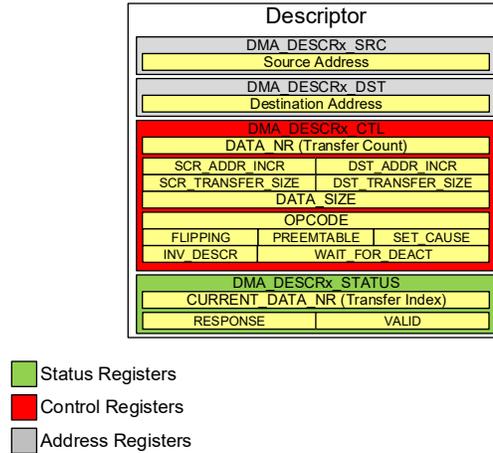
The data transfer engine is responsible for the data transfer from a source location to a destination location. When idle, the data transfer engine is ready to accept the highest priority activated channel. The configuration of the data transfer is specified by the descriptor. The data transfer engine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

- State 0 - Default State: This is the idle state of the DMA controller, where it waits for a trigger condition to initiate transfer.
- State 1 - Load Descriptor: When a trigger condition is encountered and priority is resolved, the data transfer engine enters the load descriptor state. In this state, the active descriptor (SRC, DST, and CTL) is loaded into the DMA controller to initiate the transfer. The DMAC\_STATUS, DMAC\_STATUS\_SRC\_ADDR and DMAC\_STATUS\_DST\_ADDR, and STATUS\_CH\_ACT will also reflect the currently active status.
- State 2 - Loading data from source: The data transfer engine uses the master I/F to load data from the source location.
- State 3 - Storing data at destination: The data transfer engine uses the master I/F to store data to the destination location. Depending on the Transfer mode, State 2 and 3 may be performed multiple times.
- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two cycles of the trigger activation. If it was set to 'wait indefinitely', the DMA controller will remain in this state until the trigger signal has gone low.
- State 6 - Storing Descriptor Response: In this phase, the data transfer according to the descriptor is completed and an interrupt may be generated if it was configured to do so. The Response field in DMAC\_DESCR\_PING\_STATUS or DMAC\_DESCR\_PONG\_STATUS is also populated and the state transitions to State 0.

## 5.2 Descriptors

The data transfer between a source and a destination in a channel is configured using a descriptor. Each channel in the DMA has two descriptors named PING and PONG descriptors (also called Descriptor 0 and Descriptor 1 in this document). A descriptor is a set of four 32-bit registers that contain the configuration for the transfer in the associated channel. Figure 5-3 shows the structure of a descriptor.

Figure 5-3. Descriptor Structure



### 5.2.1 Address Configuration

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

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

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

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

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

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

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

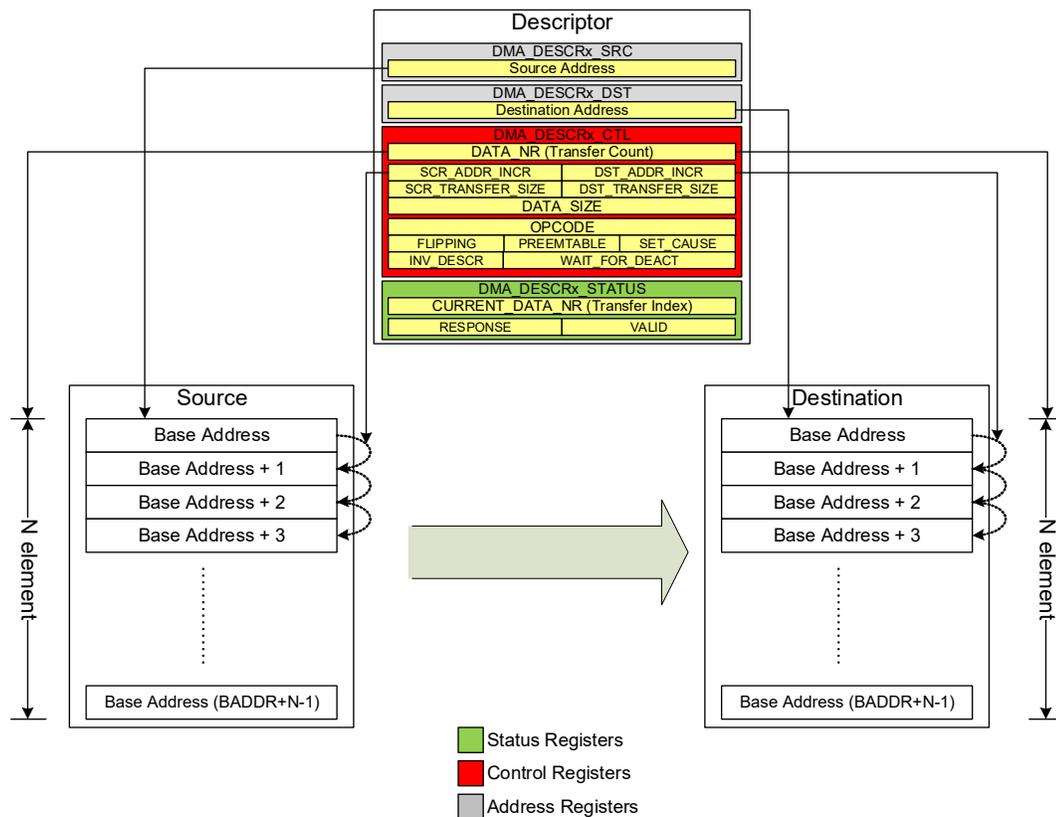
**Setting Interrupt Cause (SET\_CAUSE):** When the descriptor completes transferring all data elements, it generates an interrupt request. This interrupt request is shared among all DMA channels. Setting this bit enables the corresponding channel to be a source of this interrupt.

**Trigger Type (WAIT\_FOR\_DEACT):** When the DMA transfer based on the descriptor is completed, the data transfer engine checks the state of trigger deactivation. This is corresponding to State 5 of the data transfer engine. See [5.1.5 Data Transfer Engine on page 41](#). The type of DMA input trigger will determine when the trigger signal is considered deactivated. The DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger state is deactivated. This field is used to synchronize the controller's data transfer(s) with the agent that generated the trigger.

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

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

Figure 5-4. DMA Transfer: Address Configuration



### 5.2.2 Transfer Size

The transfer word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter (DATA\_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes, set by SCR\_TRANSFER\_SIZE and DST\_TRANSFER\_SIZE, can have a value of either the DATA\_SIZE or 32-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 5-2 summarizes the possible combinations of the transfer size settings and its description.

Table 5-2. Transfer Size Settings

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

### 5.2.3 Descriptor Chaining

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

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

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

## 5.2.4 Transfer Mode

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

### 5.2.4.1 Single Data Element Per Trigger (OPCODE 0)

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

Figure 5-5 shows a typical use case of this transfer. Here a UART receive (RX) register is the source and the destination is a peripheral register such as an SPI transmit (TX) register. The trigger is from the DMA request signal of the UART. When the trigger is received, the transfer engine will load data from the UART RX register and store the lower eight bits to the SPI TX register. Successive triggers will result in the same behavior because the descriptor will be rerun.

Note how the source and destination data widths are assigned as 32-bit. This is because all accesses to peripheral registers in PSoC must be 32-bit. Because the valid data width is only eight bits, the DATA\_SIZE is maintained as eight bit.

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

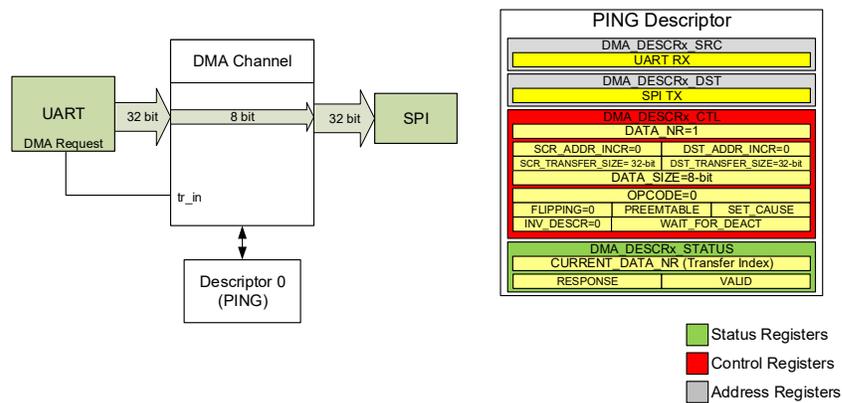
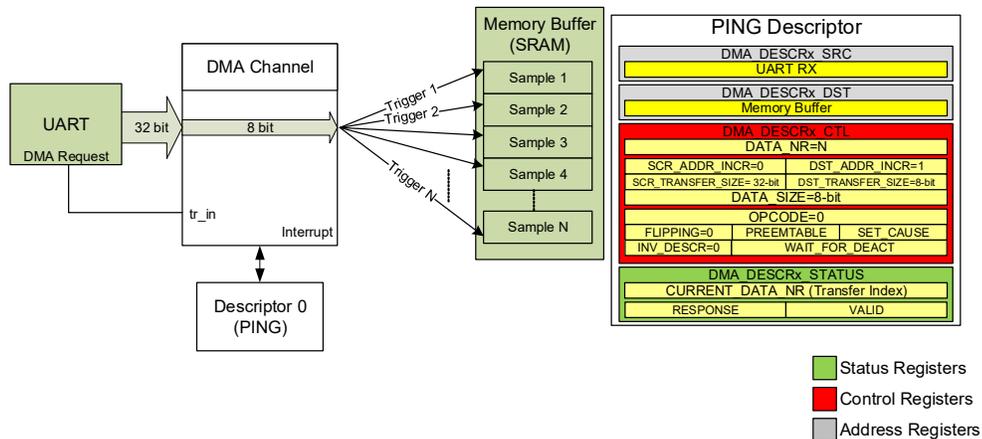


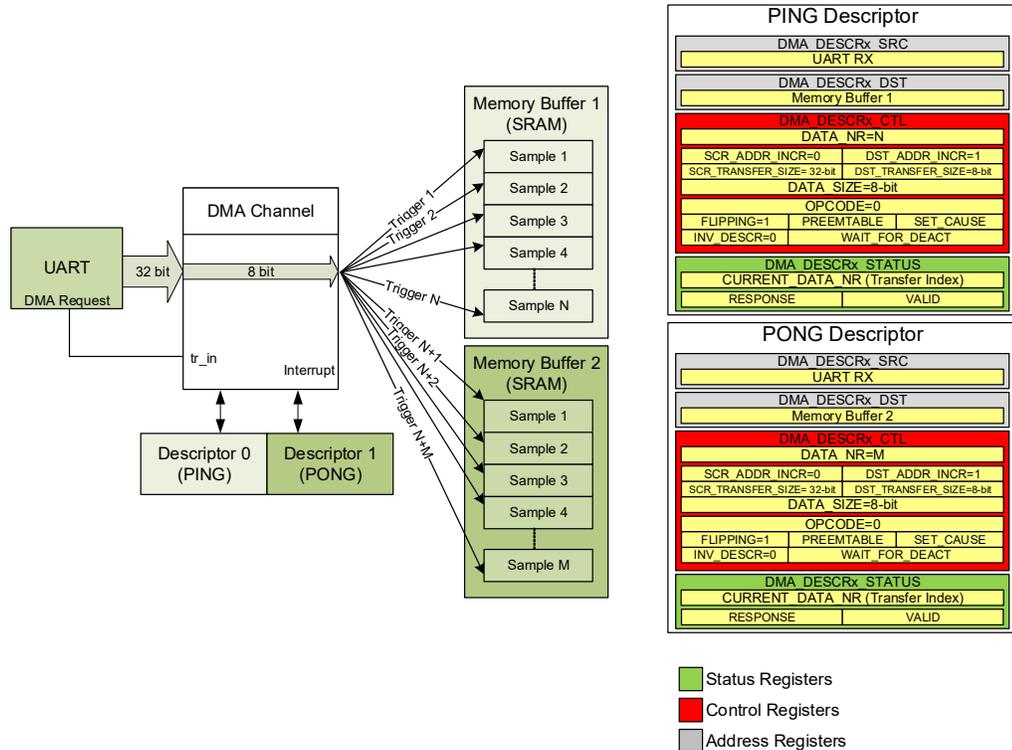
Figure 5-6 describes another use case where the data transfer is between the UART RX register and a buffer. The use case shows a PING descriptor, which is configured to increment the destination while taking data from a source location, which is a UART. When the trigger is received, the transfer engine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

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



A similar use case is shown in Figure 5-7. This demonstrates the use of the PING and PONG descriptors. On completion of the PING descriptor, the controller will flip to execute the PONG descriptor. Thus, two buffer transfers are achieved in sequence. However, note that the transfers are still done at one element transfer for every trigger.

Figure 5-7. DMA Transfer Using Flipping Feature

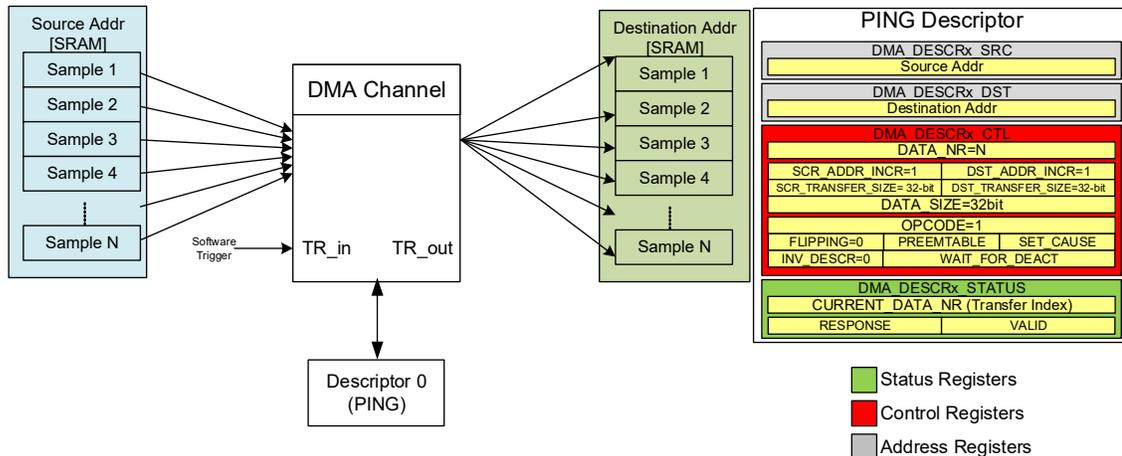


### 5.2.4.2 Entire Descriptor Per Trigger (OPCODE 1)

In this mode of operation, 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

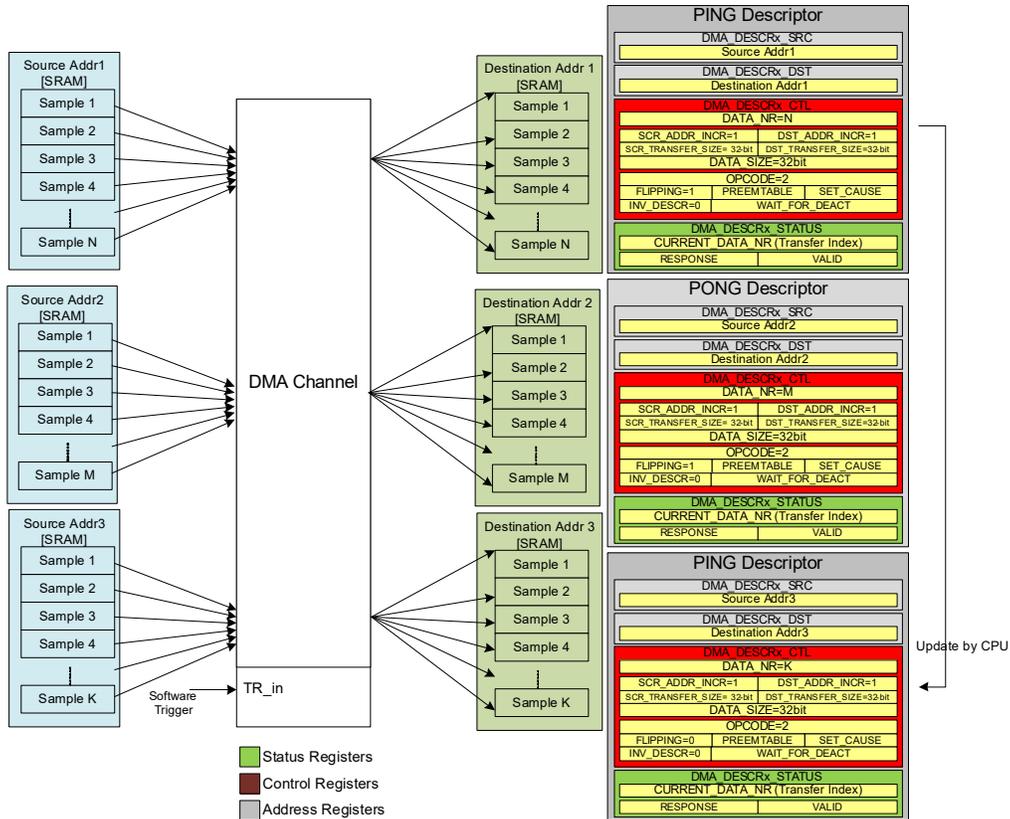


### 5.2.4.3 Entire Descriptor Chain Per Trigger (OPCODE 2)

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.

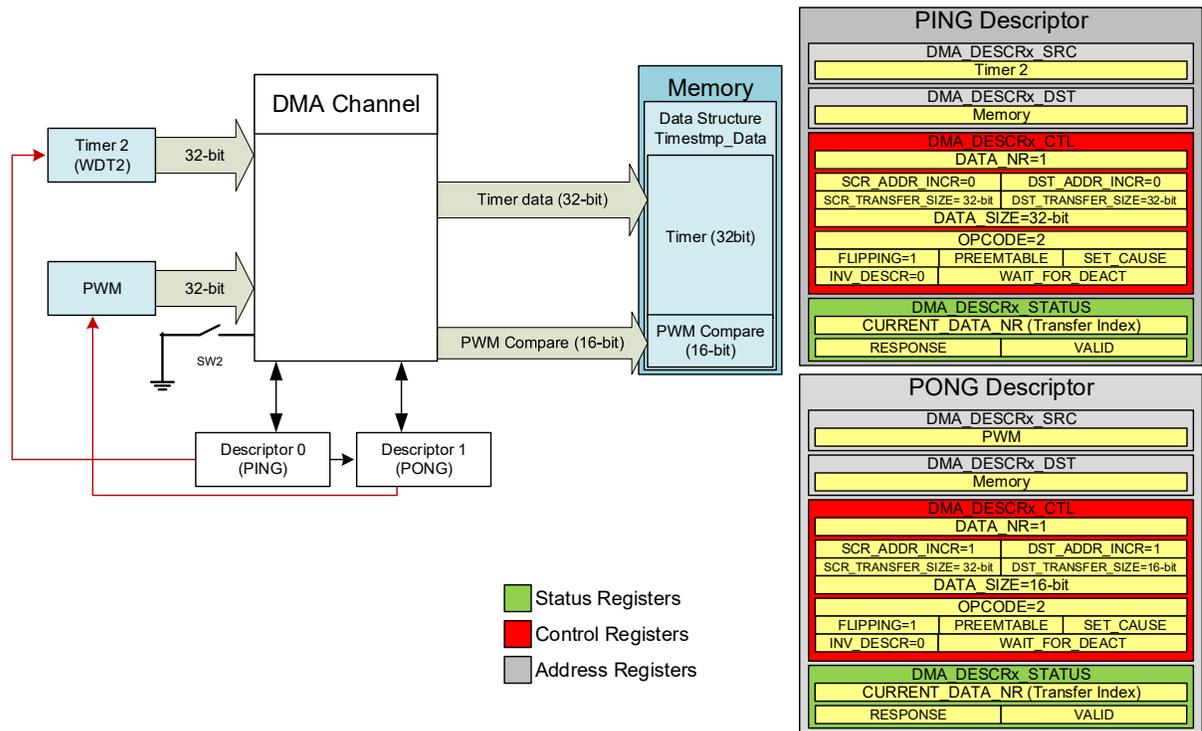
Figure 5-9 shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

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 a PWM compare register. Both the data is stored in consecutive locations in memory.

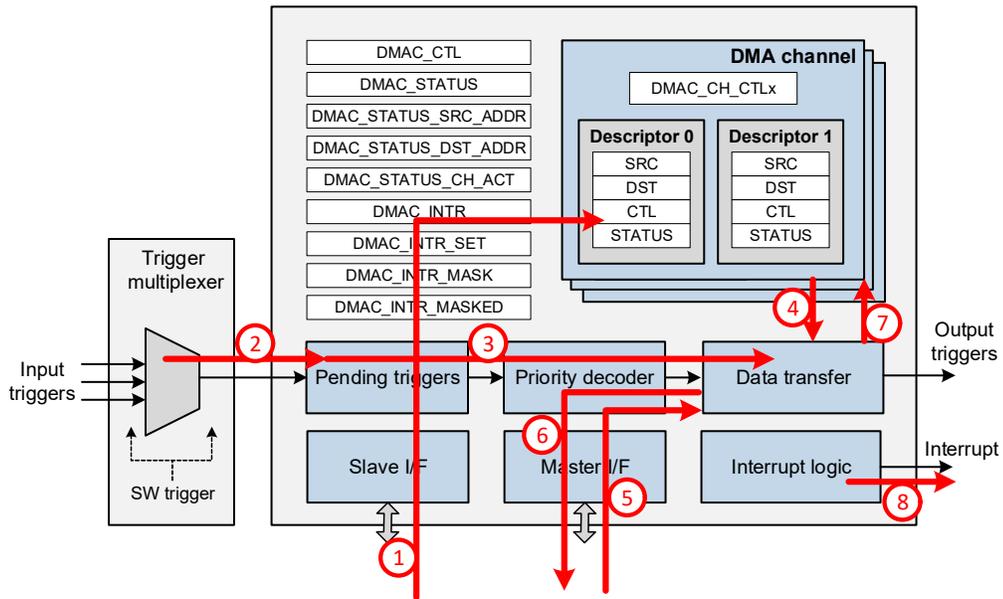
Figure 5-10. OPCODE 2: Multiple Sources to Memory



### 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:

$$\text{No of cycles} = 12 + \text{LOAD wait states} + \text{STORE wait states}$$

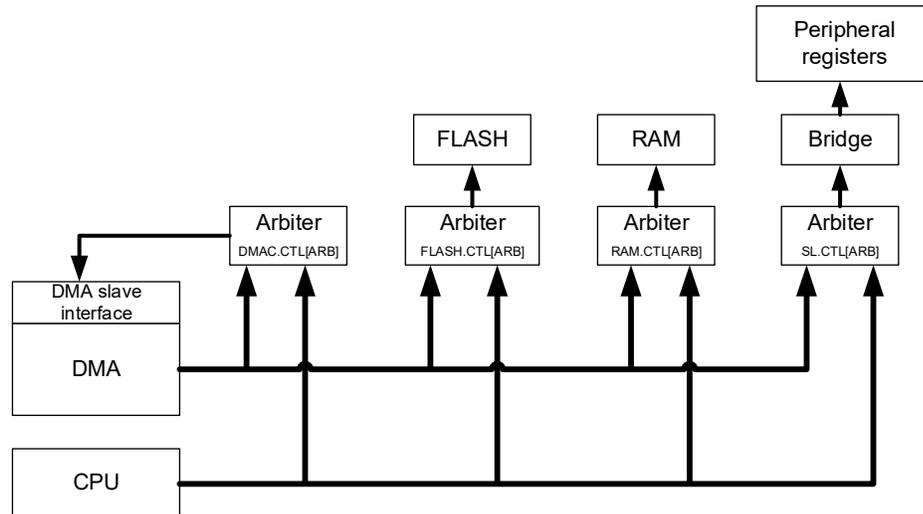
When transferring entire descriptors or chaining descriptor chains, 12 clock cycles are needed for the first data element. Subsequent elements need three cycles. This is also under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to transfer 'N' elements is:

$$\text{No of cycles} = (12 + \text{LOAD wait states} + \text{STORE wait states}) + (N-1) * (3 + \text{LOAD wait states} + \text{STORE wait states})$$

## 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

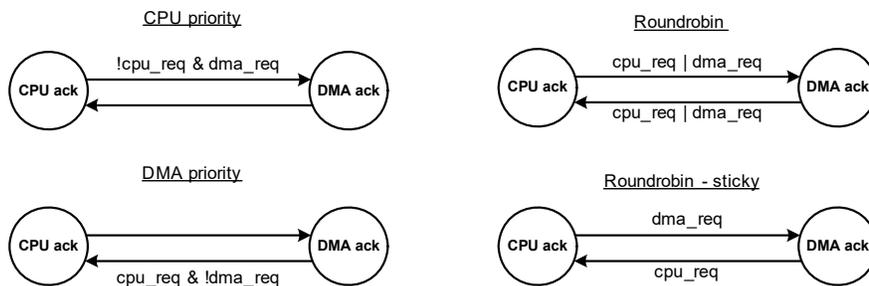


The arbitration policy for each slave can be one of the following:

- 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.

Figure 5-13 illustrates the arbitration models.

Figure 5-13. Arbitration Models



## 5.5 Register Lists

| 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

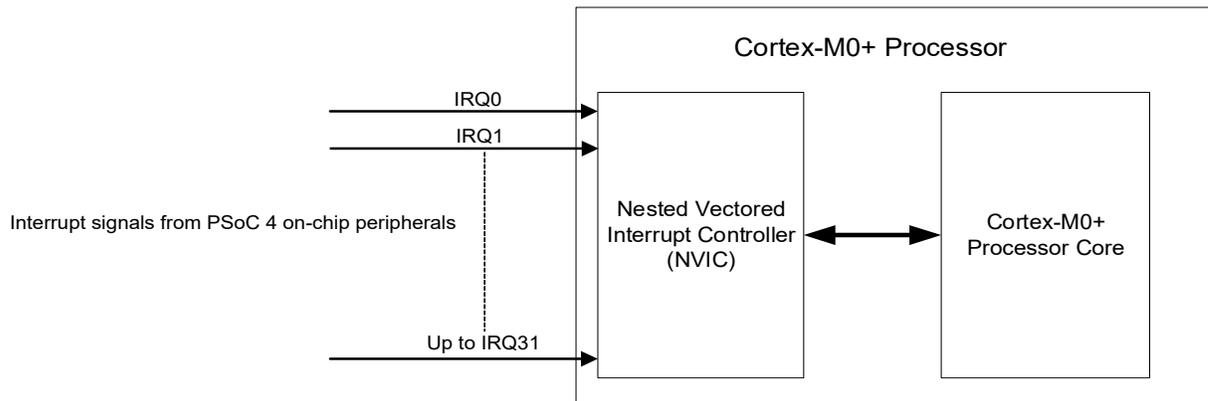


Figure 6-1 shows the interaction between interrupt signals and the Cortex-M0+ CPU. PSoC 4 has up to 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions 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.

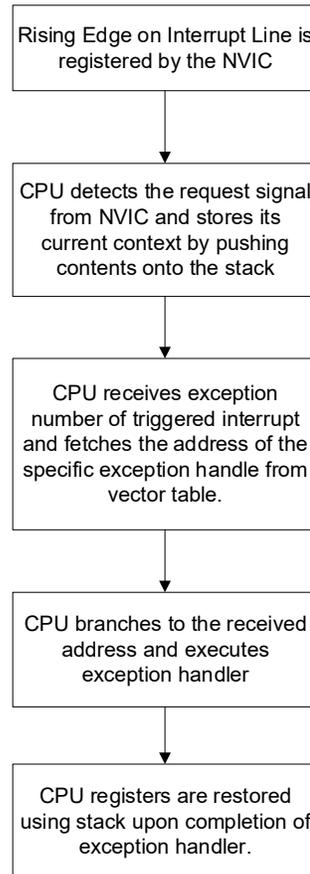
## 6.3 Interrupts and Exceptions - Operation

### 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

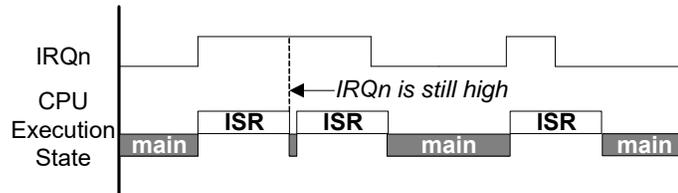


Figure 6-4. Pulse Interrupts

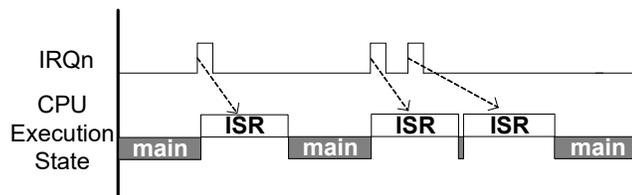


Figure 6-3 and Figure 6-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

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. The vector table can be located anywhere in the memory map (flash or SRAM) by modifying the Vector Table Offset Register (VTOR). This register is part of the System Control Space of CM0+ located at 0xE000ED08. This register takes bits 31:8 of the vector table address; bits 7:0 are reserved. Therefore, the vector table address should be 256 bytes aligned. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

Reads of flash addresses 0x00000000 and 0x00000004 are redirected to the first eight bytes of SROM to fetch the stack pointer and reset vectors, unless the DIS\_RESET\_VECT\_REL bit of the CPUSS\_SYSREQ register is set. The default value of this bit at reset is 0 ensuring that reset vector is always fetched from SROM. To allow flash read from addresses 0x00000000 and 0x00000004, the DIS\_RESET\_VECT\_REL bit should be set to '1'. The stack pointer vector holds the address that the stack pointer is loaded with on reset. The reset vector holds the address of the boot sequence. This mapping is done to use the default addresses for the stack pointer and reset vector from SROM when the device reset is released. For reset, 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 SYSCALL\_REQ bit of the CPUSS\_SYSREQ register is set, reads of flash address 0x00000008 are redirected to SROM to fetch the NMI vector address instead of from flash. Reset CPUSS\_SYSREQ to read the flash at address 0x00000008.

The exception sources (exception numbers 1 to 15) are explained in [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 (SRAM). The boot code and other data in SRAM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SRAM 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 two ways to trigger an NMI exception in the device:

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** An NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0P\_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL\_REQ bit in the CPUSS\_SYSREQ register. An NMI exception triggered by SYSCALL\_REQ bit always executes the NMI exception handler code that resides in SRAM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SRAM 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 (SVC) Exception

Supervisor Call (SVC) 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 SVC exception (see the [Chip Operational Modes chapter on page 103](#) for details on privileged mode.) There is no other privileged mode support for SVC at the architecture level in the device. The application developer must define the SVC exception handler according to the end application requirements.

The priority of a SVC 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 SVC 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 SVC 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, CM0P\_ICSR. On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSVCLR bit in the Interrupt Control State Register, CM0P\_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI\_14[23:22] of the System Handler Priority Register 3 (CM0P\_SHPR3). See the [Armv6-M Architecture Reference Manual](#) for more details.

### 6.4.6 SysTick Exception

CM0+ CPU in PSoC 4 supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various timekeeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0P\_SYST\_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two bit fields PRI\_15[31:30] of the System Handler Priority Register 3 (SHPR3). The SysTick exception can always be generated in software at any instant by writing a one to the PENDSTSETb bit in the Interrupt Control State Register, CM0P\_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a one to the PENDSTCLR bit in the Interrupt Control State Register, CM0P\_ICSR.

## 6.5 Interrupt Sources

PSoC 4 supports up to 32 interrupts (IRQ0 to IRQ31 or exception numbers 16 – 47) from peripherals. The source of each interrupt is listed in [Table 6-2](#). PSoC 4 provides flexible sourcing options for each interrupt line. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM and serial communication block. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. Interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously. See the [I/O System chapter on page 66](#) for details on GPIO interrupts.

Table 6-2. PSoC 4100S Max Interrupt Sources

| Interrupt | Cortex-M0+ Exception No. | Interrupt Source                              |
|-----------|--------------------------|---|
| NMI       | 2                        | SYSCALL_REQ                                   |
| IRQ0      | 16                       | GPIO Interrupt - Port 0                       |
| IRQ1      | 17                       | GPIO Interrupt - Port 1                       |
| IRQ2      | 18                       | GPIO Interrupt - Port 2                       |
| IRQ3      | 19                       | GPIO Interrupt - Port 3                       |
| IRQ4      | 20                       | GPIO Interrupt - All Port                     |
| IRQ5      | 21                       | LPCOMP (low- power comparator)                |
| IRQ6      | 22                       | WDT (Watch dog timer)                         |
| IRQ7      | 23                       | SCB0 (Serial Communication Block 0)           |
| IRQ8      | 24                       | SCB1 (Serial Communication Block 1)           |
| IRQ9      | 25                       | SCB2 (Serial Communication Block 2)           |
| IRQ10     | 26                       | SCB3 (Serial Communication Block 3)           |
| IRQ11     | 27                       | SCB4 (Serial Communication Block 4)           |
| IRQ12     | 28                       | CTBm (Continuous Time Block mini) – all CTBms |
| IRQ13     | 29                       | WCO WDT Interrupt                             |
| IRQ14     | 30                       | DMA Interrupt                                 |
| IRQ15     | 31                       | SPCIF Interrupt                               |
| IRQ16     | 32                       | MSC (CAPSENSE) #0                             |
| IRQ17     | 33                       | TCPWM0 (Timer/Counter/PWM0)                   |

Table 6-2. PSoC 4100S Max Interrupt Sources (continued)

| Interrupt | Cortex-M0+ Exception No. | Interrupt Source            |
|-----------|--------------------------|-----------------------------|
| IRQ18     | 34                       | TCPWM1 (Timer/Counter/PWM1) |
| IRQ19     | 35                       | TCPWM2 (Timer/Counter/PWM2) |
| IRQ20     | 36                       | TCPWM3 (Timer/Counter/PWM3) |
| IRQ21     | 37                       | TCPWM4 (Timer/Counter/PWM4) |
| IRQ22     | 38                       | TCPWM5 (Timer/Counter/PWM5) |
| IRQ23     | 39                       | TCPWM6 (Timer/Counter/PWM6) |
| IRQ24     | 40                       | TCPWM7 (Timer/Counter/PWM7) |
| IRQ25     | 41                       | SAR ADC                     |
| IRQ26     | 42                       | CAN FD Interrupt #0         |
| IRQ27     | 43                       | CAN FD Interrupt #1         |
| IRQ28     | 44                       | CRYPTO Interrupt            |
| IRQ29     | 45                       | MSC (CAPSENSE) #1           |
| IRQ30     | 46                       | EXCO Interrupt              |
| IRQ31     | 47                       | I2S Interrupt               |

## 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 56](#).

The priority of the 32 interrupts (IRQ0 to IRQ31) can be configured by writing to the Interrupt Priority registers (CM0P\_IPR). This is a group of 32-bit registers with each register storing the priority values of four interrupts, as given in [Table 6-3](#). The other bit fields in the register are not used.

Table 6-3. Interrupt Priority Register Bit Definitions

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

## 6.7 Enabling and Disabling Interrupts

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 (CM0P\_ISER) and the Interrupt Clear-Enable Register (CM0P\_ICER) are used to enable and disable the interrupts respectively. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 6-4 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 6-4. Interrupt Enable/Disable Registers

| Register                                    | Operation | Bit Value | Comment                  |
|---|-----------|-----------|--------------------------|
| Interrupt Set Enable Register (CM0P_ISER)   | Write     | 1         | To enable the interrupt  |
|   |           | 0         | No effect                |
|   | Read      | 1         | Interrupt is enabled     |
|   |           | 0         | Interrupt is disabled    |
| Interrupt Clear Enable Register (CM0P_ICER) | Write     | 1         | To disable the interrupt |
|   |           | 0         | No effect                |
|   | Read      | 1         | Interrupt is enabled     |
|   |           | 0         | Interrupt is disabled    |

The CM0P\_ISER and CM0P\_ICER registers are applicable only for interrupts IRQ0 to IRQ31. These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in “Exception Sources” on page 56.

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 (CM0P\_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0P\_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 CM0P\_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISR\_PENDING bit (bit 22) in the CM0P\_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 (CM0P\_ISPR) and the Interrupt Clear-Pending register (CM0P\_ICPR) are used to set and clear the pending status of the interrupt lines. These are 32-bit wide registers and each bit corresponds to the same numbered interrupt.

Table 6-6 shows the register access properties for these two registers. Note that writing 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 (CM0P_ISPR)   | Write     | 1         | To put an interrupt to pending state |
|  |           | 0         | No effect                            |
|  | Read      | 1         | Interrupt is pending                 |
|  |           | 0         | Interrupt is not pending             |
| Interrupt Clear-Pending Register (CM0P_ICPR) | Write     | 1         | To clear a pending interrupt         |
|  |           | 0         | No effect                            |
|  | Read      | 1         | Interrupt is pending                 |
|  |           | 0         | Interrupt is not pending             |

Setting the pending bit when the same bit is already set results in only one execution of the ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0P\_ISER register.

Note that the CM0P\_ISPR and CM0P\_ICPR registers are used only for the 32 peripheral interrupts (exception numbers 16–47). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in “[Exception Sources](#)” on page 56.

## 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 31](#) 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 and Deep-Sleep modes. The sequence of entering the different low-power modes is detailed in the [Power Modes chapter on page 104](#). Chip low-power modes have two categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available only in the Active and Deep-Sleep modes (watchdog timer interrupt,)
- 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 bits 31:28 of the VTOR register with the value of the flash or SRAM address at which the vector table will reside. This register write is done as part of device initialization code.

It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents. PSoC 4 Peripheral Device Library (PDL) uses the vector table in SRAM by default.

2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
  - a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
  - b. Define the exception handler function and write the address of the function to the exception vector table. [Table 6-1](#) gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
  - c. Set up the exception priority, as explained in [“Exception Priority” on page 58](#).
  - d. Enable the exception, as explained in [“Enabling and Disabling Interrupts” on page 59](#).

## 6.12 Registers

Table 6-7. List of Registers

| Register Name | Description                                      |
|---------------|--|
| CMOP_ISER     | Interrupt Set-Enable Register                    |
| CMOP_ICER     | Interrupt Clear Enable Register                  |
| CMOP_ISPR     | Interrupt Set-Pending Register                   |
| CMOP_ICPR     | Interrupt Clear-Pending Register                 |
| CMOP_IPR      | Interrupt Priority Registers                     |
| CMOP_ICSR     | Interrupt Control State Register                 |
| CMOP_AIRCR    | Application Interrupt and Reset Control Register |
| CMOP_SCR      | System Control Register                          |
| CMOP_CCR      | Configuration and Control Register               |
| CMOP_SHPR2    | System Handler Priority Register 2               |
| CMOP_SHPR3    | System Handler Priority Register 3               |
| CMOP_SHCSR    | System Handler Control and State Register        |
| CMOP_SYST_CSR | Systick Control and Status Register              |
| CPUSS_CONFIG  | CPU Subsystem Configuration Register             |
| CPUSS_SYSREQ  | System Request Register                          |

## 6.13 Associated Documents

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

# 7. Device Security



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

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

## 7.1 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.

## 7.2 How It Works

### 7.2.1 Device Security

The CPU operates in normal user mode or in privileged mode, and the device operates in one of four protection modes: BOOT, OPEN, PROTECTED, 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. In protected mode, only few registers are accessible; debug access to registers to reprogram flash is not available. The mode can be set back to OPEN but only after completely erasing the flash.
- **KILL mode:** 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.

## 7.2.2 Flash Security

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

Flash memory is organized in rows. You can assign one of two protection levels to each row; see [Table 7-1](#). Flash protection levels can only be changed by performing a complete flash erase.

For more details, see the [Nonvolatile Memory Programming chapter on page 374](#).

Table 7-1. Flash Protection Levels

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

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

# Section C: System Resources Subsystem (SRSS)

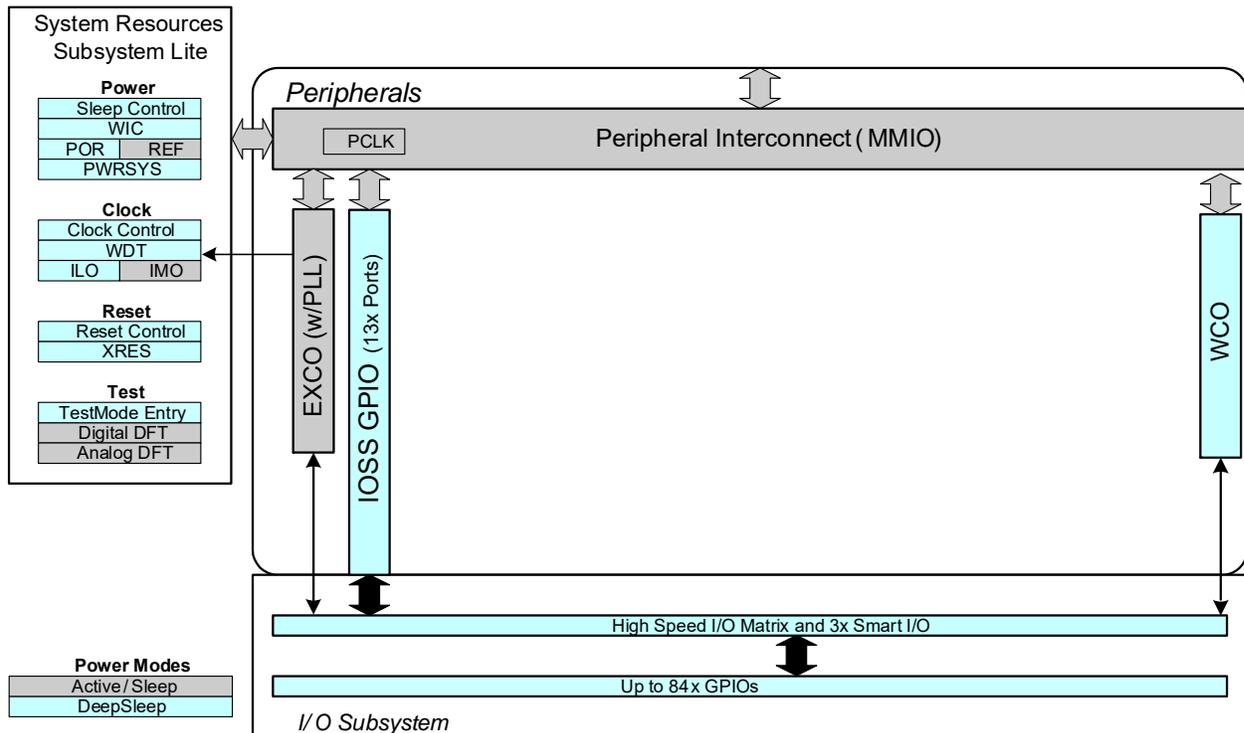


This section encompasses the following chapters:

- I/O System chapter on page 66
- Clocking System chapter on page 88
- Power Supply and Monitoring chapter on page 99
- Chip Operational Modes chapter on page 103
- Power Modes chapter on page 104
- Watchdog Timer chapter on page 108
- Trigger Multiplexer Block chapter on page 113
- Reset System chapter on page 117

## Top Level Architecture

System Resource Subsystem Block Diagram



# 8. I/O System



This chapter explains the PSoC 4 MCU I/O system, its features, architecture, operating modes, and interrupts. The I/O system provides the interface between the CPU core and peripheral components to the outside world. The flexibility of PSoC 4 MCUs and the capability of its I/O to route most signals to most pins greatly simplifies circuit design and board layout. The GPIO pins in the PSoC 4 MCU family are grouped into ports; a port can have a maximum of eight GPIO pins. The PSoC 4100S Max device has a maximum of 84 GPIOs arranged in 13 ports.

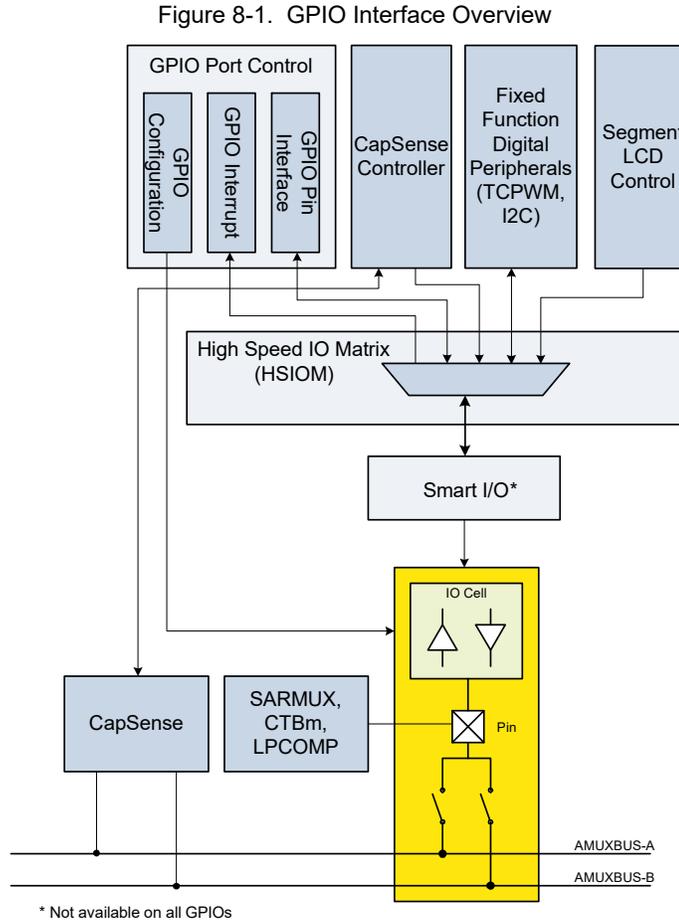
## 8.1 Features

The PSoC 4 GPIOs have these features:

- Analog and digital input and output capabilities
- Eight drive strength modes
- Separate port read and write registers
- Separate I/O supplies and voltages for groups of I/O ports
- 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 LVTTTL input buffer mode
- CapSense support
- Segment LCD drive support
- Three blocks of Smart I/O provides the ability to perform Boolean functions in the I/O signal path
- Two analog mux buses (AMUXBUS-A and AMUXBUS-B) that can be used to multiplex analog signals

## 8.2 GPIO Interface Overview

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

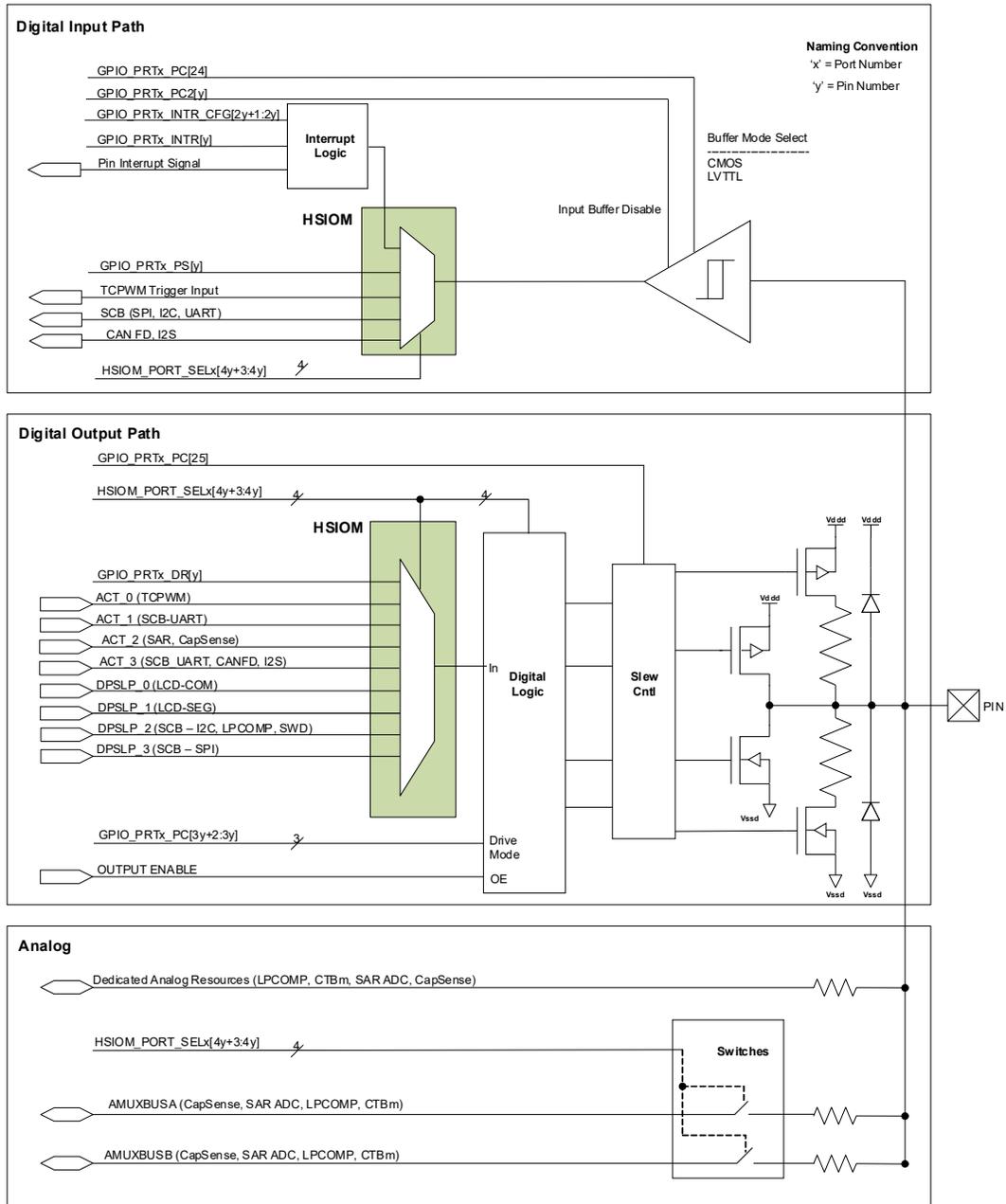


GPIO pins are connected to I/O cells. These cells are equipped with an input buffer for the digital input, providing high input impedance and a driver for the digital output signals. The digital peripherals connect to the I/O cells via the high-speed I/O matrix (HSIOM). HSIOM contains multiplexers to connect between a peripheral selected by the user and the pin. Some port pins have a Smart I/O block between the HSIOM and the pins. The Smart I/O block enables logical operations on the pin signal. The analog peripherals such as SAR ADC, Continuous Time Block-mini (CTBm), Low Power Comparator (LPCOMP) and analog mux bus connections are done in the GPIO cell directly. The CapSense block is connected to the GPIO pins through the AMUX buses.

### 8.3 I/O Cell Architecture

Figure 8-2 shows the I/O cell architecture present in every GPIO pin. It comprises an input buffer and an output driver that connect to the HSIOM multiplexers for digital input and output signals. Analog peripherals connect directly to the pin for point to point connections or use the AMUXBUS.

Figure 8-2. GPIO Block Diagram



### 8.3.1 Digital Input Buffer

The digital input buffer provides a high-impedance buffer for the external digital input. The buffer is enabled and disabled by the INP\_DIS bit of the Port Configuration Register 2 (GPIO\_PRTx\_PC2, where x is the port number). The input buffer is connected to the HSIOM for routing to the CPU port registers and selected peripherals. Writing to the HSIOM port select register (HSIOM\_PORT\_SELx) selects the pin connection. See the [device datasheet](#) for the specific connections available for each pin. If a pin is connected only to an analog signal, the input buffer should be disabled to avoid crowbar currents. The buffer is configurable for the following modes:

- CMOS
- LVTTTL

These buffer modes are selected by the PORT\_VTRIP\_SEL bit (GPIO\_PRTx\_PC[24]) of the Port Configuration register.

Table 8-1. Input Buffer Modes

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

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 connected to the HSIOM, shown in [Figure 8-2](#), are pin dependent. See the [device datasheet](#) to know the functions available for each pin.

### 8.3.2 Digital Output Driver

Pins are driven by the digital output driver. It consists of circuitry to implement different drive modes and slew rate control for the digital output signals. The HSIOM selects the control source for the output driver. The three primary types of control sources are CPU registers, configurable digital peripherals instantiated in the programmable UDB/DSI fabric, and fixed-function digital peripherals. A particular HSIOM connection is selected by writing to the HSIOM port select register (HSIOM\_PORT\_SELx). I/O ports are powered by different sources. The specific allocation of ports to supply sources can be found in the Pinout section of the [device datasheet](#).

In PSoC 4, I/Os are driven with  $V_{DD}$  supply. Each GPIO pin has ESD diodes to clamp the pin voltage to the  $V_{DD}$  source. Ensure that the voltage at the pin does not exceed the I/O supply voltage  $V_{DD}$  and drop below  $V_{SSD}$ . 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 [8.4 High-Speed I/O Matrix](#) to know about the peripheral source selection for the data and to enable or disable control source selection.

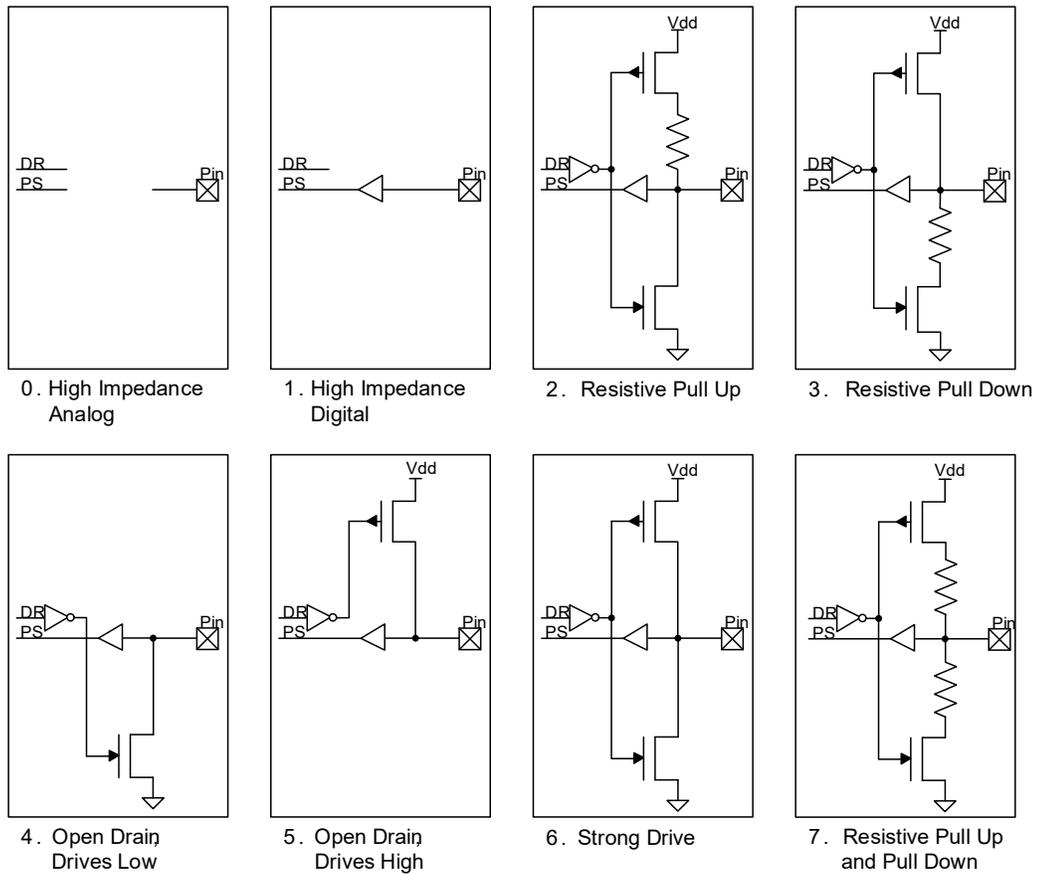
#### 8.3.2.1 Drive Modes

Each I/O is individually configurable into one of eight drive modes using the Port Configuration register, GPIO\_PRTx\_PC. [Table 8-2](#) lists the drive modes. [Figure 8-3](#) is a simplified output driver diagram that shows the pin view based on each of the eight drive modes.

Table 8-2. Drive Mode Settings

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

Figure 8-3. I/O Drive Mode Block Diagram



- High-Impedance Analog

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

- High-Impedance Digital

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

- Resistive Pull-Up or Resistive Pull-Down

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

- Open Drain Drives High and Open Drain Drives Low

Open drain modes provide high impedance in one of the data states and strong drive in the other. The pins can be used as digital input or output in these modes. Therefore, these modes are widely used in bi-directional digital communication. Open drain drive high mode is used when 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<sup>2</sup>C 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 devices.

- Resistive Pull-Up and Resistive Pull-Down

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

### 8.3.2.2 Slew Rate Control

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

## 8.4 High-Speed I/O Matrix

The high-speed I/O matrix (HSIOM) is a set of high-speed multiplexers that route internal CPU and peripheral signals to and from GPIOs. HSIOM allows GPIOs to be shared with multiple functions and multiplexes the pin connection to a user-selected peripheral. In PSoC 4100S Max, Port 1, Port 2 and Port 3 support the Smart I/O functionality. Other ports connect directly to the HSIOM. The HSIOM\_PORT\_SELx register is provided to select the peripheral. It is a 32-bit wide register available for each port, with each pin occupying four bits. This register provides up to 16 different options for a pin as listed in [Table 8-3](#).

Refer to the example of [Switching a GPIO Pin Connection between Analog and Digital Sources](#).

Table 8-3. PSoC 4100S Max HSIOM Port Settings

| HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number) |               |       |  |
|--|---------------|-------|--|
| Bits   | Name (SEL'y') | Value | Description (Selects pin 'y' source (0 ≤ y ≤ 7))                                 |
| 4y+3 : 4y  | GPIO          | 0     | Pin is regular firmware-controlled I/O or connected to dedicated hardware block. |
|  | GPIO_DSI      | 1     | The output is firmware-controlled, but OE is controlled from DSI.                |
|  | DSI_DSI       | 2     | Both output and OE are controlled from DSI.                                      |
|  | DSI_GPIO      | 3     | The output is controlled from DSI, but OE is firmware-controlled.                |
|  | CSD_SENSE     | 4     | Pin is a CSD sense pin (analog mode).  |
|  | CSD_SHIELD    | 5     | Pin is a CSD shield pin (analog mode).   |
|  | AMUXA         | 6     | Pin is connected to AMUXBUS-A.   |
|  | AMUXB         | 7     | Pin is connected to AMUXBUS-B.   |
|  | ACTIVE_0      | 8     | Pin-specific Active source #0 (TCPWM Output).                                    |
|  | ACTIVE_1      | 9     | Pin-specific Active source #1 (SCB-UART).  |
|  | ACTIVE_2      | 10    | Pin-specific Active source #2 (SAR ADC, CAPSENSE™).                              |
|  | ACTIVE_3      | 11    | Pin-specific Active source #3 (TCPWM Input, SCB-UART, CANFD, I2S).               |
|  | 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 <sup>2</sup> C, SWD, LPCOMP).           |
|  | DEEP_SLEEP_3  | 15    | Pin-specific Deep-Sleep source #3 (SCB-SPI).                                     |

**Note** The Active and Deep-Sleep sources are pin dependent. See the “Pinouts” section of the [device datasheet](#) for more details on the features supported by each pin. Refer to the example of [Switching a GPIO Pin Connection between Analog and Digital Sources](#).

## 8.5 Smart I/O

The Smart I/O™ block adds programmable logic to an I/O port. This programmable logic integrates board-level Boolean logic functionality such as AND, OR, and XOR into the port. The Smart I/O block has these features:

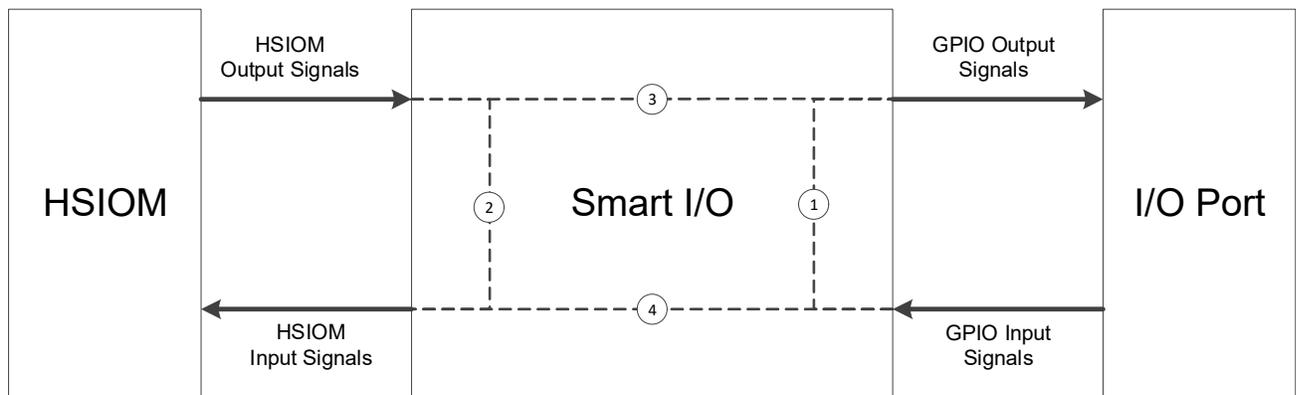
- Integrate board-level Boolean logic functionality into a port
- Ability to preprocess HSIOM input signals from the GPIO port pins
- Ability to post-process HSIOM output signals to the GPIO port pins
- Support in all device power modes
- Integrate close to the I/O pads, providing low signal delay

The PSoC 4100S Max device supports Smart I/O on three ports: Port 1, Port 2 and Port 3. Refer to the [device datasheet](#) for details on which ports provide Smart I/O and the mapping between the Port number and Smart I/O blocks number. For a general Smart I/O register description, the 'PRGIO\_PRTx' nomenclature will be used.

### 8.5.1 Overview

The Smart I/O block is positioned in the signal path between the HSIOM and the I/O port. The HSIOM multiplexes the output signals from fixed-function peripherals and CPU to a specific port pin and vice-versa. The Smart I/O block is placed on this signal path, acting as a bridge that can process signals from port pins and HSIOM, as shown in [Figure 8-4](#).

Figure 8-4. Smart I/O Interface



The signal paths supported through the Smart I/O block as shown in [Figure 8-4](#) are as follows:

1. Implement self-contained logic functions that directly operate on port I/O signals
2. Implement self-contained logic functions that operate on HSIOM
3. Operate on and modify HSIOM output signals and route the modified signals to port I/O signals
4. Operate on and modify port I/O signals and route the modified signals to HSIOM input signals

The following sections discuss the Smart I/O block components, routing, and configuration in detail. In these sections, GPIO signals (*io\_data\_in*) refer to the input/output signals from the I/O port; device or chip (*chip\_data*) signals refer to the input/output signals from HSIOM.

## 8.5.2 Block Components

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

- Clock/reset component
- Synchronizers
- Lookup Table (LUT3) components
- Data unit component

### 8.5.2.1 Clock and Reset

The clock and reset component selects the Smart I/O block's clock (`clk_block`) and reset signal (`rst_block_n`). A single clock and reset signal is used for all components in the block. The clock and reset sources are determined by the `CLOCK_SRC[4:0]` bit field of the `PRGIO_PRTx_CTL` register. The selected clock is used for the synchronous logic in the block components, which includes the I/O input synchronizers, LUT, and data unit components. The selected reset is used to asynchronously reset the synchronous logic in the LUT and data unit components.

Note that the selected clock (`clk_block`) for the block's synchronous logic is not phase-aligned with other synchronous logic in the device, operating on the same clock. Therefore, communication between Smart I/O and other synchronous logic should be treated as asynchronous.

The following clock sources are available for selection:

- GPIO input signals "`io_data_in[7:0]`". These clock sources have no associated reset.
- HSIOM output signals "`chip_data[7:0]`". These clock sources have no associated reset.
- The Smart I/O clock (`clk_prgio`) is derived from the system clock (`clk_sys`) using a peripheral clock divider. See the [Clocking System chapter on page 88](#) for details on peripheral clock dividers. This clock is only available in Active and Sleep power modes. The clock can have one out of two associated resets: `rst_sys_act_n` and `rst_sys_dpslp_n`. These resets determine in which system power modes the block synchronous state is reset; for example, `rst_sys_act_n` is intended for Smart I/O synchronous functionality in the Active power mode and reset is activated in the Deep-Sleep power mode.
- The low-frequency (40 kHz) system clock (LFCLK). This clock is available in Active, Sleep, and Deep-Sleep power mode. This clock has an associated reset, `rst_lf_dpslp_n`.

When the block is enabled, the selected clock (`clk_block`) and associated reset (`rst_block_n`) are routed to the block components. When the block is disabled, no clock is routed to the block components and the reset is activated (the LUT and data unit components are set to the reset value of '0').

The I/O input synchronizers introduce a delay of two `clk_block` cycles (when synchronizers are enabled). As a result, in the first two cycles, the block may be exposed to stale data from the synchronizer output. Hence, during the first two clock cycles, the reset is activated and the block is in bypass mode.

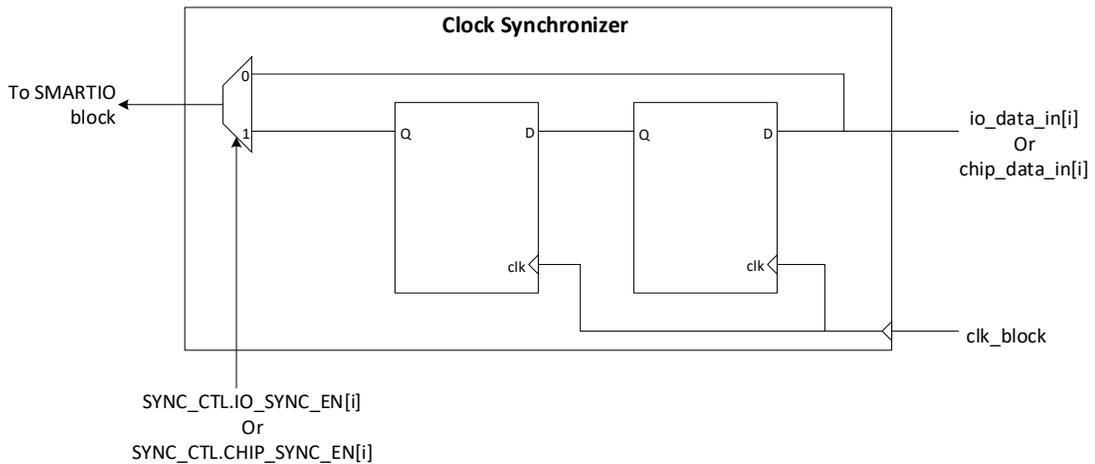
Table 8-4. Clock and Reset Register Control

| Register[BIT_POS]    | Bit Name     | Description   |
|----------------------|--------------|---|
| PRGIO_PRT0_CTL[12:8] | CLK_SRC[4:0] | <p>Clock (clk_block)/reset (rst_block_n) source selection:</p> <p>"0": io_data_in[0]/"1"</p> <p>...</p> <p>"7": io_data_in[7]/"1"</p> <p>"8": chip_data[0]/"1"</p> <p>...</p> <p>"15": chip_data[7]/"1"</p> <p>"16": clk_prgio/rst_sys_act_n; asserts reset in any power mode other than Active; that is, Smart I/O is active only in Active power mode with clock from the peripheral divider.</p> <p>"17": clk_prgio/rst_sys_dpslp_n. Smart I/O is active in all power modes with clock from the peripheral divider. However, the clock will not be active in Deep-Sleep power mode.</p> <p>"19": clk_lf/rst_lf_dpslp_n. Smart I/O is active in all power modes with clock from ILO.</p> <p>"20"- "30": Clock source is a constant '0'. Any of these clock sources should be selected when the IP is disabled to ensure low power consumption.</p> <p>"31": clk_sys/"1". This selection is NOT intended for "clk_sys" operation. However, for asynchronous operation, three "clk_sys" cycles after enabling the IP, the IP is fully functional (reset is deactivated). To be used for asynchronous (clockless) block functionality.</p> |

### 8.5.2.2 Synchronizer

Each GPIO input signal and device input signal (HSIOM input) can be used either asynchronously or synchronously. To use the signals synchronously, a double flip-flop synchronizer, as shown in [Figure 8-5](#), is placed on both these signal paths to synchronize the signal to the Smart I/O clock (clk\_block). The synchronization for each pin/input is enabled or disabled by setting or clearing the IO\_SYNC\_EN[i] bit field for GPIO input signal and CHIP\_SYNC\_EN[i] for HSIOM signal in the PRGIO\_PRT0\_SYNC\_CTL register, where 'i' is the pin number.

Figure 8-5. Smart I/O Clock Synchronizer



### 8.5.2.3 Lookup Table (LUT3)

Each Smart I/O block contains eight lookup table (LUT3) components. The LUT3 component consists of a three-input LUT and a flip-flop. Each LUT3 block takes three input signals and generates an output based on the configuration set in the PRGIO\_PRTx\_LUT\_CTLy register (y denotes the LUT3 number). For each LUT3, the configuration is determined by an 8-bit lookup vector LUT[7:0] and a 2-bit opcode OPC[1:0] in the PRGIO\_PRTx\_LUT\_CTLy register. The 8-bit vector is used as a lookup table for the three input signals. The 2-bit opcode determines the usage of the flip-flop. The LUT3 configuration for different opcode is shown in [Figure 8-6](#).

PRGIO\_PRTx\_LUT\_SELy registers select the three input signals (tr0\_in, tr1\_in and tr2\_in) going into each LUT3. The input can come from the following sources:

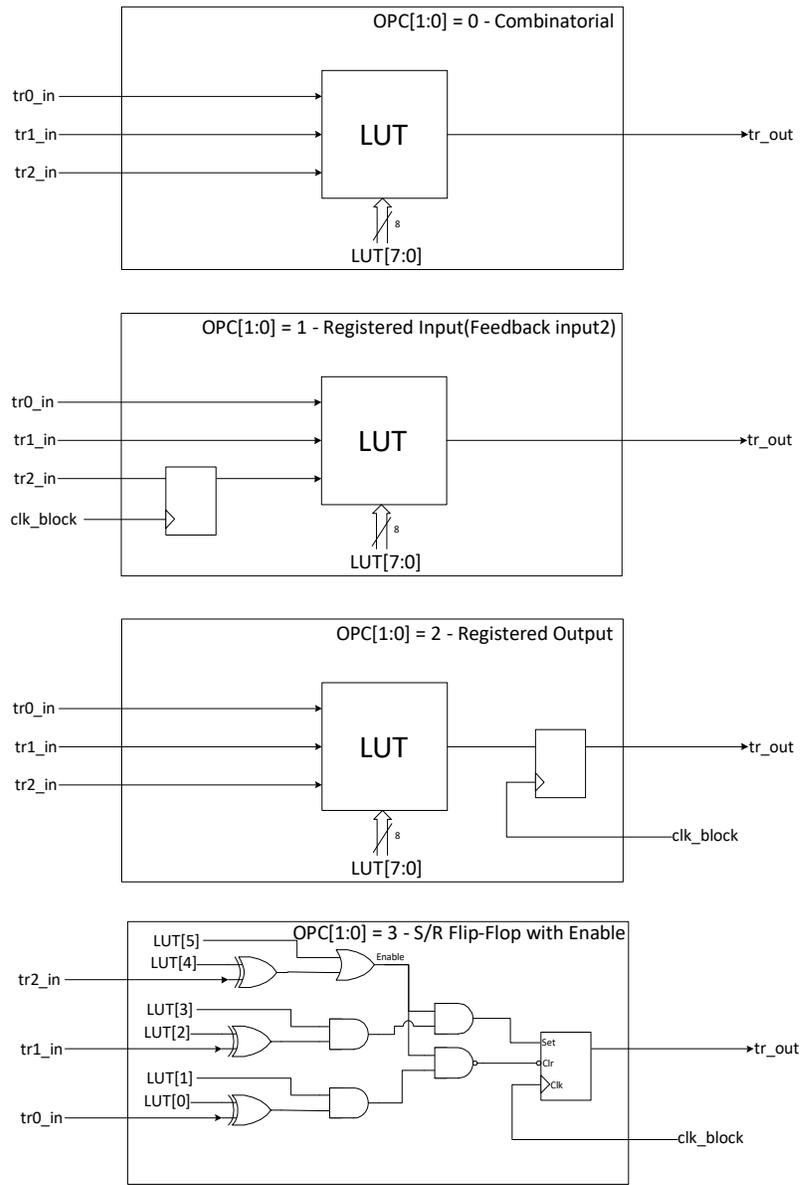
- Data unit output
- Other LUT3 output signals (tr\_out)
- HSIOM output signals (chip\_data[7:0])
- GPIO input signals (io\_data\_in[7:0])

LUT\_TR0\_SEL[3:0] bits of the PRGIO\_PRTx\_LUT\_SELy register selects the tr0\_in signal for the y<sup>th</sup> LUT3. Similarly, LUT\_TR1\_SEL[3:0] bits and LUT\_TR2\_SEL[3:0] bits select the tr1\_in and tr2\_in signals respectively. See [Table 8-5](#) for details.

Table 8-5. LUT3 Register Control

| Register[BIT_POS]          | Bit Name         | Description   |
|----------------------------|------------------|---|
| PRGIO_PRTx_LUT_CTLy[7:0]   | LUT[7:0]         | LUT configuration. Depending on the LUT opcode (LUT_OPC), the internal state, and the LUT input signals tr0_in, tr1_in, and tr2_in, the LUT configuration is used to determine the LUT output signal and the next sequential state.   |
| PRGIO_PRTx_LUT_CTLy[9:8]   | LUT_OPC[1:0]     | LUT opcode specifies the LUT operation as illustrated in <a href="#">Figure 8-6</a> .   |
| PRGIO_PRTx_LUT_SELy[3:0]   | LUT_TR0_SEL[3:0] | LUT input signal "tr0_in" source selection:<br>"0": Data unit output<br>"1": LUT 1 output<br>"2": LUT 2 output<br>"3": LUT 3 output<br>"4": LUT 4 output<br>"5": LUT 5 output<br>"6": LUT 6 output<br>"7": LUT 7 output<br>"8": chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7)<br>"9": chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7)<br>"10": chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7)<br>"11": chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7)<br>"12": io_data_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7)<br>"13": io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7)<br>"14": io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7)<br>"15": io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[7] (for LUTs 4, 5, 6, 7) |
| PRGIO_PRTx_LUT_SELy[11:8]  | LUT_TR1_SEL[3:0] | LUT input signal "tr1_in" source selection:<br>"0": LUT 0 output<br>"1": LUT 1 output<br>"2": LUT 2 output<br>"3": LUT 3 output<br>"4": LUT 4 output<br>"5": LUT 5 output<br>"6": LUT 6 output<br>"7": LUT 7 output<br>"8": chip_data[0] (for LUTs 0, 1, 2, 3); chip_data[4] (for LUTs 4, 5, 6, 7)<br>"9": chip_data[1] (for LUTs 0, 1, 2, 3); chip_data[5] (for LUTs 4, 5, 6, 7)<br>"10": chip_data[2] (for LUTs 0, 1, 2, 3); chip_data[6] (for LUTs 4, 5, 6, 7)<br>"11": chip_data[3] (for LUTs 0, 1, 2, 3); chip_data[7] (for LUTs 4, 5, 6, 7)<br>"12": io_data_in[0] (for LUTs 0, 1, 2, 3); io_data_in[4] (for LUTs 4, 5, 6, 7)<br>"13": io_data_in[1] (for LUTs 0, 1, 2, 3); io_data_in[5] (for LUTs 4, 5, 6, 7)<br>"14": io_data_in[2] (for LUTs 0, 1, 2, 3); io_data_in[6] (for LUTs 4, 5, 6, 7)<br>"15": io_data_in[3] (for LUTs 0, 1, 2, 3); io_data_in[7] (for LUTs 4, 5, 6, 7)     |
| PRGIO_PRTx_LUT_SELy[19:16] | LUT_TR2_SEL[3:0] | LUT input signal "tr2_in" source selection. Encoding is the same as for LUT_TR1_SEL.  |

Figure 8-6. Smart I/O LUT3 Configuration



### 8.5.2.4 Data Unit

Each Smart I/O block includes a data unit (DU) component. The data unit consists of a simple 8-bit datapath. It is capable of performing simple increment, decrement, increment/decrement, shift, and AND/OR operations. The operation performed by the DU is selected using a 4-bit opcode DU\_OPC[3:0] bit field in the PRGIO\_PRTx\_DU\_CTL register.

The data unit component supports up to three input trigger signals (tr0\_in, tr1\_in, tr2\_in) similar to the LUT3 component. These signals are used to initiate an operation defined by the DU opcode. In addition, the data unit also includes two 8-bit input data (data0\_in[7:0] and data1\_in[7:0]) that are used to initialize the 8-bit internal state (data[7:0]) or to provide a reference. The input to these 8-bit data can come from these sources:

- Constant '0x00'
- io\_data\_in[7:0]
- chip\_data\_in[7:0]
- DATA[7:0] bit field of PRGIO\_PRTx\_DATA register

The trigger signals are selected using the DU\_TRx\_SEL[3:0] bit field of the PRGIO\_PRTx\_DU\_SEL register. The DUT\_DATAx\_SEL[1:0] bits of the PRGIO\_PRTx\_DU\_SEL register selects the 8-bit input data source. The size of the DU (number of bits used by the datapath) is defined by the DU\_SIZE[2:0] bits of the PRGIO\_PRTx\_DU\_CTL register. See [Table 8-6](#) for register control details.

Table 8-6. Data Unit Register Control

| Register[BIT_POS]        | Bit Name          | Description  |
|--------------------------|-------------------|--|
| PRGIO_PRTx_DU_CTL[2:0]   | DU_SIZE[2:0]      | Size/width of the data unit (in bits) is DU_SIZE+1. For example, if DU_SIZE is 7, the width is 8 bits.   |
| PRGIO_PRTx_DU_CTL[11:8]  | DU_OPC[3:0]       | Data unit opcode specifies the data unit operation:<br>"0": Count Up<br>"1": Count Down<br>"2": Count Up Wrap<br>"3": Count Down Wrap<br>"4": Count Up/Down<br>"5": Count Up/Down Wrap<br>"6": Rotate Right<br>"7": Shift Right<br>"8": DATA0 & DATA1<br>"9": Majority 3<br>"10": Match DATA1<br>Otherwise: Undefined. |
| PRGIO_PRTx_DU_SEL[3:0]   | DU_TR0_SEL[3:0]   | Data unit input signal "tr0_in" source selection:<br>"0": Constant '0'.<br>"1": Constant '1'.<br>"2": Data unit output.<br>"10-3": LUT 7 - 0 outputs.<br>Otherwise: Undefined.   |
| PRGIO_PRTx_DU_SEL[11:8]  | DU_TR1_SEL[3:0]   | Data unit input signal "tr1_in" source selection. Encoding same as DU_TR0_SEL  |
| PRGIO_PRTx_DU_SEL[19:16] | DU_TR2_SEL[3:0]   | Data unit input signal "tr2_in" source selection. Encoding same as DU_TR0_SEL  |
| PRGIO_PRTx_DU_SEL[25:24] | DU_DATA0_SEL[1:0] | Data unit input data "data0_in" source selection:<br>"0": Constant 0<br>"1": data[7:0].<br>"2": gpio[7:0].<br>"3": DU Reg.   |
| PRGIO_PRTx_DU_SEL[29:28] | DU_DATA1_SEL[1:0] | Data unit input data "data1_in" source selection. Encoding same as DU_DATA0_SEL.   |
| PRGIO_PRTx_DATA[7:0]     | DATA[7:0]         | Data unit input data source.   |

The data unit generates a single output trigger signal (“tr\_out”). The internal state (du\_data[7:0]) is captured in flip-flops and requires clk\_block.

The following pseudo code describes the various datapath operations supported by the DU opcode. Note that “Comb” describes the combinatorial functionality – that is, functionalities that operate independent of previous output states. “Reg” describes the registered functionality – that is, functionalities that operate on inputs and previous output states (registered using flip-flops).

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

// INCR operation: increments data by 1 from an initial value (data0) until it reaches a
// final value (data1).
Comb:tr_out = data_eql_data1_in;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked; //tr0_in is reload signal - loads masked data0
                                // into data
    else if (tr1_in) data <= data_eql_data1_in ? data : data_incr; //increment data until
                                // it equals data1

// INCR_WRAP operation: operates similar to INCR but instead of stopping at data1, it wraps
// around to data0.
Comb:tr_out = data_eql_data1_in;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked;
    else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;

// DECR operation: decrements data from an initial value (data0) until it reaches 0.
Comb:tr_out = data_eql_0;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked;
    else if (tr1_in) data <= data_eql_0      ? data : data_decr;

// DECR_WRAP operation: works similar to DECR. Instead of stopping at 0, it wraps around to
// data0.
Comb:tr_out = data_eql_0;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked;
    else if (tr1_in) data <= data_eql_0      ? data0_masked: data_decr;

// INCR_DECR operation: combination of INCR and DECR. Depending on trigger signals it either
// starts incrementing or decrementing. Increment stops at data1 and decrement stops at 0.
Comb:tr_out = data_eql_data1_in | data_eql_0;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked; // Increment operation takes precedence over
                                // decrement when both signal are available
    else if (tr1_in) data <= data_eql_data1_in ? data : data_incr;
    else if (tr2_in) data <= data_eql_0      ? data : data_decr;

// INCR_DECR_WRAP operation: same functionality as INCR_DECR with wrap around to data0 on
// touching the limits.
Comb:tr_out = data_eql_data1_in | data_eql_0;
Reg: data <= data;
    if (tr0_in)      data <= data0_masked;
```

```

else if (tr1_in) data <= data_eql_data1_in ? data0_masked : data_incr;
else if (tr2_in) data <= data_eql_0 ? data0_masked : data_decr;

// ROR operation: rotates data right and LSB is sent out. The data for rotation is taken from
// data0.
Comb:tr_out = data[0];
Reg: data <= data;
    if (tr0_in) data <= data0_masked;
    else if (tr1_in) {
        data <= {0, data[7:1]} & mask; //Shift right operation
        data[du_size] <= data[0]; //Move the data[0] (LSB) to MSB
    }

// SHR operation: performs shift register operation. Initial data (data0) is shifted out and
// data on tr2_in is shifted in.
Comb:tr_out = data[0];
Reg: data <= data;
    if (tr0_in) data <= data0_masked;
    else if (tr1_in) {
        data <= {0, data[7:1]} & mask; //Shift right operation
        data[du_size] <= tr2_in; //tr2_in Shift in operation
    }

// SHR_MAJ3 operation: performs the same functionality as SHR. Instead of sending out the
// shifted out value, it sends out a '1' if in the last three samples/shifted-out values
// (data[0]), the signal high in at least two samples. otherwise, sends a '0'. This function
// sends out the majority of the last three samples.
Comb:tr_out = (data == 0x03)
              | (data == 0x05)
              | (data == 0x06)
              | (data == 0x07);
Reg: data <= data;
    if (tr0_in) data <= data0_masked;
    else if (tr1_in) {
        data <= {0, data[7:1]} & mask;
        data[du_size] <= tr2_in;
    }

// SHR_EQL operation: performs the same operation as SHR. Instead of shift-out, the output is
// a comparison result (data0 == data1).
Comb:tr_out = data_eql_data1_in;
Reg: data <= data;
    if (tr0_in) data <= data0_masked;
    else if (tr1_in) {
        data <= {0, data[7:1]} & mask;
        data[du_size] <= tr2_in;
    }

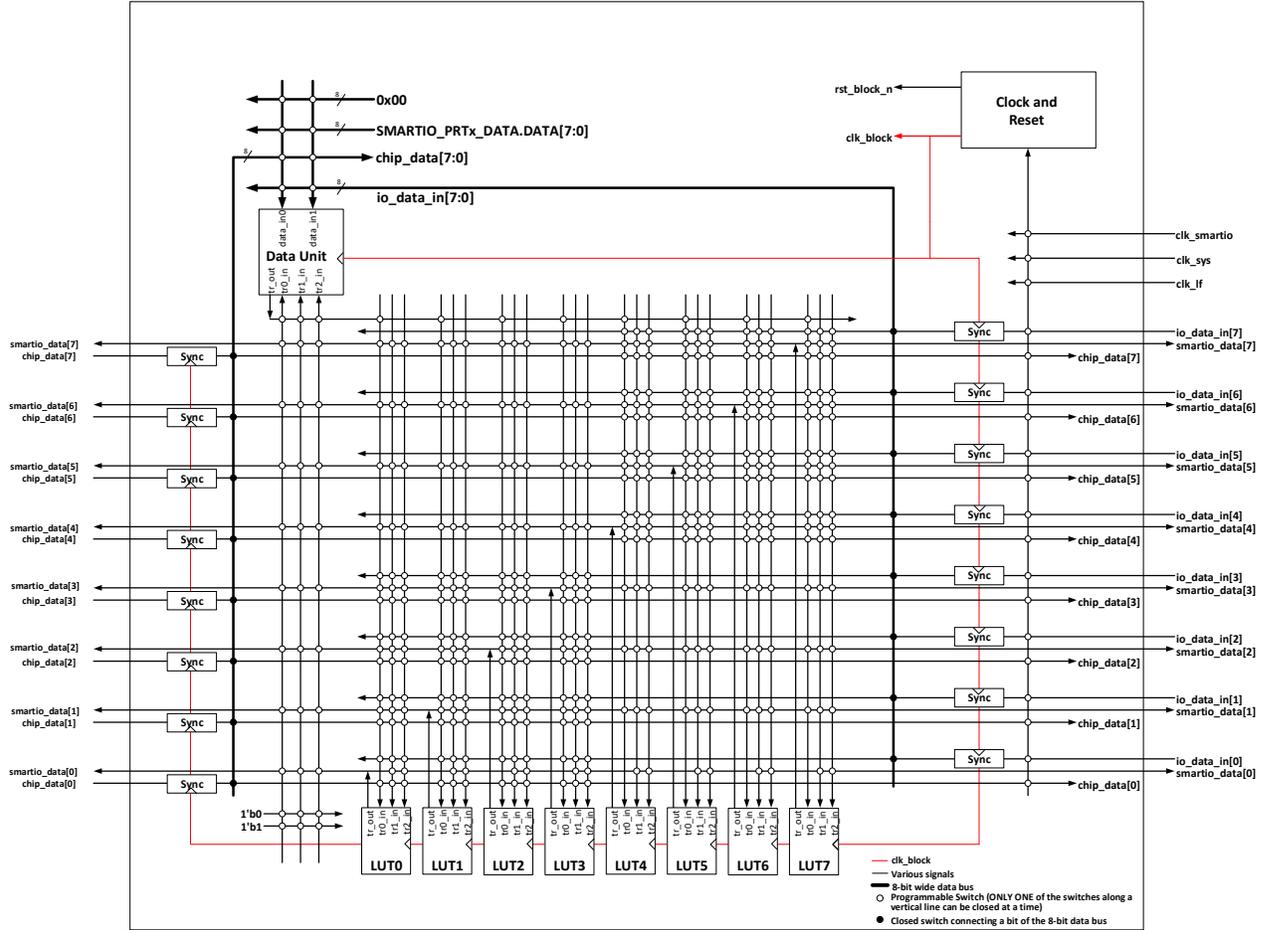
// AND_OR operation: ANDs data1 and data0 along with mask; then, ORs all the bits of the
// ANDed output.
Comb:tr_out = | (data & data1_in & mask);
Reg: data <= data;
    if (tr0_in) data <= data0_masked;

```

### 8.5.3 Routing

The Smart I/O block includes many switches that are used to route the signals in and out of the block and also between various components present inside the block. The routing switches are handled through the PRTGIO\_PRTx\_LUT\_SELy and PRGIO\_PRTx\_DU\_SEL registers. Refer to the *PSoC 4100S Max: PSoc 4 Registers TRM* for details. The Smart I/O internal routing is shown in Figure 8-7. In the figure, note that LUT7 to LUT4 operate on io\_data\_in/chip\_data[7] to io\_data\_in/chip\_data[4] whereas LUT3 to LUT0 operate on io\_data\_in/chip\_data[3] to io\_data\_in/chip\_data[0].

Figure 8-7. Smart I/O Routing



### 8.5.4 Operation

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

1. Before enabling the block, all the components should be configured and the routing should be selected, as explained in [“Block Components” on page 74](#).
2. In addition to configuring the components and routing, some block level settings need to be configured correctly for desired operation.
  - a. Bypass control: The Smart I/O path can be bypassed for a particular GPIO signal by setting the BYPASS[i] bit field in the PRGIO\_PRTx\_CTL register. When bit 'i' is set in the BYPASS[7:0] bit field, the 'i'th GPIO signal is bypassed to the HSIOM signal path directly – Smart I/O logic will not be present in that signal path. This is useful when the Smart I/O functionality is required only on select I/Os.
  - b. Pipelined trigger mode: The LUT3 input multiplexers and the LUT3 component itself do not include any combinatorial loops. Similarly, the data unit also does not include any combinatorial loops. However, when one LUT3 interacts with the other or to the data unit, inadvertent combinatorial loops are possible. To overcome this limitation, the PIPELINE\_EN bit field of the PRGIO\_PRTx\_CTL register is used. When set, all the outputs (LUT3 and data unit) are registered (flopped) before branching out to other components. The output will be unflopped when the PIPELINE\_EN bit is cleared.
3. After the Smart I/O block is configured for the desired functionality, the block can be enabled by setting the ENABLED bit field of the PRGIO\_PRTx\_CTL register. If disabled, the Smart I/O block is put in bypass mode, where the GPIO signals are directly controlled by the HSIOM signals and vice-versa. The Smart I/O block must be configured; that is, all register settings must be updated before enabling the block to prevent glitches during register updates.

Table 8-7. Smart I/O Block Controls

| Register [BIT_POS]  | Bit Name    | Description  |
|---------------------|-------------|--|
| PRGIO_PRTx_CTL[25]  | PIPELINE_EN | Enable for pipeline register:<br>'0': Disabled (register is bypassed).<br>'1': Enabled   |
| PRGIO_PRTx_CTL[31]  | ENABLED     | Enable Smart I/O. Should only be set to '1' when the Smart I/O is completely configured:<br>'0': Disabled (signals are bypassed; behavior as if BYPASS[7:0] is 0xFF). When disabled, the block (data unit and LUTs) reset is activated.<br>If the block is disabled:<br>- The PIPELINE_EN register field should be set to '1', to ensure low power consumption.<br>- The CLOCK_SRC register field should be set to 20 to 30 (clock is constant '0'), to ensure low power consumption.<br>'1': Enabled. When enabled, it takes three “clk_block” clock cycles until the block reset is deactivated and the block becomes fully functional. This action ensures that the I/O pins' input synchronizer states are flushed when the block is fully functional. |
| PRGIO_PRTx_CTL[7:0] | BYPASS[7:0] | Bypass of the Smart I/O, one bit for each I/O pin: BYPASS[i] is for I/O pin i. When ENABLED is '1', this field is used. When ENABLED is '0', this field is not used and Smart I/O is always bypassed.<br>'0': No bypass (Smart I/O is present in the signal path)<br>'1': Bypass (Smart I/O is absent in the signal path)  |

## 8.6 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.

## 8.7 Behavior in Low-Power Modes

Table 8-8 shows the status of GPIOs in low-power modes.

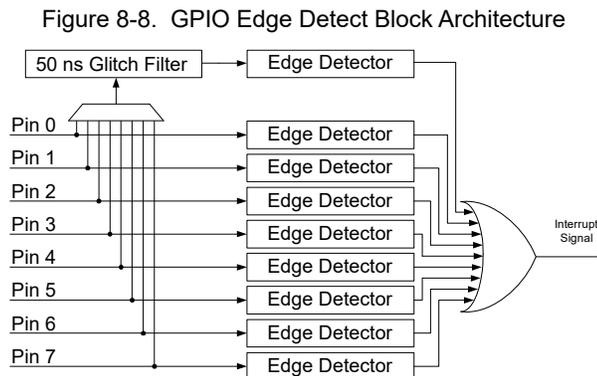
Table 8-8. GPIO in Low-Power Modes

| Low-Power Mode | Status   |
|----------------|--|
| Sleep          | <ul style="list-style-type: none"> <li>■ GPIOs are active and can be driven by peripherals such as CapSense, CTBm, SAR ADC, TCPWM, SCBs, and low-power comparators, which can operate in sleep mode.</li> <li>■ Input buffers are active; thus an interrupt on any I/O can be used to wake up the CPU.</li> <li>■ AMUXBUS connections are available.</li> </ul>  |
| Deep-Sleep     | <ul style="list-style-type: none"> <li>■ GPIO output pin states are latched and maintain the last output driver state, except the I<sup>2</sup>C and SPI pins. SCB (I<sup>2</sup>C and SPI) block can work in the deep-sleep mode and can wake up the CPU on address match or SPI slave select event. The low-power comparator can receive signals from its dedicated pins and can wake up the CPU. CTBm is also functional in this mode with dedicated pins.</li> <li>■ Input buffers are also active in this mode; pin interrupts are functional.</li> <li>■ AMUXBUS connections are not available.</li> </ul> |

## 8.8 Interrupt

In the PSoC 4 device, all the port pins have the capability to generate interrupts. As shown in Figure 8-2, the pin signal is routed to the interrupt controller through the GPIO Edge Detect block.

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



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 8-9.

Table 8-9. Edge Detector Configuration

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

Besides the pins, an edge detector is also present at the glitch filter output. This filter can be used on one of the pins of a port. The pin is selected by writing to the FLT\_SEL field of the GPIO\_PRTx\_INTR\_CFG register as shown in [Table 8-10](#).

Table 8-10. Glitch filter Input Selection

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

The edge detector outputs of a port are ORed together and then routed to the interrupt controller (NVIC in the CPU subsystem). Thus, there is only one interrupt vector per port. On a pin interrupt, it is required to know which pin caused an interrupt. This is done by reading the Port Interrupt Status register, GPIO\_PRTx\_INTR. This register not only includes the information on which pin triggered the interrupt, it also includes the pin status; it allows the CPU to read both information in a single read operation. This register has one more important use – to clear the interrupt. Writing ‘1’ to the corresponding status bit clears the pin interrupt. It is important to clear the interrupt status bit; otherwise, the interrupt will occur repeatedly for a single trigger or respond only once for multiple triggers, which is explained later in this section. Also, note that when the Port Interrupt Control Status register is read when an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code. [Table 8-11](#) shows the Port Interrupt Status register bit fields.

Table 8-11. Port Interrupt Status Register

| GPIO_PRTx_INTR   | Description  |
|------------------|--|
| 0000b to 0111b   | Interrupt status on pin 0 to pin 7. Writing ‘1’ to the corresponding bit clears the interrupt. |
| 1000b            | Interrupt status from the glitch filter  |
| 10000b to 10111b | 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](#), which gives an option of Level and Rising Edge detect. If the Level option is selected, an interrupt is triggered repeatedly as long as the Port Interrupt Status register bit is set. If the Rising Edge detect option is selected, an interrupt is triggered only once if the Port Interrupt Status register is not cleared. Thus, it is important to clear the interrupt status bit if the Edge Detect block is used.

## 8.9 Peripheral Connections

### 8.9.1 Firmware Controlled GPIO

For standard firmware-controlled GPIO using registers, the GPIO mode must be selected in the HSIOM\_PORT\_SELx register. 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 effect on the pin status.

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

### 8.9.2 Analog I/O

Analog resources, such as LPCOMP, SARMUX, and CTBm, which require low-impedance routing paths have dedicated pins. Dedicated analog pins provide direct connections to specific analog blocks. They help improve performance and should be given priority over other pins when using these analog resources. 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 8-2](#)) and the respective connection should be enabled in the specific analog resource. This can be done via registers associated with the respective analog resources.

To configure a GPIO as an analog pin connecting to AMUXBUS, it should be configured in high-impedance analog mode and then routed to AMUXBUS using the HSIOM\_PORT\_SELx register. While it is preferred for analog pins to disable the input buffer, it is acceptable to enable the input buffer if simultaneous analog and digital input features are required.

### 8.9.3 LCD Drive

All GPIOs have the capability of driving an LCD common or segment. HSIOM\_PORT\_SELx registers are used to select the pins for LCD drive. See the [LCD Direct Drive chapter on page 285](#) for details.

### 8.9.4 CapSense

CapSense (MSC) supports legacy GPIO interface and also Control Matrix (CTRL MUX) interface for limited number of pins (up to 32, i.e., 16 per channel). The pins that support MSC can be configured as CapSense widgets such as buttons, slider elements, touchpad elements, or proximity sensors. CapSense also requires shield lines. See the [PSoC 4 and PSoC 6 MCU CAPSENSE Design Guide](#) for more details. [Table 8-12](#) shows the GPIO and HSIOM settings required for CapSense.

Table 8-12. CapSense Settings

| CapSense Pin                      | GPIO Drive Mode (GPIO_PRTx_PC) | Digital Input Buffer Setting (GPIO_PRTx_PC2) | HSIOM Setting |                           |
|-----------------------------------|--------------------------------|--|---------------|---------------------------|
|                                   |                                |  | AMUX Mode     | CTRLMUX Mode <sup>a</sup> |
| Sensor                            | High-Impedance Analog          | Disable Buffer                               | CSD_SENSE     | GPIO                      |
| Shield                            | High-Impedance Analog          | Disable Buffer                               | CSD_SHIELD    | GPIO                      |
| CMOD1 or CMOD3 (normal operation) | High-Impedance Analog          | Disable Buffer                               | AMUXBUS A     | GPIO                      |
| CMOD2 or CMOD4 (normal operation) | High-Impedance Analog          | Disable Buffer                               | AMUXBUS A     | GPIO                      |

a. CTRLMUX has its controls within the CapSense block. See the [PSoC 4 and PSoC 6 MCU CAPSENSE Design Guide](#) for more details.

## 8.9.5 Serial Communication Block (SCB)

SCB blocks can be configured as UART, I<sup>2</sup>C, and SPI have dedicated connections to the I/O pins. See the [device datasheet](#) for details on these dedicated pins. When UART and SPI mode are used, the SCB controls the digital output buffer drive mode for the input pin in order to keep the pin in the high-impedance state. The SCB block disables the output buffer at the UART Rx pin and MISO pin when configured as SPI master, and MOSI and select line when configured as SPI slave. This functionality overrides the drive mode settings, which is done using the GPIO\_PRTx\_PC register.

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

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

## 8.10 Registers

Table 8-13. I/O Registers

| Name                | Description  |
|---------------------|--|
| GPIO_PRTx_DR        | Port Output Data Register  |
| GPIO_PRTx_DR_SET    | Port Output Data Set Register  |
| GPIO_PRTx_DR_CLR    | Port Output Data Clear Register  |
| GPIO_PRTx_DR_INV    | Port Output Data Inverting Register  |
| GPIO_PRTx_PS        | Port Pin State Register - Reads the logical pin state of I/O                                   |
| GPIO_PRTx_PC        | Port Configuration Register - Configures the output drive mode, input threshold, and slew rate |
| GPIO_PRTx_PC2       | Port Secondary Configuration Register - Configures the input buffer of I/O pin                 |
| GPIO_PRTx_INTR_CFG  | Port Interrupt Configuration Register  |
| GPIO_PRTx_INTR      | Port Interrupt Status Register   |
| HSIOM_PORT_SELx     | HSIOM Port Selection Register  |
| PRGIO_PRTx_CTL      | Smart I/O port control register  |
| PRGIO_PRTx_SYNC_CTL | Smart I/O Synchronization control register   |
| PRGIO_PRTx_LUT_SELy | Smart I/O y <sup>th</sup> LUT component input selection register                               |
| PRGIO_PRTx_LUT_CTLy | Smart I/O y <sup>th</sup> LUT component control register                                       |
| PRGIO_PRTx_DU_SEL   | Smart I/O data unit input selection register   |
| PRGIO_PRTx_DU_CTL   | Smart I/O data unit control register   |
| PRGIO_PRTx_DATA     | Smart I/O data unit input data source register   |

**Note** The 'x' in the GPIO register name denotes the port number. For example, GPIO\_PTR1\_DR is the Port 1 output data register. The 'x' in the Smart I/O register name denotes the Smart I/O port number. The Smart I/O port number and the actual port number may vary. See [8.5 Smart I/O on page 73](#) for details.

# 9. Clocking System



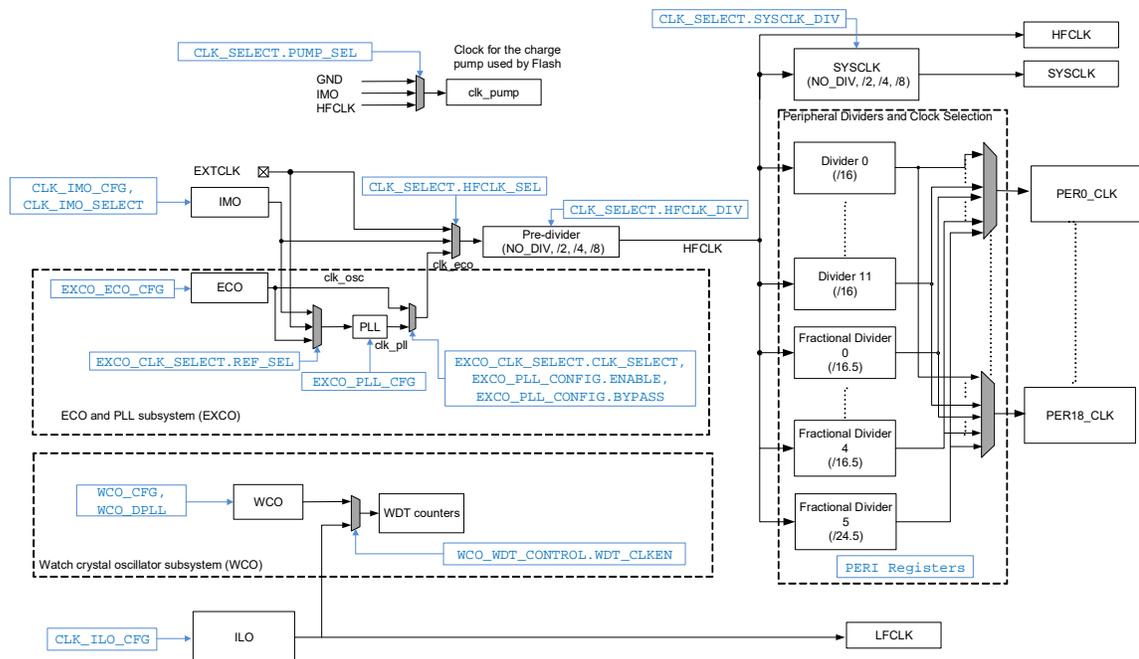
The PSoC 4 clock system includes these clock resources:

- Two internal clock sources:
  - 24-48 MHz internal main oscillator (IMO)
  - 20-80 kHz (typically 40-kHz) internal low-speed oscillator (ILO)
- 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 48 MHz, selected from IMO, ECO, external clock, or PLL
- Low-frequency clock (LFCLK) sourced by ILO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- Twelve 16-bit peripheral clock dividers
- Six fractional dividers (five 16.5 fractional dividers and one 24.5 fractional divider) for accurate clock generation
- Nineteen peripheral clocks

## 9.1 Block Diagram

Figure 9-1 gives a generic view of the clocking system in PSoC 4 devices.

Figure 9-1. Clocking System Block Diagram



The five clock sources in the device are IMO, ECO, EXTCLK, WCO, and ILO, as shown in [Figure 9-1](#). There is a PLL that can be configured to take clock sources from the IMO, the EXTCLK, or the ECO. The PLL output frequency can be as high as 104 MHz. Although the PLL output frequency can be as high as 104 MHz, not all blocks accept frequencies higher than 48 MHz. Refer to [Phase-Locked Loop \(PLL\) chapter on page 92](#) for details. The HFCLK mux selects the HFCLK source from the EXTCLK, ECO, PLL (through the EXCO mux) or the IMO. The HFCLK frequency can be a maximum of 48 MHz.

## 9.2 Clock Sources

### 9.2.1 Internal Main Oscillator (IMO)

The IMO is an accurate, high-speed internal (crystal-less) oscillator that is available as the main clock source during Active and Sleep modes. It is the default clock source for the device. Its frequency can be changed in 4-MHz steps between 24 MHz and 48 MHz. Refer to the [device datasheet](#) for the exact specifications of clocks.

The IMO frequency is changed using the CLK\_IMO\_SELECT register, as shown in [Table 9-1](#). The default frequency is 24 MHz.

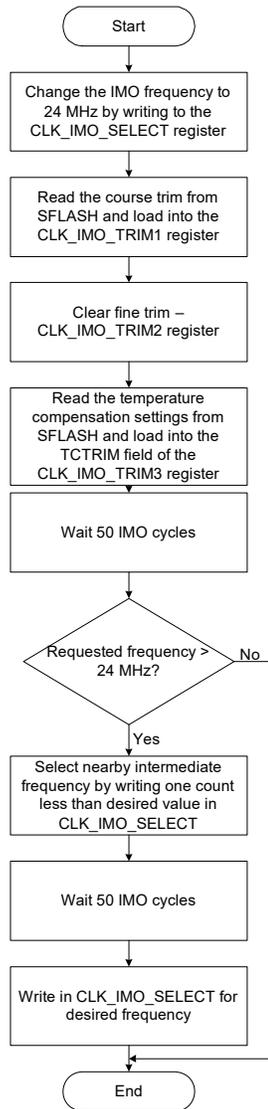
Table 9-1. IMO Frequency

| CLK_IMO_SELECT[2:0] | Nominal IMO Frequency | Corresponding SFLASH Registers with TRIM Values |
|---------------------|-----------------------|---|
| 0                   | 24 MHz                | SFLASH_IMO_TRIM_LT0, SFLASH_IMO_TCTRIM_LT0      |
| 1                   | 28 MHz                | SFLASH_IMO_TRIM_LT4, SFLASH_IMO_TCTRIM_LT4      |
| 2                   | 32 MHz                | SFLASH_IMO_TRIM_LT8, SFLASH_IMO_TCTRIM_LT8      |
| 3                   | 36 MHz                | SFLASH_IMO_TRIM_LT12, SFLASH_IMO_TCTRIM_LT12    |
| 4                   | 40 MHz                | SFLASH_IMO_TRIM_LT16, SFLASH_IMO_TCTRIM_LT16    |
| 5                   | 44 MHz                | SFLASH_IMO_TRIM_LT20, SFLASH_IMO_TCTRIM_LT20    |
| 6                   | 48 MHz                | SFLASH_IMO_TRIM_LT24, SFLASH_IMO_TCTRIM_LT24    |

To get the accurate IMO frequency, trim registers are provided – CLK\_IMO\_TRIM1 provides coarse trimming with a step size of 120 kHz, CLK\_IMO\_TRIM2 is for fine trimming with a step size of 15 kHz, and the TCTRIM field in CLK\_IMO\_TRIM3 is for temperature compensation. Trim settings are generated during manufacturing for every frequency that can be selected by CLK\_IMO\_SELECT. The trim settings CLK\_IMO\_TRIM1 and CLK\_IMO\_TRIM3 are stored in SFLASH.

The trim settings are loaded during device startup; however, firmware can load new trim values and change the frequency in run time. Follow the algorithm in [Figure 9-2](#) to change the IMO frequency.

Figure 9-2. Change IMO Frequency



### 9.2.1.1 Startup Behavior

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

### 9.2.1.2 Programming Clock (36-MHz)

IMO must be set to 48 MHz to program the flash. It is used to drive the charge pumps of flash and for program/erase timing purposes.

## 9.2.2 Internal Low-speed Oscillator (ILO)

The ILO operates with no external components and outputs a stable clock at 40-kHz nominal. The ILO is relatively low power and low accuracy. Refer to the [device datasheet](#) for ILO specifications. It can be calibrated periodically using a higher accuracy, HFCLK to improve accuracy. The ILO is available in all power modes. The ILO is used as the system low-frequency clock (LFCLK) in the device, which is used to generate low-frequency clocks. The ILO is enabled and disabled with register CLK\_ILO\_CONFIG bit ENABLE.

### 9.2.3 External Clock (EXTCLK)

The external clock (EXTCLK) is a MHz range clock that can be generated from a signal on a designated PSoC 4100S Max pin (P0.6). This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. Refer to the [device datasheet](#) for the required specifications of the external clock. The device always starts up using the IMO and the EXTCLK must be enabled in user mode; so the device cannot be started from a reset, which is clocked by the EXTCLK.

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the [I/O System chapter on page 66](#) for more details.

### 9.2.4 External Crystal Oscillator (ECO)

The PSoC 4 device contains an oscillator to drive an external 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. Refer to the [device datasheet](#) for required crystal specifications.

The ECO can be enabled by using the EXCO\_ECO\_CONFIG.CLK\_EN (bit 0) and EXCO\_ECO\_CONFIG.ENABLE (bit 31) register bit fields.

#### 9.2.4.1 ECO Trimming

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

Maximum peak to peak value:

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

Transconductance:

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

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

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

Table 9-2. ATRIM and WDTRIM Setting

| VPP   | ATRIM | WDTRIM |
|---|-------|--------|
| $0.4 \text{ V} \leq V_{PP} < 0.5 \text{ V}$ | 0x00  | 0x00   |
| $V_{PP} < 0.6 \text{ V}$                    | 0x01  | 0x00   |
| $V_{PP} < 0.7 \text{ V}$                    | 0x02  | 0x01   |
| $V_{PP} < 0.8 \text{ V}$                    | 0x03  | 0x01   |
| $V_{PP} < 0.9 \text{ V}$                    | 0x04  | 0x02   |
| $V_{PP} < 1.025 \text{ V}$                  | 0x05  | 0x02   |
| $V_{PP} < 1.15 \text{ V}$                   | 0x06  | 0x03   |
| $V_{PP} < 1.275 \text{ V}$                  | 0x07  | 0x03   |

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

Table 9-3. GTRIM Setting

| $G_m$  | GTRIM |
|--|-------|
| $0 \text{ mA/V} < g_m \leq 4.5 \text{ mA/V}$   | 0x00  |
| $4.5 \text{ mA/V} < g_m \leq 9 \text{ mA/V}$   | 0x01  |
| $9 \text{ mA/V} < g_m \leq 13.5 \text{ mA/V}$  | 0x02  |
| $13.5 \text{ mA/V} < g_m \leq 18 \text{ mA/V}$ | 0x03  |

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

Table 9-4. FTRIM and RTRIM Setting

| Nominal Frequency $f$ (MHz)              | RTRIM | FTRIM |
|--|-------|-------|
| $f > 30 \text{ MHz}$                     | 0x00  | 0x00  |
| $30 \text{ MHz} \geq f > 24 \text{ MHz}$ | 0x01  | 0x01  |
| $24 \text{ MHz} \geq f > 17 \text{ MHz}$ | 0x02  | 0x02  |
| $17 \text{ MHz} \geq f$                  | 0x03  | 0x03  |

### 9.2.5 Phase-Locked Loop (PLL)

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

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

### 9.2.6 Watch Crystal Oscillator (WCO)

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

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]. 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  $\pm 1\%$ . The IMO trimming logic of the WCO can be enabled by the use of the DPLL\_ENABLE bit of the WCO\_CONFIG. The user firmware, when using this feature, must make sure that there is a minimum time of 500 ms between the WCO enable and the DPLL\_ENABLE events.

## 9.3 Clock Supervision

In PSoC 4100S Max device, ECO and PLL subsystem is called as the EXCO block. The ECO and PLL Subsystem (EXCO) block can supervise its output clock (clk\_eco). There are mainly two blocks (CSV and PGM\_DELAY) to implement this feature.

### 9.3.1 Clock Supervision Block (CSV)

The PSoC 4100S Max device has one clock supervision block, which uses a reference clock (IMO clock) to check that a monitored clock (clk\_exco) is within an allowed frequency window. The CSV can be enabled by using the EXCO\_REF\_CTL.CSV\_EN (bit 31) register bit fields.

If the monitored clock fails, the CSV block can switch the source of clk\_exco to IMO clock by setting the EXCO\_REF\_CTL.CSV\_CLK\_SW\_EN bit. If setting the EXCO\_REF\_CTL.CSV\_INT\_EN bit or EXCO\_REF\_CTL.CSV\_TRIG\_EN bit, switching the clk\_exco source clock event can also assert the interrupt and trigger outputs. While CSV block will continue to monitor the output clock, it will not assert the interrupt or trigger outputs again until the original event has been cleared by FW.

### 9.3.2 Clock Supervision Interrupt (CSI)

Enable or disable the CSV interrupt by setting or clearing the CSV\_CLK\_SW, WD\_ERR, and PLL\_LOCK bits in the EXCO\_INTR\_MASK register. FW can get specific interrupt information from the EXCO\_INTR\_MASKED register.

### 9.3.3 Programmable Delay Block (PGM\_DELAY)

For the CSV block clock switch over to be successful, it is essential that the failing clock continues to produce clock cycles. If on the other hand, the clock suddenly stops completely the switch over mechanism will fail. To address this type of failure, PSoC 4100S Max device has one programmable delay (PGM\_DELAY) block.

PGM\_DELAY block is a down counter clocked by IMO clock. The starting value is 16-bit programmable. PGM\_DELAY block can be enabled and configured using the EXCO\_RSTDLY\_CTL.EN bit and EXCO\_RSTDLY\_CTL.LOAD bit.

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

## 9.4 Clock Distribution

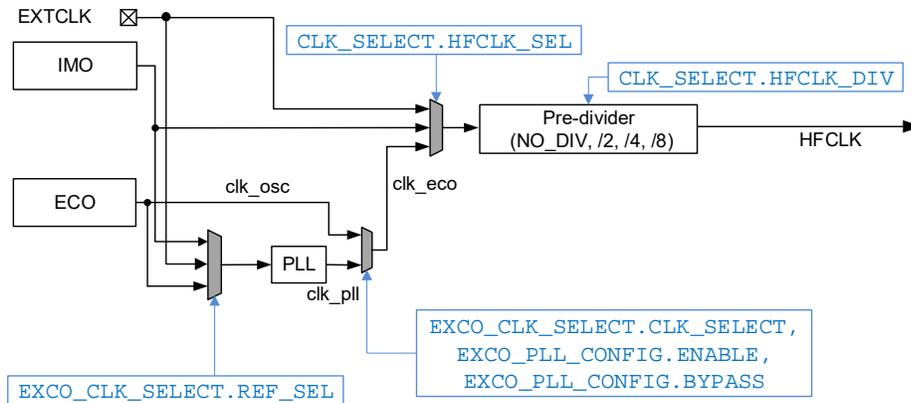
PSoC 4 clocks are developed and distributed throughout the device, as shown in [Figure 9-1](#). The distribution configuration options are as follows:

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

## 9.4.1 HFCLK and PLL Input Selection

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

Figure 9-3. HFCLK and PLL Selection Options



The PLL input has a multiplexer that selects between IMO, ECO, and EXTCLK output signals. The input selection for the PLL is in the EXCO\_CLK\_SELECT[2:1] register.

For HFCLK, it is a two-stage selection. The two sources clk\_osc and clk\_pll are first multiplexed using an EXCO multiplexer whose selection is configured in EXCO\_CLK\_SELECT[0], EXCO\_PLL\_CONFIG[31] and EXCO\_PLL\_CONFIG[21:20] registers. The selection options for EXCO clock source (clk\_eco) are as shown in Table 9-5. The second multiplexer called the HFCLK multiplexer selects between the EXCO output (clk\_eco), IMO, and EXTCLK. This selection is configured in the CLK\_SELECT[1:0] register.

Table 9-5. EXCO Clock Source Selection Options

| EXCO_PLL_CONFIG.ENABLE | EXCO_CLK_SELECT.CLK_SELECT | EXCO_PLL_CONFIG.BYPASS_SEL | EXCO_PLL_STATUS.LOCKED <sup>a</sup> | clk_eco |
|------------------------|----------------------------|----------------------------|-------------------------------------|---------|
| 0                      | x                          | x                          | x                                   | clk_osc |
| 1                      | 0                          | 0x                         | 0                                   | clk_osc |
| 1                      | 0                          | 0x                         | 1                                   | clk_pll |
| 1                      | 0                          | 10                         | x                                   | clk_osc |
| 1                      | 0                          | 11                         | x                                   | clk_pll |
| 1                      | 1                          | xx                         | x                                   | clk_pll |

a. This field is not firmware writable, it is chosen by the PLL Hardware.

In PSoC 4100S Max devices, when IMO is used as a reference to the PLL and the ECO is disabled, or ECO crystal is not mounted on the board, the firmware should toggle the CLK\_ECO bit in the EXCO\_PGM\_CLK register to start the PLL. The EXCO\_PGM\_CLK register is explained in the *PSoC 4100S Max: PSoC 4 Registers TRM*.

Toggle the EXCO\_PGM\_CLK.CLK\_ECO bit five times by alternatively writing a 0 and 1 to it after setting the EXCO\_PGM\_CLK.ENABLE bit. After the toggle, clear the EXCO\_PGM\_CLK.ENABLE bit by writing a 0 to start the PLL.

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

### 9.4.2 High-Frequency Clock (HFCLK) Input Selection

HFCLK in PSoC 4 has four input options: IMO, ECO, PLL, and EXTCLK. The ECO (clk\_osc) and PLL (clk\_pll) are first selected in a stage to get the ECO and PLL subsystem clock (clk\_eco). This input is selected using the EXCO.CLK\_SELECT register's CLK\_SELECT bit, EXCO.PLL\_CONFIG register's ENABLE bit and BYPASS\_SEL bits. The HFCLK input is selected using the CLK\_SELECT register's HFCLK\_SEL bits. Refer to the *PSoC 4100S Max: PSoC 4 Registers TRM* for the details of these registers and included bit fields.

Pre-divider is provided for HFCLK to limit the peak current of the device. The divider options are 2, 4, and 8 configured using HFCLK\_DIV bits of the CLK\_SELECT register. Default divider is 4.

### 9.4.3 Low-Frequency (LFCLK) Input Selection

Only ILO can be the source for LFCLK in the PSoC 4100S Max device.

### 9.4.4 System Clock (SYSCLK) Prescaler Configuration

The SYSCLK Prescaler allows the device to divide the HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by powers of 2 between  $2^0 = 1$  and  $2^3 = 8$ . The prescaler divide value is set using register CLK\_SELECT bits SYSCLK\_DIV, as described in [Table 9-6](#). The prescaler is initially configured to divide by 1.

Table 9-6. SYSCLK Prescaler Divide Value Bits SYSCLK\_DIV

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

### 9.4.5 Peripheral Clock Divider Configuration

PSoC 4 has eighteen clock dividers, which include twelve 16-bit clock dividers, five 16.5-bit fractional clock dividers, and one 24.5-bit fractional clock divider. Fractional clock dividers allow the clock divisor to include a fraction of 0..31/32. The formula for the output frequency of a fractional divider is  $F_{out} = F_{in} / (INT16\_DIV + (FRAC5\_DIV/32))$ . For example, a 16.5-divider with an integer divide value of 2 (INT16\_DIV=3, FRAC5\_DIV=0), produces signals to generate a 16-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 3 (INT16\_DIV=3, FRAC5\_DIV=0), produces signals to generate a 12-MHz clock from a 48-MHz HFCLK. A 16.5-divider with an integer divide value of 2 (INT16\_DIV=3) and a fractional divider of 16 (FRAC5\_DIV=16) produces signals to generate a 13.7-MHz clock from a 48-MHz HFCLK. Not all 13.7-MHz clock periods are equal in size; half of them will be 3 HFCLK cycles and half of them will be 2 HFCLK cycles.

Fractional dividers are useful when a high-precision clock is required (for example, for a UART/SPI serial interface). Fractional dividers are not used when a low jitter clock is required, because the clock periods have a jitter of 1 HFCLK cycle. The divide value for each of the 12 integer clock dividers are configured with the PERI\_DIV\_16\_CTLx registers and the five 16.5-bit fractional clock dividers are configured with the PERI\_DIV\_16\_5\_CTLx registers. [Table 9-7](#) and [Table 9-8](#) describe the configurations for these registers. The 24.5-bit fractional divider is configured using the PERI\_DIV\_24\_5\_CTL register. [Table 9-9](#) describes the configuration for these registers.

Table 9-7. Non-Fractional Peripheral Clock Divider Configuration Register PERI\_DIV\_16\_CTLx

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

Table 9-8. Fractional Peripheral Clock Divider Configuration Register PERI\_DIV\_16\_5\_CTLx

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

Table 9-9. Fractional Peripheral Clock Divider Configuration Register PERI\_DIV\_24\_5\_CTL

| 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 HFCLK 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].   |

This register acts as the command register for all 16 integer dividers and four fractional dividers. Each divider can be enabled using the PERI\_DIV\_CMD register format is as follows:

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

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

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

The (PA\_SEL\_TYPE, PA\_SEL\_DIV) field pair allows a divider to be phase-aligned with another divider. 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 9-10 shows the mapping of the MUX output to the corresponding peripheral blocks (shown in Figure 9-1). Any of the peripheral clock dividers can be mapped to a specific peripheral by using their respective PERI\_PCLK\_CTLx register.

Table 9-10. PSoC 4100S Max Device Peripheral Clock Multiplexer Output Mapping

| PERI#_CLK | Peripheral    |
|-----------|---------------|
| 0         | SCB0          |
| 1         | SCB1          |
| 2         | SCB2          |
| 3         | SCB3          |
| 4         | SCB4          |
| 5         | MSC0          |
| 6         | TCPWM0        |
| 7         | TCPWM1        |
| 8         | TCPWM2        |
| 9         | TCPWM3        |
| 10        | TCPWM4        |
| 11        | TCPWM5        |
| 12        | TCPWM6        |
| 13        | TCPWM7        |
| 14        | SmartIO Port1 |
| 15        | SmartIO Port2 |
| 16        | SmartIO Port3 |
| 17        | LCD           |
| 18        | SAR ADC       |
| 19        | MSC1          |
| 20        | CAN FD        |
| 21        | MSC0 SYNC     |
| 22        | MSC1 SYNC     |
| 23        | I2S           |

Table 9-11. Programmable Clock Control Register - PERI\_PCLK\_CTLx

| Bits | Name     | Description   |
|------|----------|---|
| 3:0  | 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<br>1: 16.0 (integer) clock dividers<br>2: 16.5 (fractional) clock dividers<br>3: 24.5 (fractional) clock dividers   |

## 9.5 Low-Power Mode Operation

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

## 9.6 Register List

Table 9-12. Clocking System Register List

| Register Name         | Description   |
|-----------------------|---|
| CLK_IMO_TRIM1         | IMO Trim Register - This register contains IMO trim for coarse correction.  |
| CLK_IMO_TRIM2         | IMO Trim Register - This register contains IMO trim for fine correction.  |
| CLK_IMO_TRIM3         | IMO Trim Register - This register contains the temperature compensation trim settings for IMO and trim settings to adjust the step size of the coarse and fine correction of IMO frequency. |
| PWR_BG_TRIM1          | Bandgap Trim Registers - These registers control the trim of the bandgap reference, allowing manipulation of the voltage references in the device.  |
| PWR_BG_TRIM2          |   |
| CLK_ILO_CONFIG ILO    | Configuration Register - This register controls the ILO configuration.  |
| CLK_IMO_CONFIG IMO    | Configuration Register - This register controls the IMO configuration.  |
| CLK_SELECT            | Clock Select - This register controls clock tree configuration and selects different sources for the system clocks.   |
| EXCO_CLK_SELECT       | Clock select for ECO and PLL subsystem - This register controls selecting the PLL input and output of the ECO and PLL subsystem.  |
| EXCO_ECO_CONFIG       | This register controls the ECO configuration.   |
| EXCO_PLL_CONFIG       | This register controls the PLL configuration.   |
| WCO_CONFIG            | WCO Enable. This register enables or disables the external WCO  |
| EXCO_REF_CTL          | This register controls clock supervision for a clock tree.  |
| EXCO_REF_LIMIT        | Clock Supervision Reference Limits.   |
| EXCO_MON_CTL          | Clock Supervision Monitor Control.  |
| EXCO_INTR             | Interrupt Request Register.   |
| EXCO_INTR_SET         | Interrupt Set Register.   |
| EXCO_INTR_MASK.       | Interrupt Mask Register   |
| EXCO_INTR_MASKED      | Interrupt Masked Register.  |
| EXCO_RSTDLY_CTL       | Programmable Delay Counter Control.   |
| EXCO_RSTDLY           | Programmable Delay Counter Initial Amount.  |
| EXCO_RSTDLY_COUNT_VAL | Programmable Delay Counter Value.   |
| PERI_DIV_16_CTLx      | Peripheral Clock Divider Control Registers - These registers configure the peripheral clock dividers, setting integer divide value, and enabling or disabling the divider.                  |
| PERI_DIV_16_5_CTLx    | Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting fractional divide value, and enabling or disabling the divider.    |
| PERI_PCLK_CTLx        | Programmable Clock Control Registers - These registers are used to select the input clocks to peripherals.  |
| PERI_DIV_24_5_CTL     | Peripheral Clock Fractional Divider Control Registers - These registers configure the peripheral clock dividers, setting the fractional divide value and enabling or disabling the divider. |

# 10. Power Supply and Monitoring



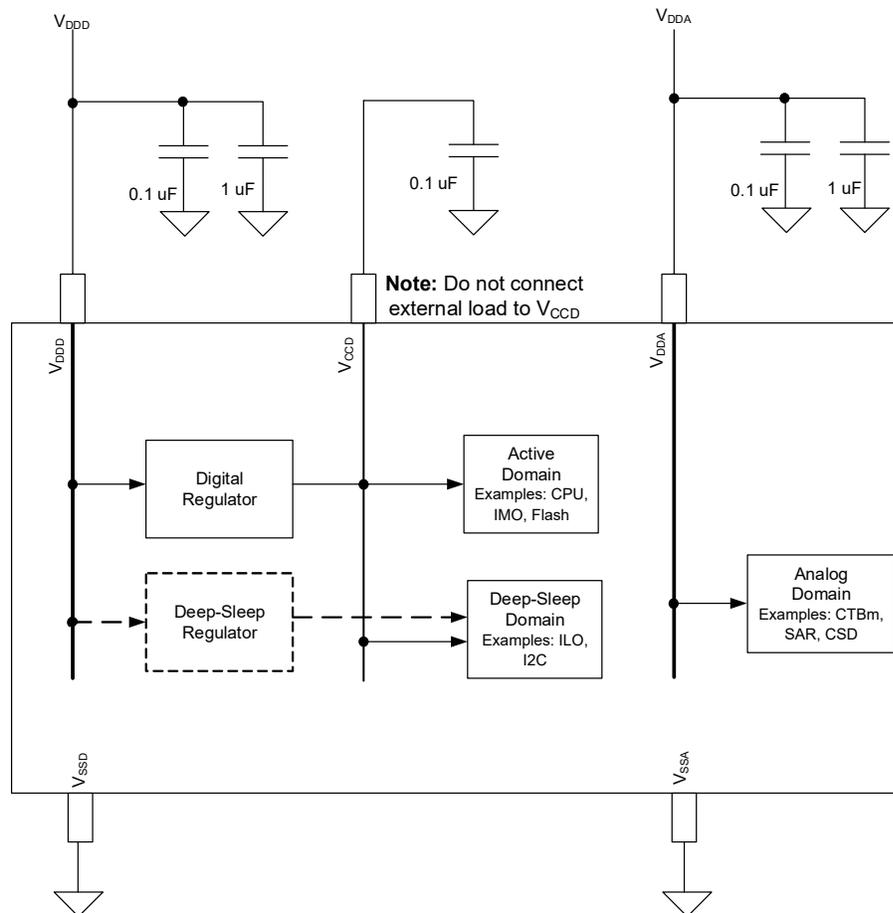
PSoC 4 is capable of operating from a 1.71 V to 5.5 V externally supplied voltage. This is supported through one of the two following operating ranges:

- 1.80 V to 5.50 V supply input to the internal regulators
- 1.71 V to 1.89 V<sup>1</sup> direct supply

There are two internal regulators to support the various power modes - Active digital regulator and Deep-Sleep regulator.

## 10.1 Block Diagram

Figure 10-1. Power System Block Diagram



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

Figure 10-1 shows the power system diagram and all the power supply pins. The system has one regulator in Active mode for the digital circuitry. There is no analog regulator; the analog circuits run directly from the  $V_{DDA}$  input. There is a separate regulator for Deep-Sleep mode.

The supply voltage range is 1.71 V to 5.5 V with all functions and circuits operating in that range. The device allows two distinct modes of power supply operation: unregulated external supply and regulated external supply modes.

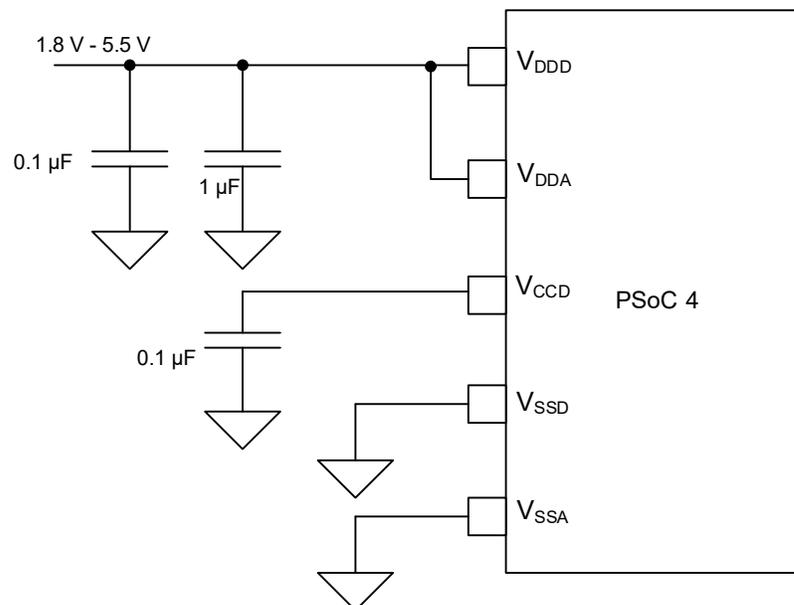
## 10.2 Power Supply Scenarios

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

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

If a 1.8-V to 5.5-V supply is to be used as the unregulated power supply input, it should be connected as shown in Figure 10-2.

Figure 10-2. Single Regulated  $V_{DD}$  Supply



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

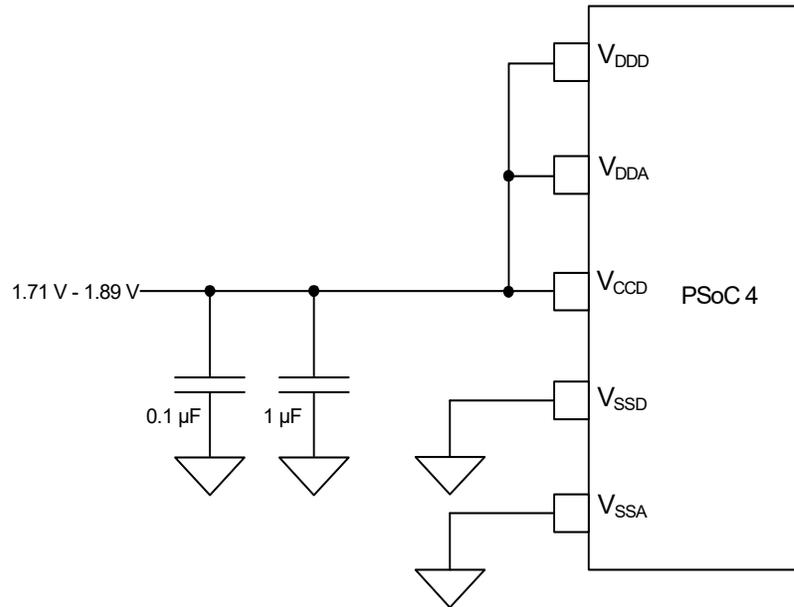
Bypass capacitors are also required from  $V_{DD}$  to ground; typical practice for systems in this frequency range is to use a bulk capacitor in the  $1 \mu\text{F}$  to  $10 \mu\text{F}$  range in parallel with a smaller ceramic capacitor ( $0.1 \mu\text{F}$ , for example). Note that these are simply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

## 10.2.2 Direct 1.71 V to 1.89 V Regulated Supply

In direct supply configuration,  $V_{CCD}$  and  $V_{DDD}$  are shorted together and connected to a 1.71-V to 1.89-V supply. This regulated supply should be connected to the device, as shown in [Figure 10-3](#).

In this mode,  $V_{CCD}$  and  $V_{DDD}$  pins are shorted together and bypassed.

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



## 10.3 How It Works

The regulators in [Figure 10-1](#) power the various domains of the device. All the core regulators draw their input power from the  $V_{DDD}$  pin supply. The analog circuits run directly from the  $V_{DDA}$  input.

### 10.3.1 Regulator Summary

#### 10.3.1.1 Active Digital Regulator

Table 10-1. Regulator Status in Different Power Modes

| Mode       | Active Digital Regulator | Deep-Sleep Regulator |
|------------|--------------------------|----------------------|
| Deep-Sleep | Off                      | On                   |
| Sleep      | On                       | On                   |
| Active     | On                       | On                   |

For external supplies from 1.8 V and 5.5 V, the Active digital regulator provides the main digital logic in Active and Sleep modes. This regulator has its output connected to a pin ( $V_{CCD}$ ) and requires an external decoupling capacitor (0.1  $\mu$ F X5R).

For supplies below 1.8 V,  $V_{CCD}$  must be supplied directly. In this case,  $V_{CCD}$  and  $V_{DDD}$  must be shorted together, as shown in [Figure 10-3](#). The Active digital regulator is available only in Active and Sleep power modes.

#### 10.3.1.2 Deep-Sleep Regulator

This regulator supplies the circuits that remain powered in Deep-Sleep mode, such as the ILO, WCO, and SCB ( $I^2C$ /SPI), and low-power comparator. The Deep-Sleep regulator is available in all power modes. In Active and Sleep power modes, the main output of this regulator is connected to the output of the Active digital regulator ( $V_{CCD}$ ).

## 10.4 Voltage Monitoring

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

### 10.4.1 Power-on-Reset (POR)

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

#### 10.4.1.1 Brownout-Detect (BOD)

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

To ensure reliable operation of the device, the watchdog timer should be used in all designs. Watchdog timer provides protection against abnormal brownout conditions that may compromise the CPU functionality. See [Watchdog Timer chapter on page 108](#) for more details.

## 10.5 Register List

Table 10-2. Power Supply and Monitoring Register List

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

# 11. Chip Operational Modes



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

## 11.1 Boot

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

## 11.2 User

User mode is an operational mode where normal user firmware from flash is executed. User mode cannot execute code from SROM. Firmware execution in this mode includes the automatically generated firmware by the ModusToolbox 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.

## 11.3 Privileged

Privileged mode is an operational mode, which allows execution of special subroutines that are stored in the device ROM. These subroutines cannot be modified 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 376](#). Exit from this mode returns the device to user mode.

## 11.4 Debug

Debug mode is an operational mode that allows observation of the PSoC 4 operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as ModusToolbox 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 366](#).

For more details on protection modes, see the [Device Security chapter on page 63](#).

# 12. Power Modes



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

- Active
- Sleep
- Deep-Sleep

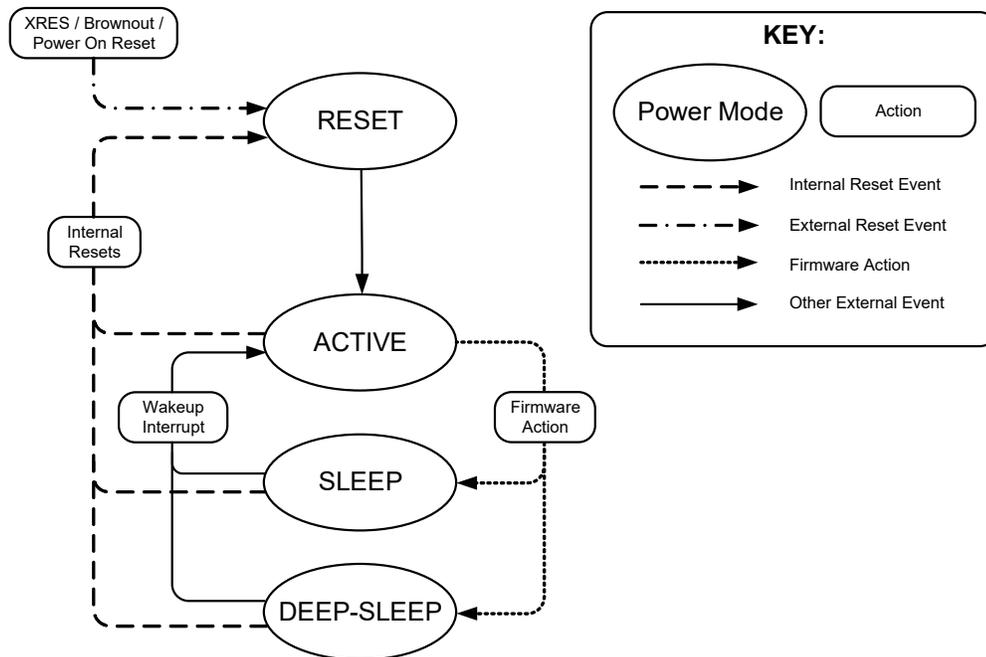
Active, Sleep, and Deep-Sleep are standard Arm-defined power modes, supported by the Arm CPUs.

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

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

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

Figure 12-1. Power Mode Transitions State Diagram



**Note:** Arm nomenclature for Deep-Sleep power mode is 'SLEEPDEEP'.

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

Table 12-1. PSoC 4 Power Modes

| Power Mode | Description  | Entry Condition   | Wakeup Sources  | Active Clocks                       | Wakeup Action | Available Regulators          |
|------------|--|---|---|-------------------------------------|---------------|-------------------------------|
| Active     | Primary mode of operation; all peripherals are available (programmable).   | Wakeup from other power modes, internal and external resets, brownout, power on reset | Not applicable  | All (programmable)                  | N/A           | All regulators are available. |
| Sleep      | CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable).  | Manual register write   | Any enabled interrupt                                     | All (programmable) except CPU clock | Interrupt     | All regulators are available. |
| Deep-Sleep | All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency clock is available.<br>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 (40 kHz), WCO (32 kHz)          | Interrupt     | Deep-Sleep regulator          |

In addition to the wakeup sources mentioned in Table 12-1, external reset (XRES) and brownout reset bring the device to Active mode from any power mode.

## 12.1 Active Mode

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

## 12.2 Sleep Mode

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

## 12.3 Deep-Sleep Mode

In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The high-frequency clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (40 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<sup>2</sup>C 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 12-3.

## 12.4 Power Mode Summary

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

Table 12-2. Available Peripherals

| Peripheral   | Active    | Sleep                  | Deep-Sleep           |
|--|-----------|------------------------|----------------------|
| CPU  | Available | Retention <sup>a</sup> | Retention            |
| SRAM   | Available | Retention              | Retention            |
| High-speed peripherals   | Available | Available              | Retention            |
| Low-speed peripherals  | Available | Available              | Available (optional) |
| Internal main oscillator (IMO)   | Available | Available              | Not Available        |
| Internal low-speed oscillator (ILO, 40 kHz)                              | Available | Available              | Available (optional) |
| Asynchronous peripherals (peripherals that do not run on internal clock) | Available | Available              | Available            |
| Power-on-reset, Brownout detection                                       | Available | Available              | Available            |
| Analog mux bus connection  | Available | Available              | Available            |
| GPIO output state  | Available | Available              | Available            |

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

Table 12-3. Wakeup Sources

| Power Mode | Wakeup Source                | Wakeup Action   |
|------------|------------------------------|-----------------|
| Sleep      | Any enabled interrupt source | Interrupt       |
|            | Any reset source             | Reset           |
| Deep-Sleep | GPIO interrupt               | Interrupt       |
|            | I2C address match            | Interrupt       |
|            | Watchdog timer               | Interrupt/Reset |
|            | Low-power comparator         | Interrupt       |
|            | CTBm                         | Interrupt       |

**Note:** In addition to the wakeup sources mentioned in Table 12-3, external reset (XRES) and brownout reset bring the device to Active mode from any power mode. XRES and brownout trigger 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.

## 12.5 Low-Power Mode Entry and Exit

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

The transition to Sleep and Deep-Sleep modes are controlled by the flags SLEEPDEEP in the CM0P System Control Register (CM0P\_SCR).

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

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

In Sleep and Deep-Sleep modes, a selection of peripherals are available (see [Table 12-3](#)), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode. See the [Interrupts chapter on page 52](#) and the [Reset System chapter on page 117](#) for details.

## 12.6 Register List

Table 12-4. Power Mode Register List

| Register Name | Description  |
|---------------|--|
| CM0P_SCR      | System Control - Sets or returns system control data.  |
| PWR_CONTROL   | Power Mode Control - Controls the device power mode options and allows observation of current state. |

# 13. Watchdog Timer



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

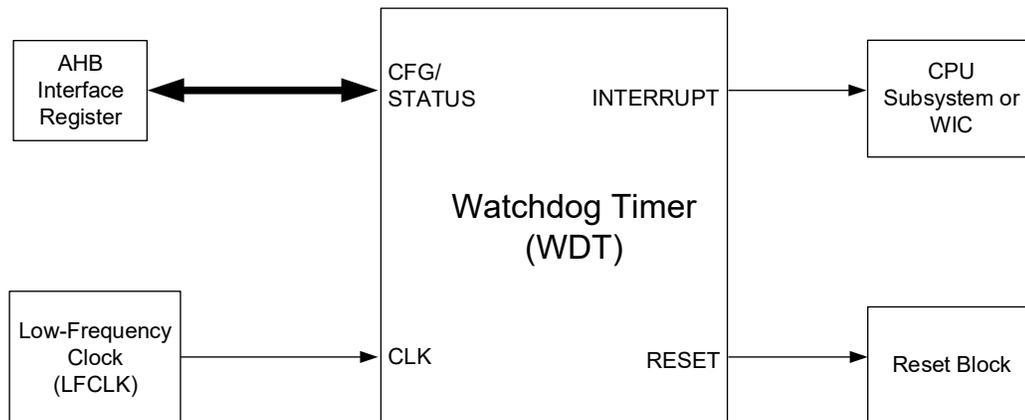
## 13.1 Features

The WDT has these features:

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

## 13.2 Block Diagram

Figure 13-1. Watchdog Timer Block Diagram



## 13.3 How It Works

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

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

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

The IGNORE\_BITS in the WDT\_MATCH register can be used to reduce the entire WDT counter period. The ignore bits can specify the number of MSBs that need to be discarded. For example, if the IGNORE\_BITS value is 3, then the WDT counter becomes a 13-bit counter. For details, see the WDT\_COUNTER, WDT\_MATCH, and SRSS\_INTR registers in the *PSoC 4100S Max: PSoC 4 Registers TRM*.

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

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

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

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

1. Write the desired IGNORE\_BITS in the WDT\_MATCH register to set the counter resolution.
2. Write the desired match value to the WDT\_MATCH register.
3. Clear the WDT\_MATCH bit in SRSS\_INTR to clear any pending WDT interrupt.
4. Enable the WDT interrupt by setting the WDT\_MATCH bit in SRSS\_INTR\_MASK
5. Enable global WDT interrupt in the CM0\_ISER register (see the [Interrupts chapter on page 52](#) for details).
6. In the ISR, clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.

For more details on interrupts, see the [Interrupts chapter on page 52](#).

### 13.3.1 Enabling and Disabling WDT

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

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

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

### 13.3.2 WDT Interrupts and Low-Power Modes

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

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

For more details on device power modes, see the [Power Modes chapter on page 104](#).

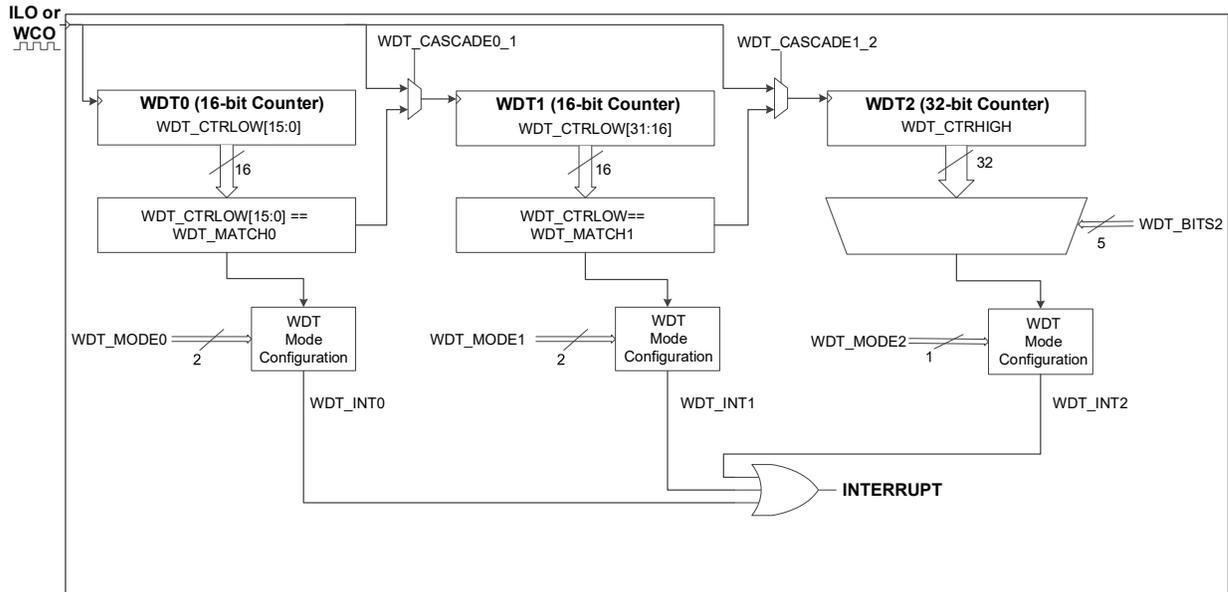
### 13.3.3 WDT Reset Mode

The RESET\_WDT bit in the RES\_CAUSE register indicates the reset generated by the WDT. This bit remains set until cleared or until a power-on reset (POR), brownout reset (BOD), or external reset (XRES) occurs. All other resets leave this bit untouched. For more details, see the [Reset System chapter on page 117](#).

## 13.4 Additional Timers

Besides WDT, there are three additional up-counting timers for general-purpose use – WDT0, WDT1, and WDT2. These three timers are clocked either from ILO or WCO, selected by writing into the WCO\_WDT\_CLKEN register. These timers can run in Active, Sleep, and Deep-Sleep modes and are capable of generating interrupts.

Figure 13-2. WDT Additional Timers Block Diagram



### 13.4.1 WDT0 and WDT1

These are 16-bit timers, which can be operated in two configurations:

- Free running
- Clear on match (configurable period)

In the free-running mode, the timer counts throughout the 16-bit range. On reaching 65535 ( $2^{16}-1$ ), the timer resets to 0 and starts counting again. In the Clear-on-match mode, the match count written in WDT\_MATCH0 and WDT\_MATCH1 of the WCO\_WDT\_MATCH register decides the period of WDT0 and WDT1, respectively. When the timer count reaches the match value, the timer resets to 0 and starts counting again. One of these two configurations is selected using WDT\_CLEAR0 and WDT\_CLEAR1 bits of the WCO\_WDT\_CONFIG register. The Clear-on-match mode is selected by writing '1' to WDT\_CLEARx. Writing '0' to this bit disables the clearing of timer on match count and the free-running mode is configured. Note that changing the match count requires three input clock cycles to come into effect. Before putting the device to deep sleep, ensure delay of at least one input clock cycle after the match count update.

An interrupt can be generated on match or timer overflow by writing into WDT\_MODE bits of the WCO\_WDT\_CONFIG register. On interrupt, the WDT\_INTx bit of the WCO\_WDT\_CONTROL register is set. This bit must be cleared by firmware to allow the next interrupt trigger. Note that the interrupts from all the three timers are ORed to generate a single trigger to the CPU. To identify which timer caused an interrupt, read the WDT\_INTx bit.

The timers are enabled by writing '1' to the WDT\_ENABLEx bit of the WCO\_WDT\_CONTROL register. Note that it takes three clock cycles to take effect. It is not recommended to toggle this bit more than once during this time. After enabling the timer, it is not recommended to write to the configuration register (WCO\_WDT\_CONFIG). The present value of the timers can be read from the WDT\_CTRL0W register; it can be reset by writing '1' to the WDT\_RESETx bit of the WCO\_WDT\_CONTROL register.

### 13.4.2 WDT2

It is similar to WDT0 and WDT1 with following differences:

- WDT2 is a 32-bit up-counting timer
- Supports only free-running configuration with counting range of 0 to ( $2^{32}-1$ )
- The interrupt is triggered when one out of 32 bits toggles during counting. The bit position is configured using the 5-bit WDT\_BITS2 field of the WCO\_WDT\_CONFIG register. Setting it to '0' results in an interrupt on every input clock; setting it to '1' results in an interrupt on alternate clocks; setting it to '31' results in an interrupt every  $2^{31}$  clocks.

### 13.4.3 Cascading

The cascading options are as follows:

- WDT0 and WDT1 timers can be cascaded by writing into WDT\_CASCADE0\_1 bit of the WCO\_WDT\_CONFIG register. When cascaded, WDT1 increments after WDT0 reaches its match count.
- WDT1 and WDT2 timers can also be cascaded by writing into WDT\_CASCADE1\_2 bit of the WCO\_WDT\_CONFIG register. When cascaded, WDT2 increments after WDT1 reaches its match count.
- All the three timers are cascaded when WDT\_CASCADE0\_1 and WDT\_CASCADE1\_2 bits are set.

## 13.5 Register List

Table 13-1 provides the register control details.

Table 13-1. WDT Registers

| Register Name   | Description  |
|-----------------|--|
| WDT_DISABLE_KEY | Disables the WDT when 0XACED8865 is written; for any other value WDT works normally.   |
| WDT_COUNTER     | Provides the count value of the WDT.   |
| WDT_MATCH       | Holds the match value of the WDT.  |
| SRSS_INTR       | Services the WDT to avoid reset.   |
| WCO_WDT_CTRLLOW | Stores the current WDT0 and WDT1 timer value.  |
| WCO_WDT_CTRHIGH | Stores the current WDT2 timer value.   |
| WCO_WDT_MATCH   | Holds the match count for WDT0 and WDT1.   |
| WCO_WDT_CONFIG  | Configures WDT0, WDT1, and WDT2 – selection of clock source, selection of free running or clear on match, interrupt generation, and cascading. |
| WCO_WDT_CONTROL | Used for enabling and resetting the timer.   |
| WCO_WDT_CLKEN   | Enables the clock (ILO/WCO) to be used with the timer.   |

# 14. Trigger Multiplexer Block



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

## 14.1 Features

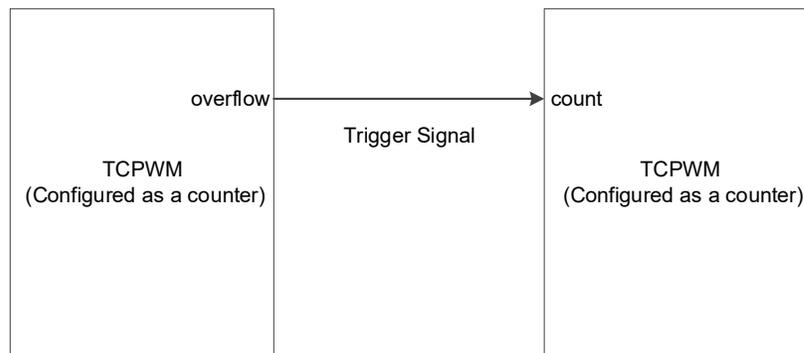
The Trigger Multiplex Block has these features:

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

## 14.2 Architecture

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

Figure 14-1. Trigger Signal Example



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

### 14.2.1 Trigger Multiplexer Group

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

**Note** The triggers output into different peripherals, which may have more routing than is shown on the trigger routing diagram. For more information on this routing, go to the trigger destination peripheral block.

Figure 14-2. Trigger Multiplexer Groups

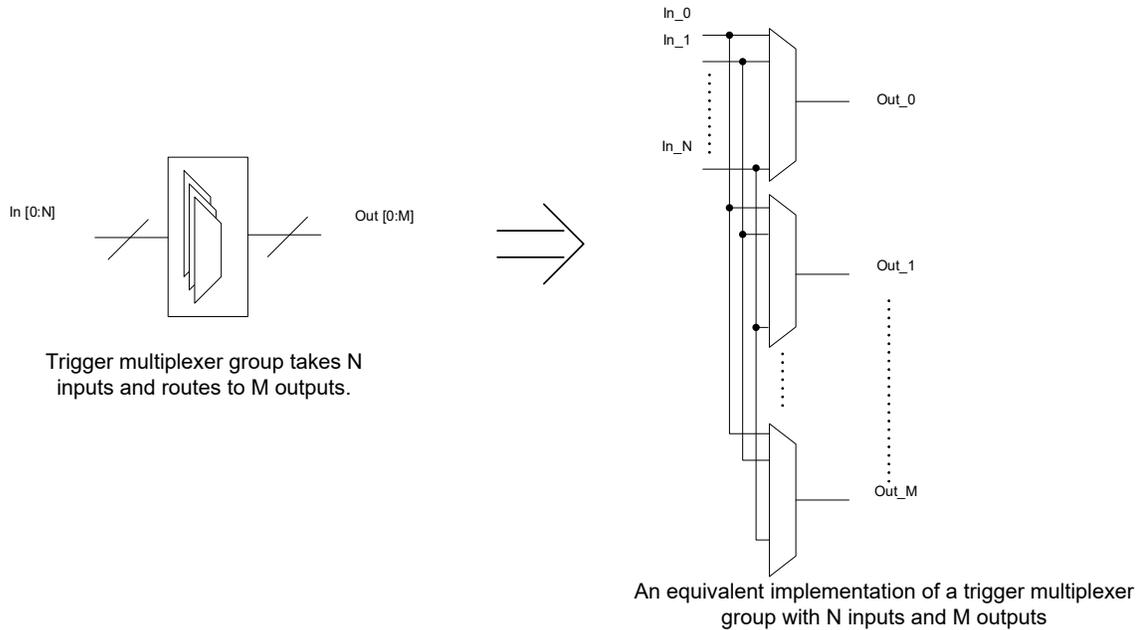
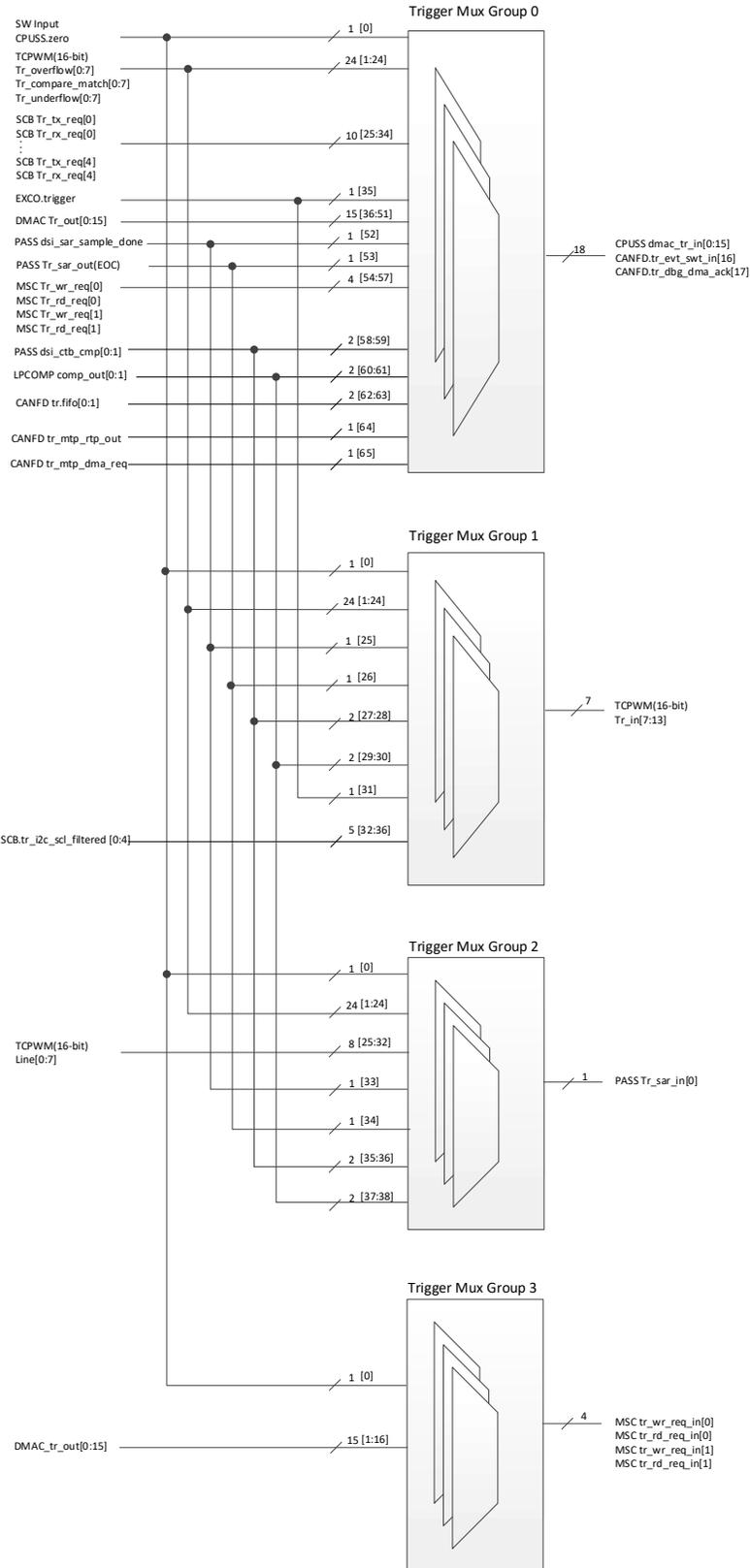


Figure 14-3. PSoC 4100S Max Trigger Multiplexer Block Architecture



## 14.2.2 Software Triggers

All input and output signals to a trigger multiplexer can be triggered from software. This is accomplished by writing into the PERI\_TR\_CTL register. This register allows you to trigger the corresponding signal for a number of peripheral clock cycles. The PERI\_TR\_CTL[TR\_GROUP] bitfield selects the trigger group of the signal being activated. The PERI\_TR\_CTL[TR\_OUT] bitfield determines whether the trigger signal is in output or input of the multiplexer. PERI\_TR\_CTL[TR\_SEL] selects the specific line in the trigger group. The PERI\_TR\_CTL[TR\_COUNT] bitfield sets up the number of peripheral clocks the trigger will be activated. The PERI\_TR\_CTL[TR\_ACT] bitfield is set to '1' to activate the trigger line specified. Hardware resets this bit after the trigger is deactivated after the number of cycles set by the PERI\_TR\_CTL[TR\_COUNT].

## 14.3 Register List

Table 14-1. Register List

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

# 15. Reset System



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

## 15.1 Reset Sources

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

### 15.1.1 Power-on Reset

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

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

### 15.1.2 Brownout 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.

### 15.1.3 Watchdog Reset

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

The RESET\_WDT status bit of the RES\_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR, XRES, or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched. For more details, see the [Watchdog Timer chapter on page 108](#).

### 15.1.4 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 (CM0P\_AIRCR) forces a device reset when a '1' is written into the SYSRESETREQ bit. CM0P\_AIRCR requires a value of 05FA written to the top two bytes for writes. Therefore, write 05FA0004 for the reset.

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

### 15.1.5 External Reset

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

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

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

### 15.1.6 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 103](#).

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

## 15.2 Identifying Reset Sources

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

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

## 15.3 Register List

Table 15-1. Reset System Register List

| Register Name   | Description   |
|-----------------|---|
| WDT_DISABLE_KEY | Disables the WDT when 0XACED8865 is written, for any other value WDT works normally   |
| CM0P_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.   |

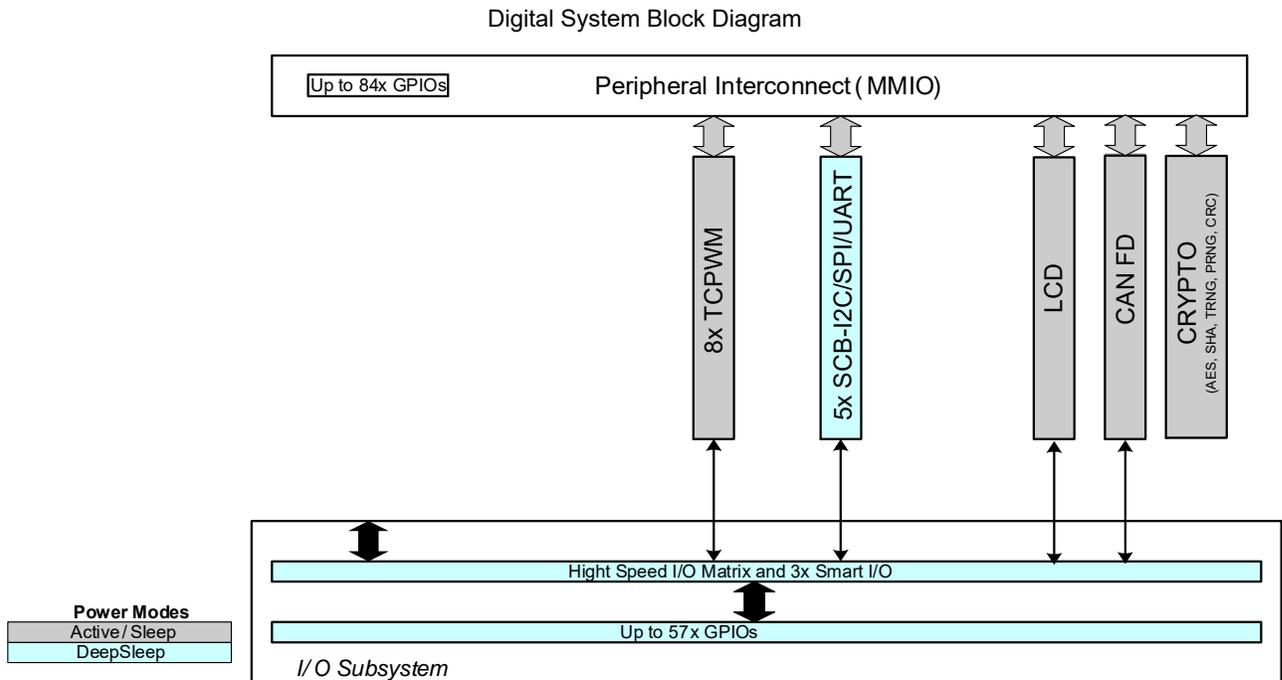
# Section D: Digital System



This section encompasses the following chapters:

- Serial Communications Block (SCB) chapter on page 86
- CAN FD Controller chapter on page 179
- Timer, Counter, and PWM chapter on page 190
- True Random Number Generator chapter on page 281
- LCD Direct Drive chapter on page 285
- Inter-IC Sound Bus chapter on page 294
- CRYPTO chapter on page 301

## Top Level Architecture



# 16. Serial Communications Block (SCB)



The Serial Communications Block (SCB) supports three serial communication protocols: Serial Peripheral Interface (SPI), Universal Asynchronous Receiver Transmitter (UART), and Inter Integrated Circuit (I<sup>2</sup>C or IIC). Only one of the protocols is supported by an SCB at any given time. The maximum number of SCBs in the PSoC 4 MCU devices varies by part number. Refer to the [device datasheet](#) to determine the number of SCBs and the SCB pin locations.

## 16.1 Features

The SCB supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, Local Interconnect Network (LIN), and IrDA protocols
  - Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
- Standard I<sup>2</sup>C master and slave functionality
- Trigger outputs for connection to DMA
- Multiple interrupt sources to indicate status of FIFOs and transfers
- SCBs are Deep Sleep capable and allow for operation without CPU intervention.
  - Deep Sleep only works for EZ mode for SPI and I2C configured as a slave.
  - During Deep Sleep the SCB clock is not available so an external clock must be used.
  - Deep Sleep wakeup when I2C or SPI are configured in FIFO mode.

## 16.2 Architecture

The operation modes supported by SCB are described in the following sections.

### 16.2.1 Buffer Modes

Each SCB has 32 bytes of dedicated RAM for transmit and receive operation. The RAM can be configured in two different modes (FIFO and EZ). The following sections give a high-level overview of each mode. The sections on each protocol will provide more details.

- Masters can only use FIFO mode
- I<sup>2</sup>C and SPI slaves can use both FIFO and EZ modes
- UART only uses FIFO mode

#### 16.2.1.1 FIFO Mode

In this mode the RAM is split into two 16-byte FIFOs, one for transmit (TX) and one for receive (RX). The FIFOs can be configured to be 8 bits × 16 elements or 16 bits × 8 elements; this is done by setting the BYTE\_MODE bit in the SCB control register.

FIFO mode of operation is available only in Active and Sleep power modes. However, the I<sup>2</sup>C address or SPI slave select can be used to wake the device from Deep Sleep.

Statuses are provided for both the RX and TX FIFOs. There are multiple interrupt sources available, which indicate the status of the FIFOs, such as full or empty; see [“SCB Interrupts” on page 169](#).

### 16.2.1.2 EZ Mode

In easy (EZ) mode the RAM is used as a single 32-byte buffer. The external master sets a base address and reads and writes start from that base address.

EZ Mode is available only for SPI slave and I<sup>2</sup>C slave.

EZ mode is available in Active, Sleep, and Deep Sleep power modes.

**Note:** This document discusses hardware implementation of the EZ mode.

### 16.2.2 Clocking Modes

The SCB can be clocked either by an internal clock provided by the peripheral clock dividers (referred to as `clk_scb` in this document), or it can be clocked by the external master.

- UART, SPI master, and I<sup>2</sup>C master modes must use `clk_scb`.
- Only SPI slave and I<sup>2</sup>C slave can use the clock from and external master.

Internally- and externally-clocked slave functionality is determined by two register fields of the SCB CTRL register:

- `EC_AM_MODE` indicates whether SPI slave selection or I<sup>2</sup>C address matching is internally ('0') or externally ('1') clocked.
- `EC_OP_MODE` indicates whether the rest of the protocol operation (besides SPI slave selection and I<sup>2</sup>C address matching) is internally ('0') or externally ('1') clocked.

**Notes:**

- FIFO mode supports an internally- or externally-clocked address match (`EC_AM_MODE` is '0' or '1'); however, data transfer must be done with internal clocking. (`EC_OP_MODE` is '1').
- EZ mode is supported with externally clocked operation (`EC_OP_MODE` is '1').

Table 16-1 provides an overview of the clocking and buffer modes supported for each communication mode.

Table 16-1. Clock Mode Compatibility

|                               | Internally clocked (IC) |     | Externally clocked (EC)<br>(Deep Sleep SCB only) |     |
|-------------------------------|-------------------------|-----|--|-----|
|                               | FIFO                    | EZ  | FIFO   | EZ  |
| I <sup>2</sup> C master       | Yes                     | No  | No   | No  |
| I <sup>2</sup> C slave        | Yes                     | Yes | No <sup>a</sup>                                  | Yes |
| I <sup>2</sup> C master-slave | Yes                     | No  | No   | No  |
| SPI master                    | Yes                     | No  | No   | No  |
| SPI slave                     | Yes                     | Yes | No <sup>b</sup>                                  | Yes |
| UART transmitter              | Yes                     | No  | No   | No  |
| UART receiver                 | Yes                     | No  | No   | No  |

a. In Deep Sleep mode the external-clocked logic can handle slave address matching, it then triggers an interrupt to wake up the CPU. The slave can be programmed to stretch the clock, or NACK until internal logic takes over.

b. In Deep Sleep mode the external-clocked logic can handle slave selection detection, it then triggers an interrupt to wake up the CPU. Writes will be ignored and reads will return 0xFF until internal logic takes over.

Table 16-2. Clock Configuration and Mode Support

| Mode      | <code>EC_AM_MODE</code> is '0';<br><code>EC_OP_MODE</code> is '0' | <code>EC_AM_MODE</code> is '1';<br><code>EC_OP_MODE</code> is '0' | <code>EC_AM_MODE</code> is '1';<br><code>EC_OP_MODE</code> is '1' |
|-----------|---|---|---|
| FIFO mode | Yes   | Yes   | No  |
| EZ mode   | Yes   | Yes   | Yes   |

## 16.3 Serial Peripheral Interface (SPI)

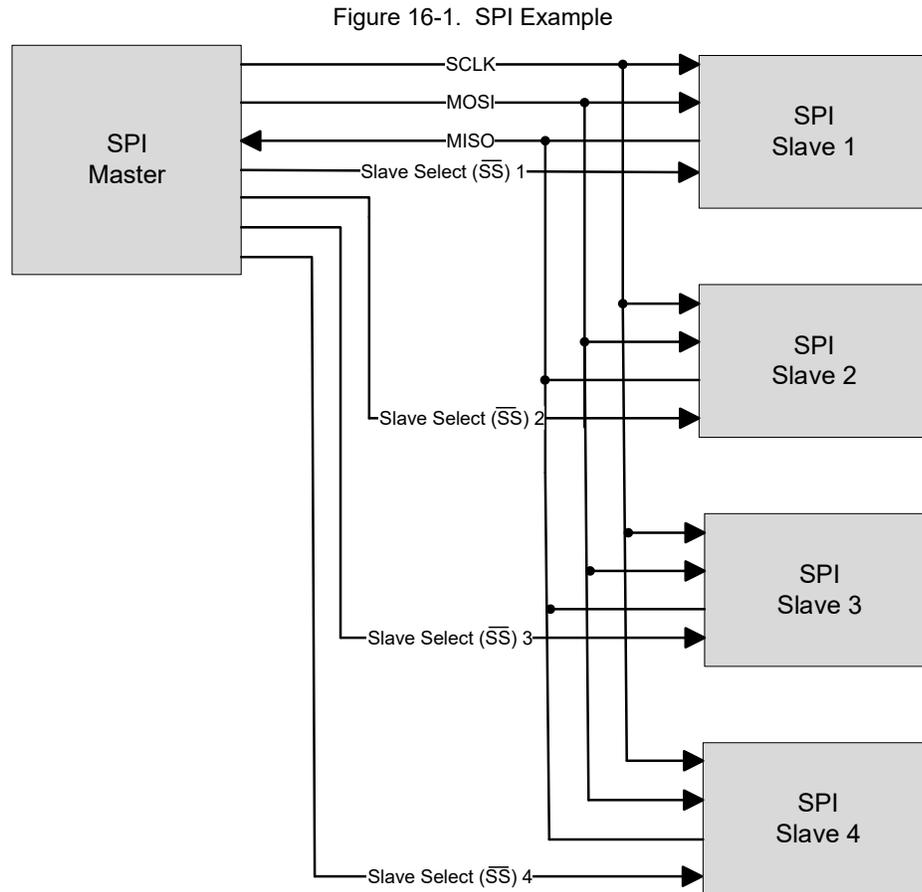
The SPI protocol is a synchronous serial interface protocol. Devices operate in either master or slave mode. The master initiates the data transfer. The SCB supports single-master-multiple-slaves topology for SPI. Multiple slaves are supported with individual slave select lines.

### 16.3.1 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 – mode 1 only
  - National Semiconductor (MicroWire) SPI – mode 0 only
- Master supports up to four slave select lines
  - Each slave select has configurable active polarity (high or low)
  - Slave select can be programmed to stay active for a whole transfer, or just for each byte
- Master supports late sampling for better timing margin
- Master supports continuous SPI clock
- Data frame size programmable from 4 bits to 16 bits
- Programmable oversampling
- MSb or LSb first
- Median filter available for inputs
- Supports FIFO Mode
- Supports EZ Mode (slave only)

### 16.3.2 General Description

Figure 16-1 illustrates an example of SPI master with four slaves.



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.

## 16.3.3 SPI Modes of Operation

### 16.3.3.1 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'. When not transmitting data, the  $\overline{SS}$  line is held at '1'.

#### Clock Modes of Motorola SPI

The Motorola SPI protocol has four different clock 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 16-2 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

Figure 16-2. SPI Motorola, 4 Modes

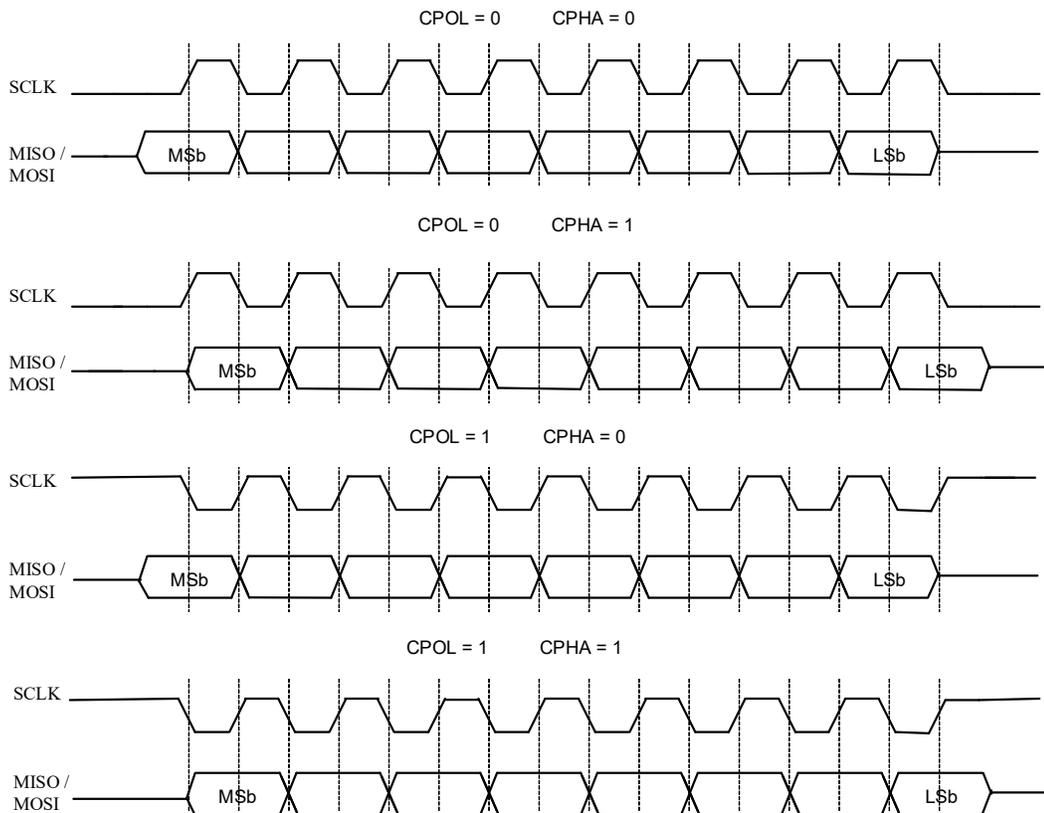


Figure 16-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 16-3. SPI Motorola Data Transfer Example

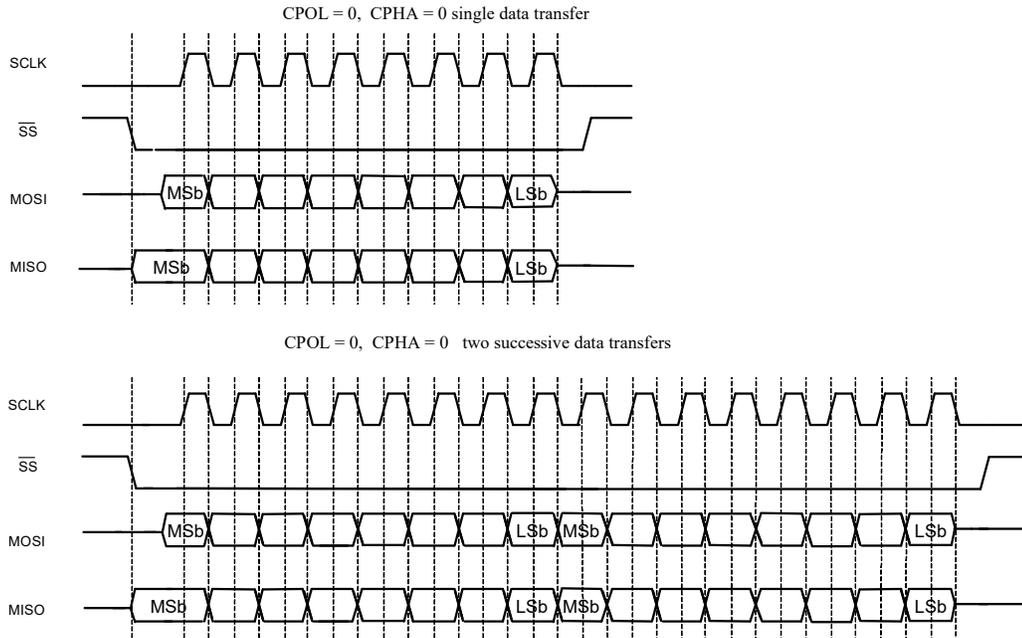
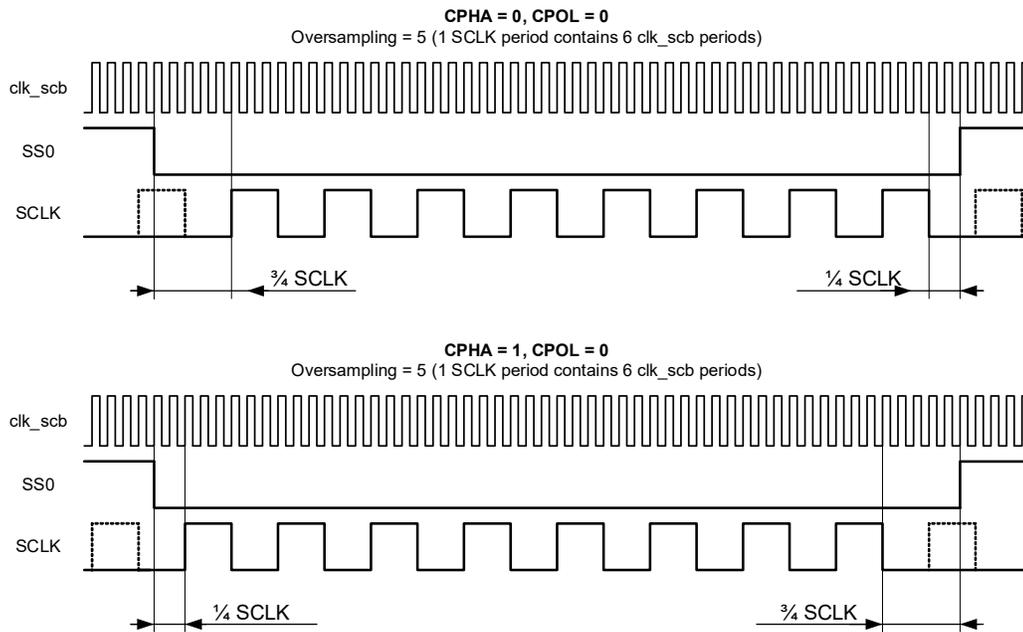


Figure 16-4. SELECT and SCLK Timing Correlation



For example above: OversamplingReg = 6 - 1 = 5.  
 $\frac{3}{4} * SCLK = ((5 / 2) + 1) + (5 / 4 + 1) * clk\_scb = (3 + 2) * clk\_scb = 5 * clk\_scb.$   
 $\frac{1}{4} * SCLK = ((5 / 4) + 1) * clk\_scb = 2 * clk\_scb.$

**Note** The value  $\frac{3}{4} * SCLK$  is equal to  $((OversamplingReg / 2) + 1) + (OversamplingReg / 4) + 1$ ), where OversamplingReg = Oversampling - 1.  
 The value  $\frac{1}{4} * SCLK$  is equal to  $((OversamplingReg / 4) + 1)$ .  
 The result of any division operation is rounded down to the nearest integer.

**Note** The provided timings are guaranteed by SCB block but do not take into account signal propagation time from SCB block to pins.

### 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 clock mode 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 135](#).

For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

#### 16.3.3.2 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. 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 16-5 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 16-5. SPI TI Data Transfer Example

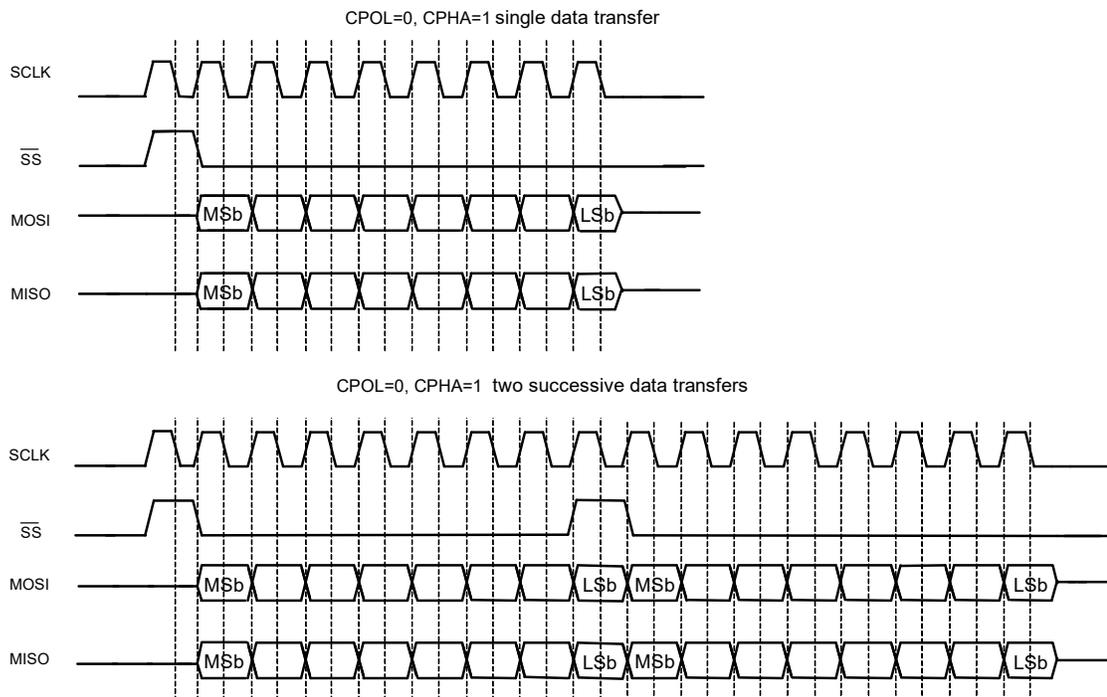
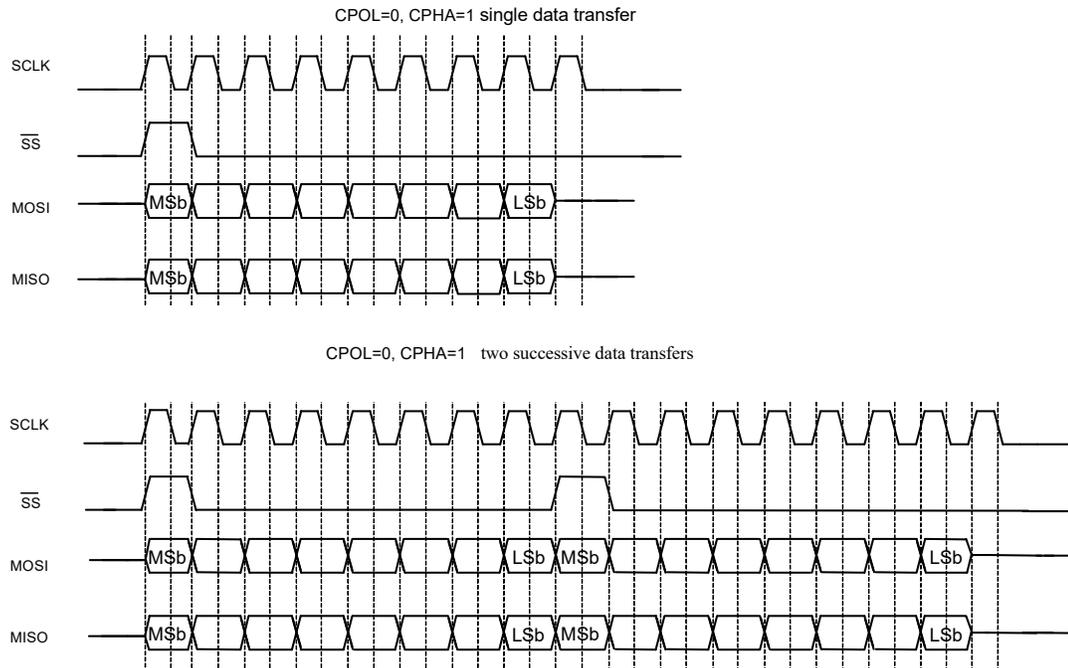


Figure 16-6 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

Figure 16-6. 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. Set the CPHA (bit 2) of the SCB\_SPI\_CONTROL register to '0', and the CPOL (bit 3) of the same register to '1'.
5. Follow steps 2 to 4 mentioned in "Enabling and Initializing SPI" on page 135.

For more information on these registers, see the *PSoC 4100S Max: PSoc 4 Registers TRM*.

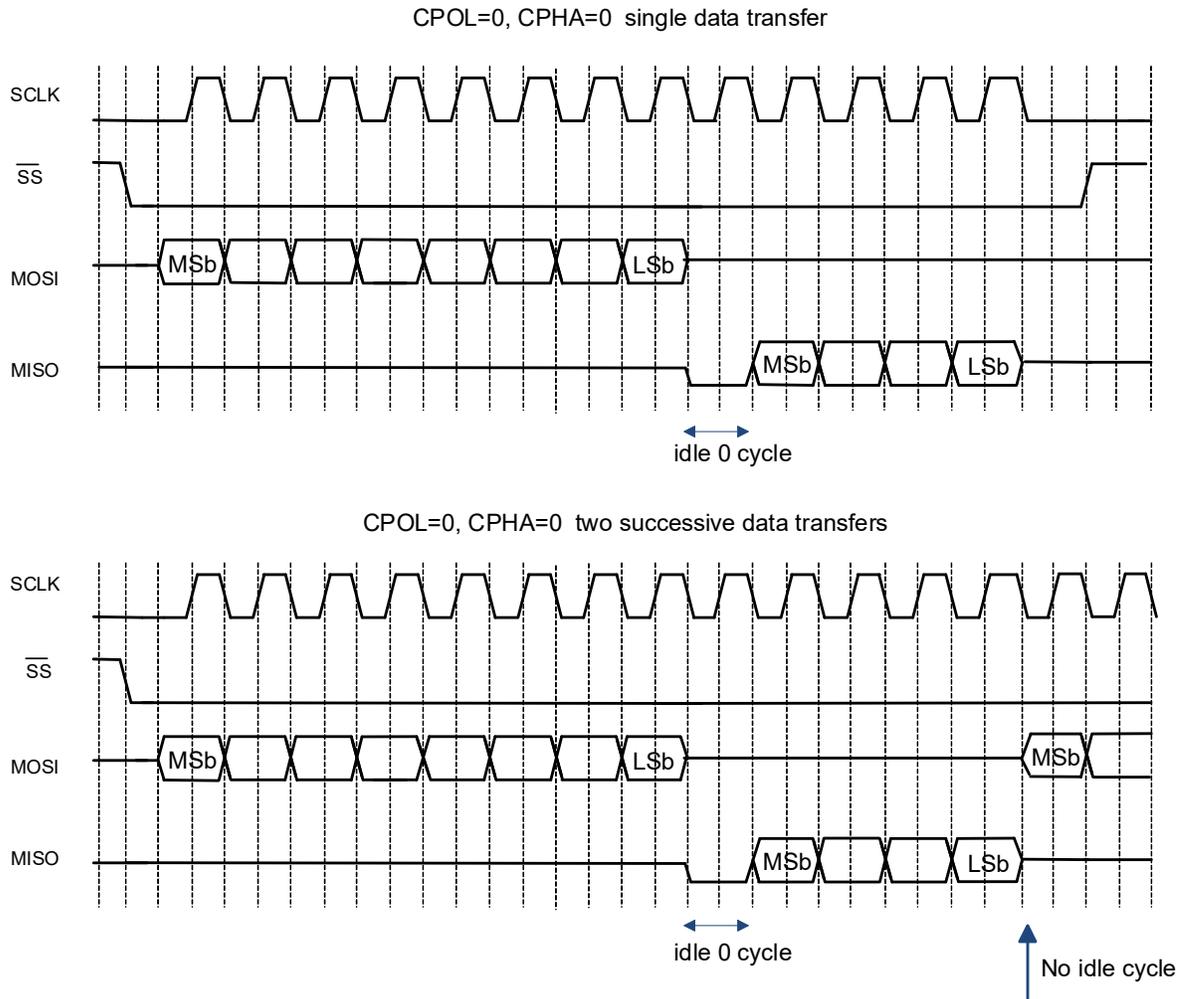
### 16.3.3.3 National Semiconductors SPI

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 transfers from reception. However, successive data transfers are not separated by an idle bit transfer period.

The National Semiconductors SPI protocol only supports CPOL/CPHA mode 0.

Figure 16-7 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 16-7. 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. Set the CPOL and CPHA bits of the SCB\_SPI\_CTRL register to '0'.
4. Follow steps 2 to 4 mentioned in ["Enabling and Initializing SPI" on page 135](#).

For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

## 16.3.4 SPI Buffer Modes

SPI can operate in two different buffer modes – FIFO and EZ modes. The buffer is used in different ways in each of these modes. The following subsections explain each of these buffer modes in detail.

### 16.3.4.1 FIFO Mode

The FIFO mode has a TX FIFO for the data being transmitted and an RX FIFO for the data received. Each FIFO is constructed out of the SRAM buffer. The FIFOs are either 8 elements deep with 16-bit data elements or 16 elements deep with 8-bit data elements. The width of a FIFO is configured using the `BYTE_MODE` bitfield of the `SCB.CTRL` register.

FIFO mode of operation is available only in Active and Sleep power modes, and not in the Deep Sleep mode.

Transmit and receive FIFOs allow write and read accesses. A write access to the transmit FIFO uses the `TX_FIFO_WR` register. A read access from the receive FIFO uses the `RX_FIFO_RD` register. For SPI master mode, data transfers are started when data is written into the TX FIFO. Note that when a master is transmitting and the FIFO becomes empty the slave is de-selected.

Transmit and receive FIFO status information is available through status registers, `TX_FIFO_STATUS` and `RX_FIFO_STATUS`, and through the `INTR_TX` and `INTR_RX` registers.

Each FIFO has a trigger output. This trigger output can be routed through the trigger mux to various other peripheral on the device such as DMA or TCPWMs. The trigger output of the SCB is controlled through the `TRIGGER_LEVEL` field in the `RX_CTRL` and `TX_CTRL` registers.

- For a TX FIFO a trigger is generated when the number of entries in the transmit FIFO is less than `TX_FIFO_CTRL.TRIGGER_LEVEL`.
- For the RX FIFO a trigger is generated when the number of entries in the FIFO is greater than the `RX_FIFO_CTRL.TRIGGER_LEVEL`.

Note that the DMA has a trigger deactivation setting. For the SCB this should be set to 16.

### Active to Deep Sleep Transition

Before going to deep sleep ensure that all active communication is complete. For a master this can easily be done by checking the `SPI_DONE` bit in the `INTR_M` register, and ensuring the TX FIFO is empty.

For a slave this can be achieved by checking the `BUS_BUSY` bit in the SPI Status register. Also the RX FIFO should be empty before going to deep sleep. Any data in the FIFO will be lost during deep sleep.

Also before going to deep sleep the clock to the SCB needs to be disabled. This is done automatically by ModusToolbox Software with the provided API. This can be done manually by disabling the clock divider for the SCB clock. For more information on clock dividers, consult the [Clocking System chapter on page 88](#).

Lastly, when the device goes to deep sleep the SCB stops driving the GPIO lines. This leads to floating pins and can lead to undesirable current during deep sleep power modes. To avoid this condition before entering deep sleep mode change the HSIOM settings of the SCB pins to GPIO driven, then change the drive mode and drive state to the appropriate state to avoid floating pins. Consult the [I/O System chapter on page 66](#) for more information on pin drive modes.

### Deep Sleep to Active Transition

`EC_AM = 1`, `EC_OP = 0`, FIFO Mode.

When the SPI Slave Select line is asserted the device will be awoken by an interrupt. After the device is awoken change the SPI pin drive modes and HSIOM settings back to what they were before deep sleep. When `clk_hf[0]` is at the desired frequency, enable the clock to the SCB, this is done by enabling the clock divider. See [Clocking System chapter on page 88](#) for more information. At this point, the master can read valid data from the slave. Before that any data read by the master will be invalid.

### 16.3.4.2 EZSPI Mode

The easy SPI (EZSPI) protocol only works in the Motorola mode, with any of the clock modes. It allows communication between master and slave without the need for CPU intervention.

The EZSPI protocol defines a single memory buffer with an 5-bit EZ address that indexes the buffer (32-entry array of eight bit per entry) located on the slave device. The EZ address is used to address these 32 locations. All EZSPI data transfers have 8-bit data frames.

The CPU writes and reads to the memory buffer through the SCB\_EZ\_DATA registers. These accesses are word accesses, but only the least significant byte of the word is used.

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.

**Note:** When multiple bytes are read or written the master must keep SSEL low during the entire transfer.

#### **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 acknowledged (0xFE) or not (0xFF). The second byte on the MOSI line is the EZ address.

#### **Memory Array Write**

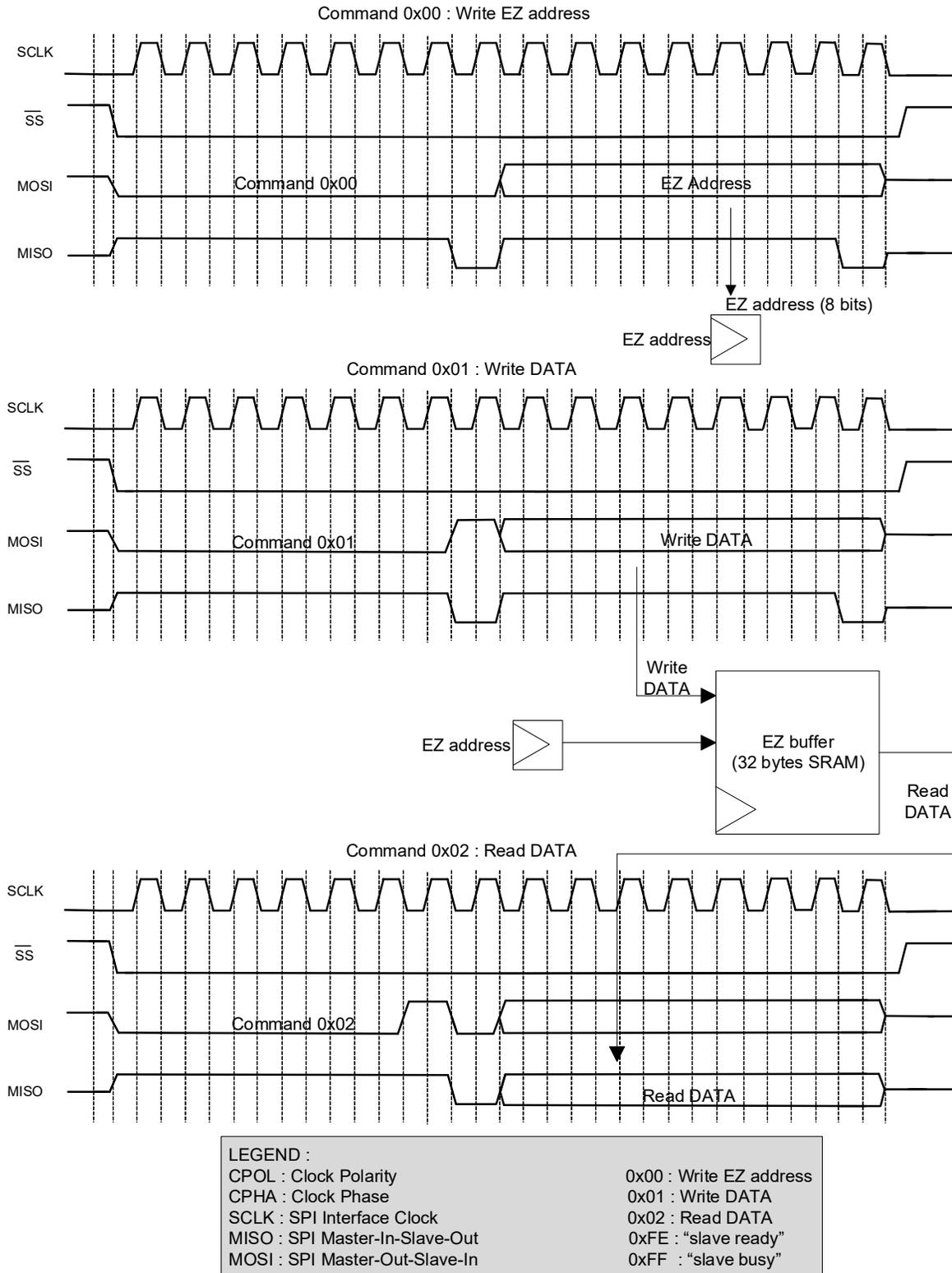
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was registered (0xFE) or not (0xFF). Any additional bytes on the MOSI line are 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. The EZ base address is reset to the address written in the EZ Address Write phase on each slave selection.

#### **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. The EZ base address is reset to the address written in the EZ Address Write phase on each slave selection.

Figure 16-8 illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.

Figure 16-8. EZSPI Example



## 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. Set the EC\_AM and EC\_OP modes in the SCB\_CTRL register as appropriate.
3. Set the BYTE\_MODE bit of the SCB\_CTRL register to '1'.
4. Follow the steps in [“Configuring SCB for SPI Motorola Mode” on page 126](#).
5. Follow steps 2 to 4 mentioned in [“Enabling and Initializing SPI” on page 135](#).

For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

## Active to Deep Sleep Transition

Before going to deep sleep ensure the master is not currently transmitting to the slave. This can be done by checking the BUS\_BUSY bit in the SPI\_STATUS register.

If the bus is not busy, disable the clock to the SCB by disabling the clock divider for the clock going to the SCB, see the [Clocking System chapter on page 88](#).

## Deep Sleep to Active Transition

- **EC\_AM = 1, EC\_OP = 0, EZ Mode.** MISO transmits 0xFF until the internal clock is enabled. Data on MOSI is ignored until the internal clock is enabled. Do not enable the internal clock until clk\_hf[0] is at the desired frequency. After clk\_hf[0] is at the desired frequency, enable the clock to the SCB, this is done by enabling the clock divider. See the [Clocking System chapter on page 88](#) for more information. The external master needs to be aware that when it reads 0xFF on MISO the device is not ready yet.
- **EC\_AM = 1, EC\_OP = 1, EZ Mode.** Do not enable the internal clock until clk\_hf[0] is at the desired frequency. After clk\_hf[0] is at the desired frequency, enable the clock to the SCB, this is done by enabling the clock divider. See the [Clocking System chapter on page 88](#) for more information.

## 16.3.5 Clocking and Oversampling

### 16.3.5.1 Clock Modes

The SCB SPI supports both internally and externally clocked operation modes. Two bitfields (EC\_AM\_MODE and EC\_OP\_MODE) in the SCB\_CTRL register determine the SCB clock mode. EC\_AM\_MODE indicates whether SPI slave selection is internally (0) or externally (1) clocked. EC\_OP\_MODE indicates whether the rest of the protocol operation (besides SPI slave selection) is internally (0) or externally (1) clocked.

An externally-clocked operation uses a clock provided by the external master (SPI SCLK).

An internally-clocked operation uses the programmable clock dividers. For SPI, an integer clock divider must be used for both master and slave. For more information on system clocking, see the [Clocking System chapter on page 88](#).

The SCB\_CTRL bitfields EC\_AM\_MODE and EC\_OP\_MODE can be configured in the following ways.

- EC\_AM\_MODE is '0' and EC\_OP\_MODE is '0': Use this configuration when only Active mode functionality is required.
  - FIFO mode: Supported.
  - EZ mode: Supported.
- EC\_AM\_MODE is '1' and EC\_OP\_MODE is '0': Use this configuration when both Active and Deep Sleep functionality are required. This configuration relies on the externally-clocked functionality to detect the slave selection and relies on the internally-clocked functionality to access the memory buffer.

The “hand over” from external to internal functionality relies on a busy/ready byte scheme. This scheme relies on the master to retry the current transfer when it receives a busy byte and requires the master to support busy/ready byte interpretation. When the slave is selected, INTR\_SPI\_EC.WAKE\_UP is set to '1'. The associated Deep Sleep functionality interrupt brings the system into Active power mode.

- FIFO mode: Supported. The slave (MISO) transmits 0xFF until the CPU is awoken and the TX FIFO is populated. Any data on the MOSI line will be dropped until `clk_scb` is enabled see “Deep Sleep to Active Transition” on page 129 for more details.
- EZ mode: Supported. In Deep Sleep power mode, the slave (MISO) transmits a busy (0xFF) byte during the reception of the command byte. In Active power mode, the slave (MISO) transmits a ready (0xFE) byte during the reception of the command byte.
- EC\_AM\_MODE is ‘1’ and EC\_OP\_MODE is ‘1’. Use this mode when both Active and Deep Sleep functionality are required. When the slave is selected, `INTR_SPI_EC.WAKE_UP` is set to ‘1’. The associated Deep Sleep functionality interrupt brings the system into Active power mode. When the slave is deselected, `INTR_SPI_EC.EZ_STOP` and/or `INTR_SPI_EC.EZ_WRITE_STOP` are set to ‘1’.
  - FIFO mode: Not supported.
  - EZ mode: Supported.

If `EC_OP_MODE` is ‘1’, the external interface logic accesses the memory buffer on the external interface clock (SPI SCLK). This allows for EZ functionality in Active and Deep Sleep power modes.

In Active system power mode, the memory buffer requires arbitration between external interface logic (on SPI SCLK) and the CPU interface logic (on system peripheral clock). This arbitration always gives the highest priority to the external interface logic (host accesses). The external interface logic takes two serial interface clock/bit periods for SPI. During this period, the internal logic is denied service to the memory buffer. The PSoC 4 MCU provides two programmable options to address this “denial of service”:

- If the `BLOCK` bitfield of `SCB_CTRL` is ‘1’: An internal logic access to the memory buffer is blocked until the memory buffer is granted and the external interface logic has completed access. This option provides normal SCB register functionality, but the blocking time introduces additional internal bus wait states.
- If the `BLOCK` bitfield of `SCB_CTRL` is ‘0’: An internal logic access to the memory buffer is not blocked, but fails when it conflicts with an external interface logic access. A read access returns the value 0xFFFF:FFFF and a write access is ignored. This option does not introduce additional internal bus wait states, but an access to the memory buffer may not take effect. In this case, the following failures are detected:
  - Read Failure: A read failure is easily detected because the returned value is 0xFFFF:FFFF. This value is unique as non-failing memory buffer read accesses return an unsigned byte value in the range 0x0000:0000-0x0000:00ff.
  - Write Failure: A write failure is detected by reading back the written memory buffer location, and confirming that the read value is the same as the written value.

For both options, a conflicting internal logic access to the memory buffer sets `INTR_TX.BLOCKED` field to ‘1’ (for write accesses) and `INTR_RX.BLOCKED` field to ‘1’ (for read accesses). These fields can be used as either status fields or as interrupt cause fields (when their associated mask fields are enabled).

If a series of read or write accesses is performed and `CTRL.BLOCKED` is ‘0’, a failure is detected by comparing the “logical-or” of all read values to 0xFFFF:FFFF and checking the `INTR_TX.BLOCKED` and `INTR_RX.BLOCKED` fields to determine whether a failure occurred for a series of write or read operations.

Table 16-3. SPI Modes Compatibility

|            | Internally Clocked (IC) |     | Externally Clocked (EC) |     |
|------------|-------------------------|-----|-------------------------|-----|
|            | FIFO                    | EZ  | FIFO                    | EZ  |
| SPI master | Yes                     | No  | No                      | No  |
| SPI slave  | Yes                     | Yes | Yes <sup>a</sup>        | Yes |

a. In SPI slave FIFO mode, the external-clocked logic does selection detection, then triggers an interrupt to wake up the CPU. Writes will be ignored and reads will return 0xFF until the CPU is ready and the FIFO is populated.

### 16.3.5.2 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 while 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.

### 16.3.5.3 Oversampling and Bit Rate

#### SPI Master Mode

The SPI master does not support externally clocked mode. In internally clocked mode, the logic operates under internal clock. The internal clock has higher frequency than the interface clock (SCLK), such that the master can oversample its input signals (MISO).

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. Therefore, 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'.

$$\text{Bit Rate} = \text{clk\_scb}/\text{OVS}$$

**Equation 16-1**

The numbers above indicate how fast the SCB hardware can run SCLK. It does not indicate that the master will be able to correctly receive data from a slave at those speeds. To determine that, the path delay of MISO must be calculated. It can be calculated using [Equation 16-2](#).

$$\frac{1}{2}t_{SCLK} \geq t_{SCLK\_PCB\_D} + t_{DSO} + t_{SCLK\_PCB\_D} + t_{DSI}$$

**Equation 16-2**

Where:

$t_{SCLK}$  is the period of the SPI clock

$t_{SCLK\_PCB\_D}$  is the SCLK PCB delay from master to slave

$t_{DSO}$  is the total internal slave delay, time from SCLK edge at slave pin to MISO edge at slave pin

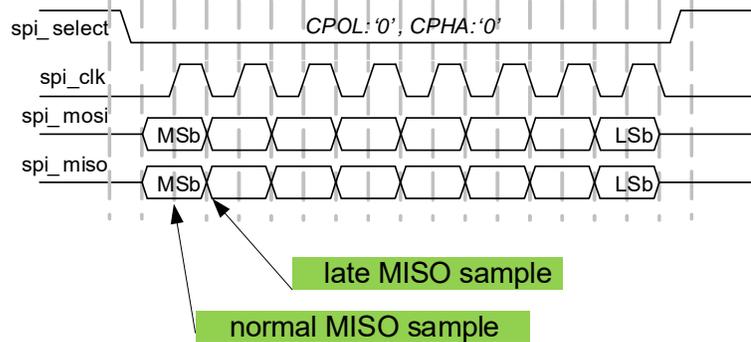
$t_{SCLK\_PCB\_D}$  is the MISO PCB delay from slave to master

$t_{DSI}$  is the master setup time

Most slave datasheets will list  $t_{DSO}$ . It may have a different name; look for MISO output valid after SCLK edge. Most master datasheets will also list  $t_{DSI}$ , or master setup time.  $t_{SCLK\_PCB\_D}$  and  $t_{SCLK\_PCB\_D}$  must be calculated based on specific PCB geometries.

If after doing these calculations the desired speed cannot be achieved, then consider using the MISO late sample feature of the SCB. This can be done by setting the SPI\_CTRL.LATE\_MISO\_SAMPLE register. MISO late sample tells the SCB to sample the incoming MISO signal on the next edge of SCLK, thus allowing for  $\frac{1}{2}$  SCLK cycle more timing margin, see [Figure 16-9](#).

Figure 16-9. MISO Sampling Timing



This changes the equation to:

$$t_{SCLK} \geq t_{SCLK\_PCB\_D} + t_{DSO} + t_{SCLK\_PCB\_D} + t_{DSI} \quad \text{Equation 16-3}$$

Because late sample allows for better timing, leave it enabled all the time. The  $t_{DSI}$  specification in the PSoC 4 MCU datasheet assumes late sample is enabled.

**Note:** The SPI\_CTRL.LATE\_MISO\_SAMPLE is set to '0' by default.

### SPI Slave Mode

In SPI slave mode, the OVS field (bits [3:0]) of SCB\_CTRL register is not used. The data rate is determined by Equation 16-2 and Equation 16-3. Late MISO sample is determined by the external master in this case, not by SPI\_CTRL.LATE\_MISO\_SAMPLE.

For PSoC 4 MCUs,  $t_{DSO}$  is given in the device datasheet. For internally-clocked mode, it is proportional to the frequency of the internal clock. For example it may be  $20 \text{ ns} + 3 * t_{CLK\_SCB}$ . Assuming 0 ns PCB delays, and a 0 ns external master  $t_{DSI}$  Equation 26-1 can be re-arranged to  $t_{CLK\_SCB} \leq ((t_{SCLK}) - 40 \text{ ns})/6$ .

## 16.3.6 Enabling and Initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCB\_SPI\_CTRL register. This includes selecting the sub-modes of the protocol and selecting master-slave functionality. EZSPI can be used with slave mode only.
2. Program the OVS field and configure `clk_scb` as appropriate. See the [Clocking System chapter on page 88](#) for more information on how to program clocks and connect it to the SCB.
3. Configure SPI GPIO by setting appropriate drive modes and HSIOM settings.
4. Select the desired Slave Select line and polarity in the SCB\_SPI\_CTRL register.
5. Program the generic transmitter and receiver information using the SCB\_TX\_CTRL and SCB\_RX\_CTRL registers:
  - 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.
6. Program the transmitter and receiver FIFOs using the SCB\_TX\_FIFO\_CTRL and SCB\_RX\_FIFO\_CTRL registers respectively, as shown in SCB\_TX\_FIFO\_CTRL/SCB\_RX\_FIFO\_CTRL registers. Only for FIFO mode.
  - a. Set the trigger level.
  - b. Clear the transmitter and receiver FIFO and Shift registers.
7. 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:** Re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

## 16.3.7 I/O Pad Connection

### 16.3.7.1 SPI Master

Figure 16-10 and Table 16-4 list the use of the I/O pads for SPI Master.

Figure 16-10. SPI Master I/O Pad Connections

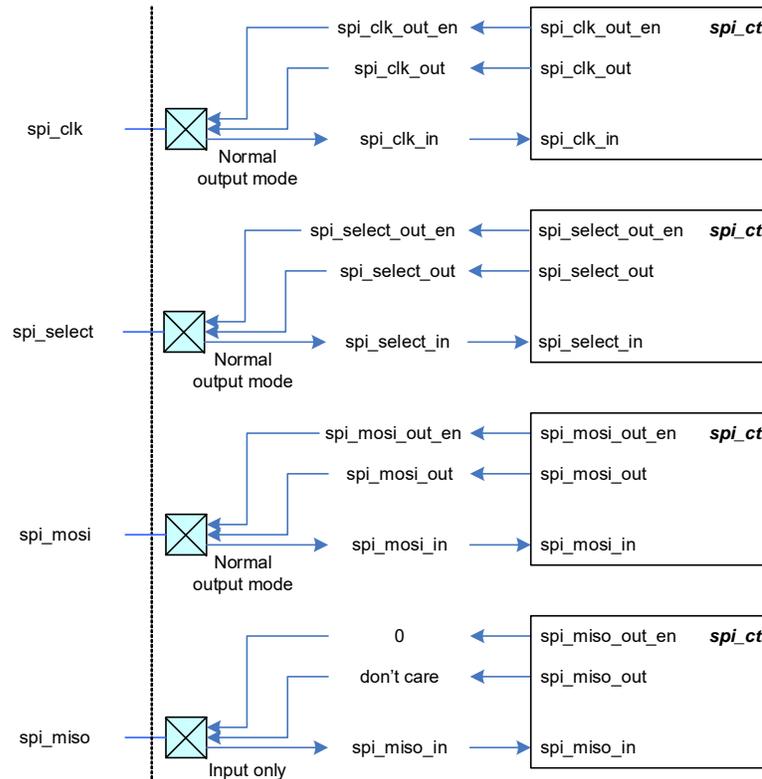


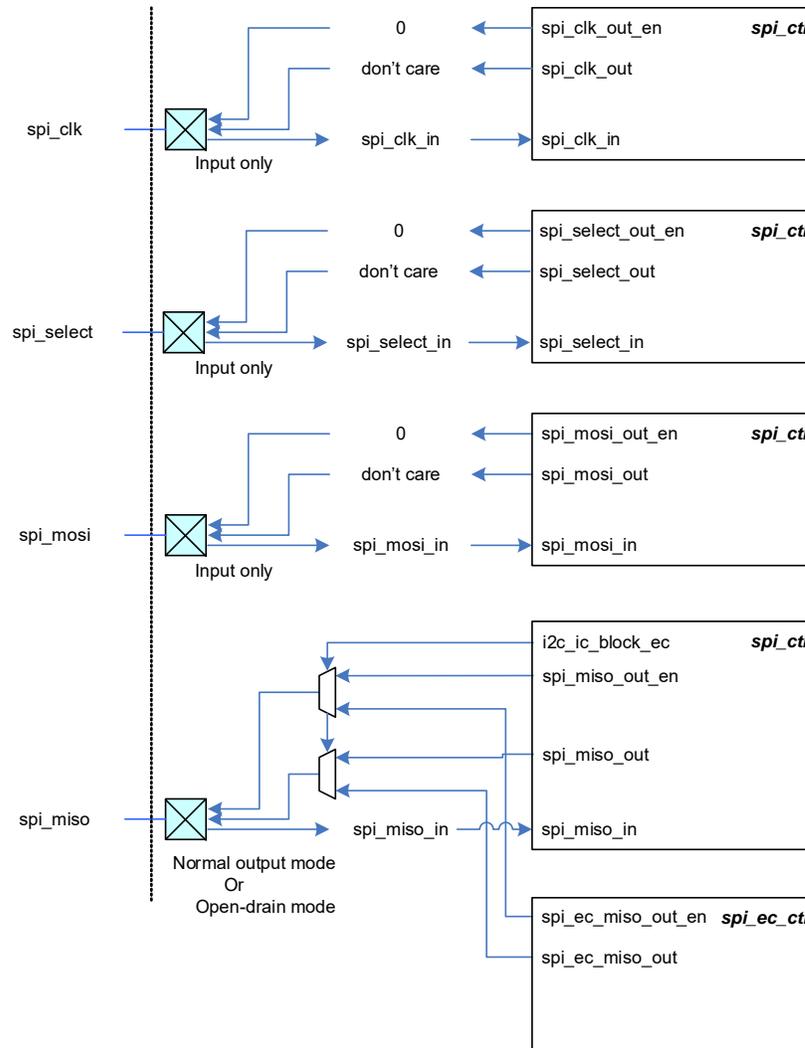
Table 16-4. SPI Master I/O Pad Connection Usage

| I/O Pads   | Drive Mode         | On-chip I/O Signals                 | Usage                    |
|------------|--------------------|-------------------------------------|--------------------------|
| spi_clk    | Normal output mode | spi_clk_out_en<br>spi_clk_out       | Transmit a clock signal  |
| spi_select | Normal output mode | spi_select_out_en<br>spi_select_out | Transmit a select signal |
| spi_mosi   | Normal output mode | spi_mosi_out_en<br>spi_mosi_out     | Transmit a data element  |
| spi_miso   | Input only         | spi_miso_in                         | Receive a data element   |

### 16.3.7.2 SPI Slave

Figure 16-11 and Table 16-5 list the use of I/O pads for SPI Slave.

Figure 16-11. SPI Slave I/O Pad Connections



Open\_Drain is set in the TX\_CTRL register. In this mode the SPI MISO pin is actively driven low, and then high-z for driving high. This means an external pull-up is required for the line to go high. This mode is useful when there are multiple slaves on the same line. This helps to avoid bus contention issues.

Table 16-5. SPI Slave I/O Signal Description

| I/O Pads   | Drive Mode         | On-chip I/O Signals             | Usage                   |
|------------|--------------------|---------------------------------|-------------------------|
| spi_clk    | Input mode         | spi_clk_in                      | Receive a clock signal  |
| spi_select | Input mode         | spi_select_in                   | Receive a select signal |
| spi_mosi   | Input mode         | spi_mosi_in                     | Receive a data element  |
| spi_miso   | Normal output mode | spi_miso_out_en<br>spi_miso_out | Transmit a data element |

### 16.3.7.3 Glitch Avoidance at System Reset

The SPI outputs are in high-impedance digital state when the device is coming out of system reset. This can cause glitches on the outputs. This is important if you are concerned with SPI master SS0 – SS3 or SCLK output pins activity at either device startup or when coming out of Hibernate mode. External pull-up or pull-down resistor can be connected to the output pin to keep it in the inactive state.

## 16.3.8 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in [Table 16-6](#). For more information on these registers, see the *PSoC 4100S Max: PSoC 4 Registers TRM*.

Table 16-6. SPI Registers

| Register Name         | Operation  |
|-----------------------|--|
| SCB_CTRL              | Enables the SCB, selects the type of serial interface (SPI, UART, I <sup>2</sup> C), and selects internally and externally clocked operation, and 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 SS signal in TI SPI.  |
| SCB_SPI_STATUS        | Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode.  |
| SCB_TX_CTRL           | Specifies the data frame width and specifies whether MSb or LSb is the first bit in transmission.  |
| SCB_RX_CTRL           | Performs the same function as that of the SCB_TX_CTRL register, but for the receiver. Also decides whether a median filter is to be used on the input interface lines.   |
| SCB_TX_FIFO_CTRL      | Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO.   |
| SCB_RX_FIFO_CTRL      | Performs the same function as that of the SCB_TX_FIFO_CTRL register, but for the receiver.   |
| SCB_TX_FIFO_WR        | Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.   |
| SCB_RX_FIFO_RD        | Holds the data frame read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO - behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.                    |
| SCB_RX_FIFO_RD_SILENT | Holds the data frame read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.  |
| SCB_TX_FIFO_STATUS    | Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides whether the transmitter FIFO holds the valid data. |
| SCB_RX_FIFO_STATUS    | Performs the same function as that of the SCB_TX_FIFO_STATUS register, but for the receiver.   |
| SCB_EZ_DATA           | Holds the data in EZ memory location.  |

## 16.4 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

Additionally, two side-band signals are used to implement flow control in UART. Note that the flow control applies only to TX functionality.

- Clear to Send (CTS): This is an input signal to the transmitter. When active, the receiver signals to the transmitter that it is ready to receive.
- Ready to Send (RTS): This is an output signal from the receiver. When active, it indicates that the receiver is ready to receive data.

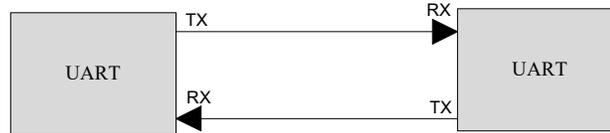
### 16.4.1 Features

- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader
  - IrDA
- Multi-processor mode
- 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)
- 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)
- Median filter on RX input
- Programmable oversampling
- Start skipping
- Hardware flow control

## 16.4.2 General Description

Figure 16-12 illustrates a standard UART TX and RX.

Figure 16-12. 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. Because the interface does not have a clock (asynchronous), the transmitter and receiver use their own clocks; thus, the transmitter and receiver need to agree on the baud rate.

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.

**Note:** UART interface does not support external clocking operation. Hence, UART operates only in the Active and Sleep system power modes. UART also supports only the FIFO buffer mode.

## 16.4.3 UART Modes of Operation

### 16.4.3.1 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.

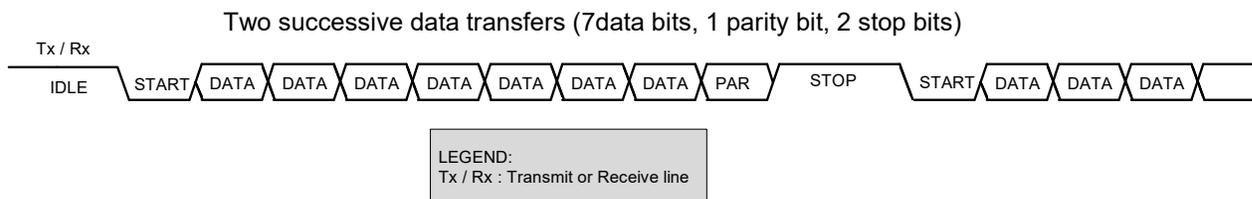
Because the interface does not have a clock, the transmitter and receiver must agree upon the baud rate. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample the incoming signal.

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.

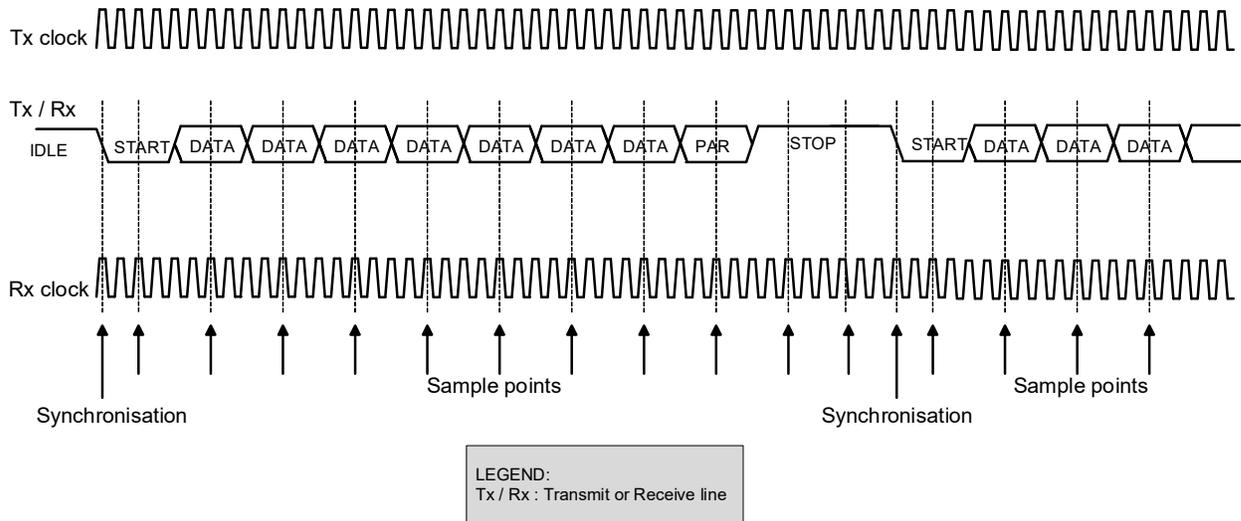
Figure 16-13 illustrates the UART protocol.

Figure 16-13. UART, Standard Protocol Example



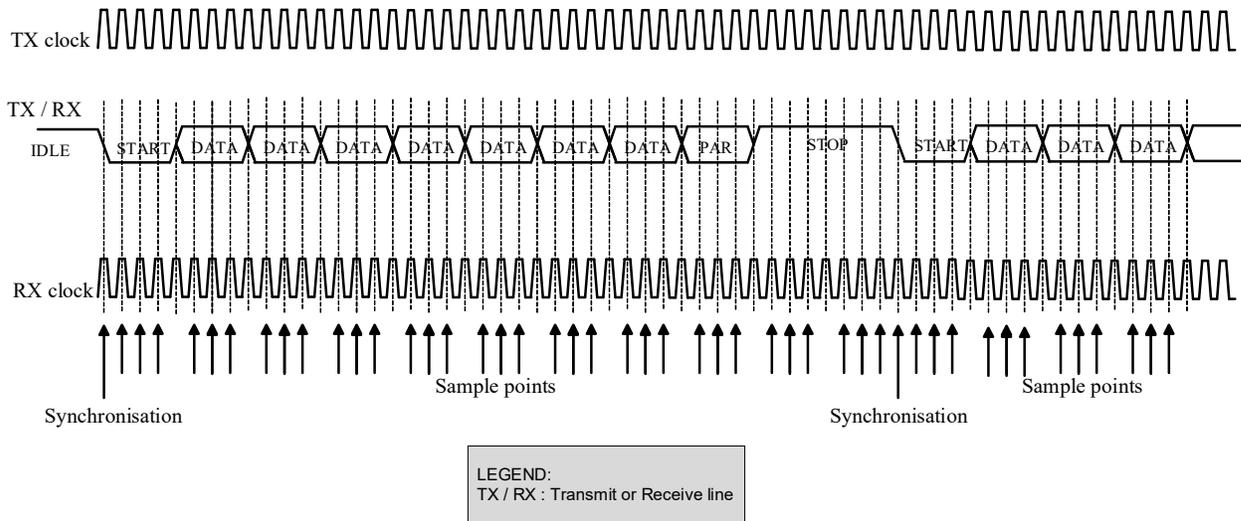
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. [Figure 16-14](#) illustrates this.

Figure 16-14. 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; this is enabled by enabling the MEDIAN filter in the SCB\_RX\_CTRL register. [Figure 16-15](#) illustrates this.

Figure 16-15. UART, Standard Protocol (Multiple Samples)



## Parity

This functionality adds a parity bit to the data frame and is used to identify single-bit data frame errors. The parity bit is always directly after the data frame bits.

The transmitter calculates the parity bit (when `UART_TX_CTRL.PARITY_ENABLED` is 1) from the data frame bits, such that data frame bits and parity bit have an even (`UART_TX_CTRL.PARITY` is 0) or odd (`UART_TX_CTRL.PARITY` is 1) parity. The receiver checks the parity bit (when `UART_RX_CTRL.PARITY_ENABLED` is 1) from the received data frame bits, such that data frame bits and parity bit have an even (`UART_RX_CTRL.PARITY` is 0) or odd (`UART_RX_CTRL.PARITY` is 1) parity.

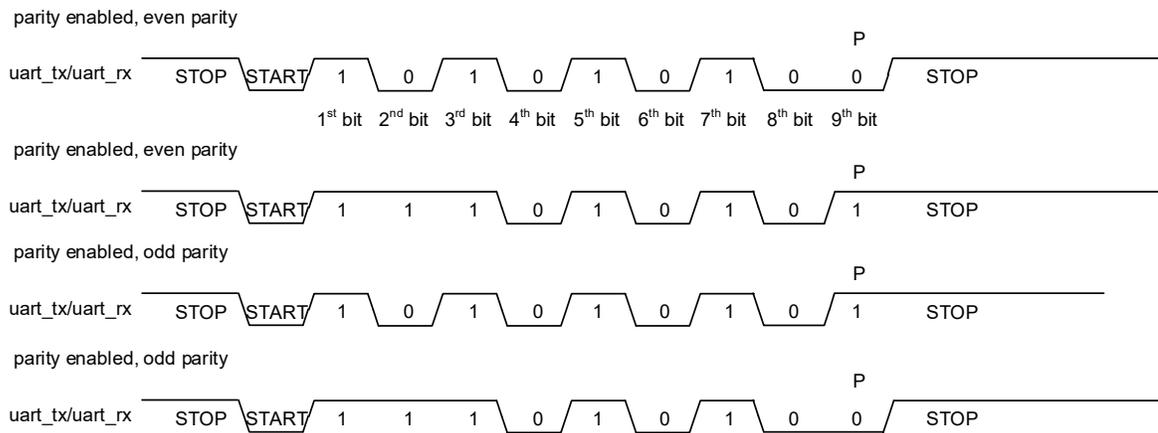
Parity applies to both TX and RX functionality and dedicated control fields are available.

- Transmit functionality: `UART_TX_CTRL.PARITY` and `UART_TX_CTRL.PARITY_ENABLED`.
- Receive functionality: `UART_RX_CTRL.PARITY` and `UART_RX_CTRL.PARITY_ENABLED`.

When a receiver detects a parity error, the data frame is either put in RX FIFO (`UART_RX_CTRL.DROP_ON_PARITY_ERROR` is 0) or dropped (`UART_RX_CTRL.DROP_ON_PARITY_ERROR` is 1).

Figure 16-6 illustrates the parity functionality (8-bit data frame).

Figure 16-16. UART Parity Examples



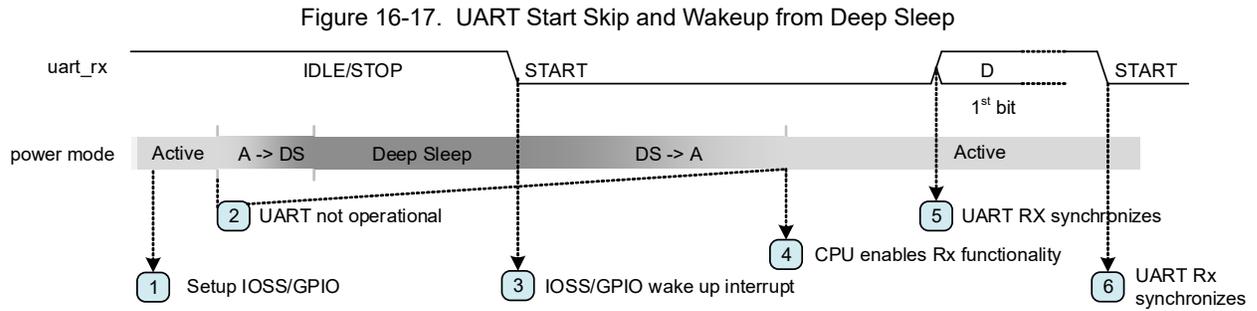
## Start Skipping

Start skipping applies only to receive functionality. The standard UART mode supports “start skipping”. Regular receive operation synchronizes on the START bit period (a 1 to 0 transition on the UART RX line), start skipping receive operation synchronizes on the first received data frame bit, which must be a ‘1’ (a 0 to 1 transition on UART RX).

Start skipping is used to allow for wake up from system Deep Sleep mode using UART. The process is described as follows:

1. Before entering Deep Sleep power mode, UART receive functionality is disabled and the GPIO is programmed to set an interrupt cause to ‘1’ when UART RX line has a ‘1’ to ‘0’ transition (START bit).
2. While in Deep Sleep mode, the UART receive functionality is not functional.
3. The GPIO interrupt is activated on the START bit and the system transitions from Deep Sleep to Active power mode.
4. The CPU enables UART receive functionality, with `UART_RX_CTRL.SKIP_START` bitfield set to ‘1’.
5. The UART receiver synchronizes data frame receipt on the next ‘0’ to ‘1’ transition. If the UART receive functionality is enabled in time, this is the transition from the START bit to the first received data frame bit.
6. The UART receiver proceeds with normal operation; that is, synchronization of successive data frames is on the START bit period.

Figure 16-17 illustrates the process.



Note that the above process works only for lower baud rates. The Deep Sleep to Active power mode transition and CPU enabling the UART receive functionality should take less than 1-bit period to ensure that the UART receiver is active in time to detect the '0' to '1' transition.

In step 4 of the above process, the firmware takes some time to finish the wakeup interrupt routine and enable the UART receive functionality before the block can detect the input rising edge on the UART RX line.

If the above steps cannot be completed in less than 1 bit time, first send a “dummy” byte to the device to wake it up before sending real UART data. In this case, the SKIP\_START bit can be left as 0.

**Note:** If skip start is used and a wakeup occurs from another source then a dummy byte will be received in the RX\_FIFO when the RX line is in idle.

### Break Detection

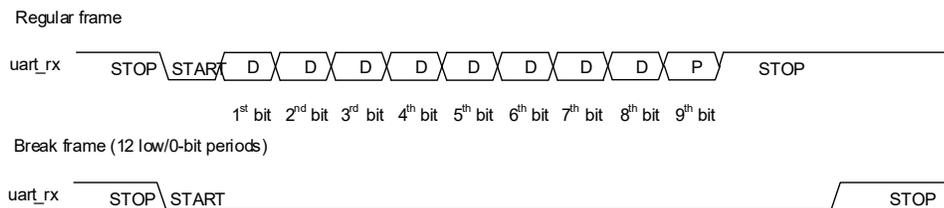
Break detection is supported in the standard UART mode. This functionality detects when UART RX line is low (0) for more than UART\_RX\_CTRL.BREAK\_WIDTH bit periods. The break width should be larger than the maximum number of low (0) bit periods in a regular data transfer, plus an additional 1-bit period. The additional 1-bit period is a minimum requirement and preferably should be larger. The additional bit periods account for clock inaccuracies between transmitter and receiver.

For example, for an 8-bit data frame with parity support, the maximum number of low (0) bit periods is 10 (START bit, 8 '0' data frame bits, and one '0' parity bit). Therefore, the break width should be larger than 10 + 1 = 11 (UART\_RX\_CTRL.BREAK\_WIDTH can be set to 11).

Note that the break detection applies only to receive functionality. A UART transmitter can generate a break by temporarily increasing TX\_CTRL.DATA\_WIDTH and transmitting an all zeroes data frame. A break is used by the transmitter to signal a special condition to the receiver. This condition may result in a reset, shut down, or initialization sequence at the receiver.

Break detection is part of the LIN protocol. When a break is detected, the INTR\_RX.BREAK\_DETECT interrupt cause is set to '1'. Figure 16-18 illustrates a regular data frame and break frame (8-bit data frame, parity support, and a break width of 12-bit periods).

Figure 16-18. UART – Regular Frame and Break Frame



### Flow Control

The standard UART mode supports flow control. Modem flow control controls the pace at which the transmitter transfers data to the receiver. Modem flow control is enabled through the `UART_FLOW_CTRL.CTS_ENABLED` register field. When this field is '0', the transmitter transfers data when its TX FIFO is not empty. When '1', the transmitter transfers data when UART CTS line is active and its TX FIFO is not empty.

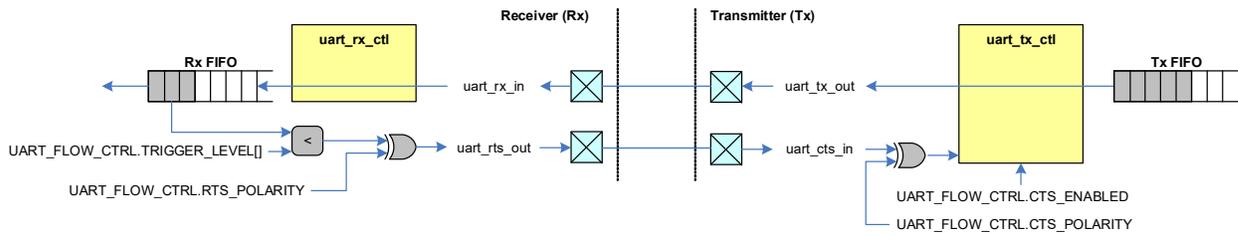
Note that the flow control applies only to TX functionality. Two UART side-band signal are used to implement flow control:

- UART RTS (`uart_rts_out`): This is an output signal from the receiver. When active, it indicates that the receiver is ready to receive data (RTS: Ready to Send).
- UART CTS (`uart_cts_in`): This is an input signal to the transmitter. When active, it indicates that the transmitter can transfer data (CTS: Clear to Send).

The receiver's `uart_rts_out` signal is connected to the transmitter's `uart_cts_in` signal. The receiver's `uart_rts_out` signal is derived by comparing the number of used receive FIFO entries with the `UART_FLOW_CTRL.TRIGGER_LEVEL` field. If the number of used receive FIFO entries are less than `UART_FLOW_CTRL.TRIGGER_LEVEL`, `uart_rts_out` is activated.

Typically, the UART side-band signals are active low. However, sometimes active high signaling is used. Therefore, the polarity of the side-band signals can be controlled using bitfields `UART_FLOW_CTRL.RTS_POLARITY` and `UART_FLOW_CTRL.CTS_POLARITY`. Figure 16-19 gives an overview of the flow control functionality.

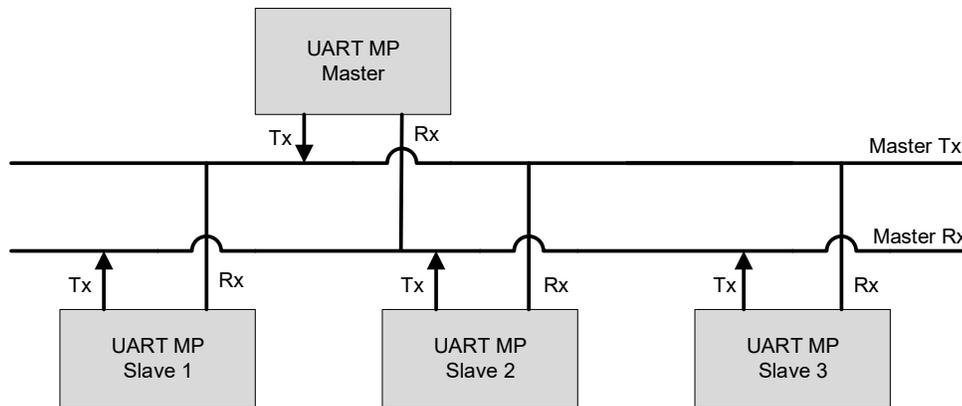
Figure 16-19. UART Flow Control Connection



### UART Multi-Processor Mode

The `UART_MP` (multi-processor) mode is defined with single-master-multi-slave topology, as Figure 16-20 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.

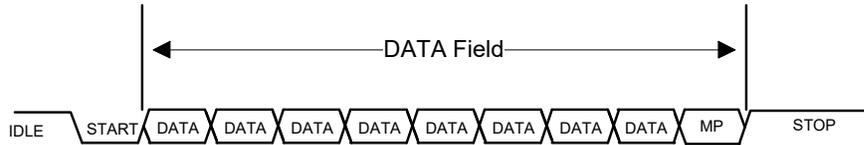
Figure 16-20. UART MP Mode Bus Connections



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 16-21](#).
- Parity bit is disabled.

Figure 16-21. UART MP Address and 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 the received address does not match its own address, then the interface ignores the following data, until next address is received for compare.

### Configuring the SCB as Standard UART Interface

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

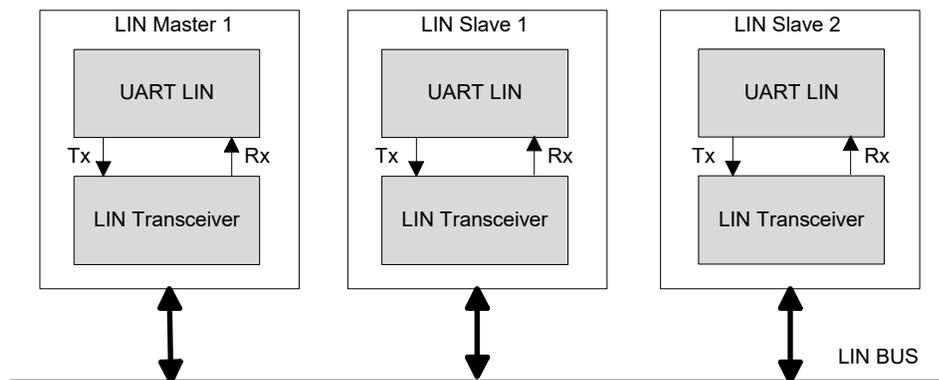
1. Configure the SCB as UART interface by writing '10b' to the MODE field (bits [25:24]) of the SCB\_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCB\_UART\_CTRL register.
3. To enable the UART MP Mode or UART 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 4 described in [“Enabling and Initializing the UART” on page 151](#).

For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

#### 16.4.3.2 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 16-22](#) illustrates the UART\_LIN and LIN transceiver.

Figure 16-22. UART\_LIN and LIN Transceiver

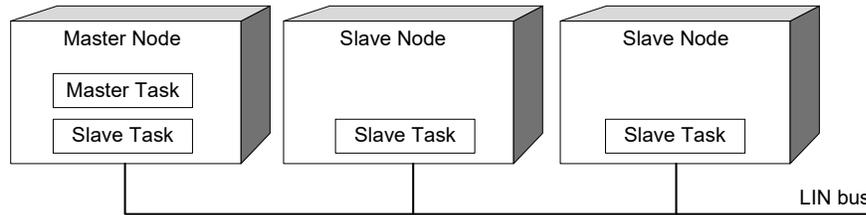


LIN protocol defines two tasks:

- 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 16-23](#).

Figure 16-23. LIN Bus Nodes and Tasks

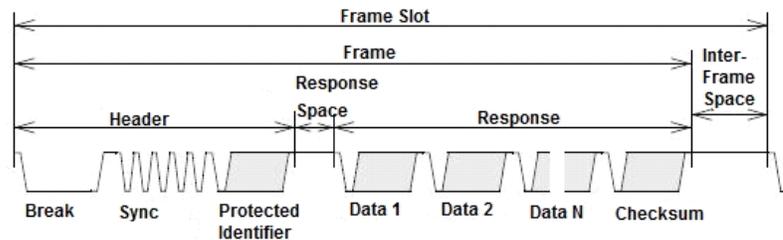


### LIN Frame Structure

LIN is based on the transmission of frames at pre-determined moments of time. A frame is divided into header and response fields, as shown in [Figure 16-24](#).

- 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 16-24. 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 field has logical zero with a minimum of 13 bit times and followed by a break delimiter. The break field structure is as shown in [Figure 16-25](#).

Figure 16-25. 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 16-26](#) shows the LIN sync field structure.

Figure 16-26. 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 16-27](#).

- Frame identifier: The frame identifiers are divided 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 16-27](#) shows the PID field structure.

Figure 16-27. PID Field



### Data

In LIN, every frame can carry a minimum of one byte and maximum of eight bytes of data. Here, the LSb of the data byte is sent first and the MSb of the data byte is sent last.

### Checksum

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

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

### LIN Frame Types

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

#### Unconditional Frame

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

### Event-Triggered Frame

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

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

### Sporadic Frame

The purpose of the sporadic frames is to merge some dynamic behavior into the schedule table without affecting the rest of the schedule table. These frames have a group of unconditional frames that share the frame slot. When the sporadic frame is due for transmission, the unconditional frames are checked whether 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 whether it will be transmitted or whether 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  $\mu$ s 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 in UART LIN Mode

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 LIN Mode, write '1' to the LIN\_MODE (bit 12) of the SCB\_UART\_RX\_CTRL register.
4. Follow steps 2 to 4 described in Enabling and Initializing UART.

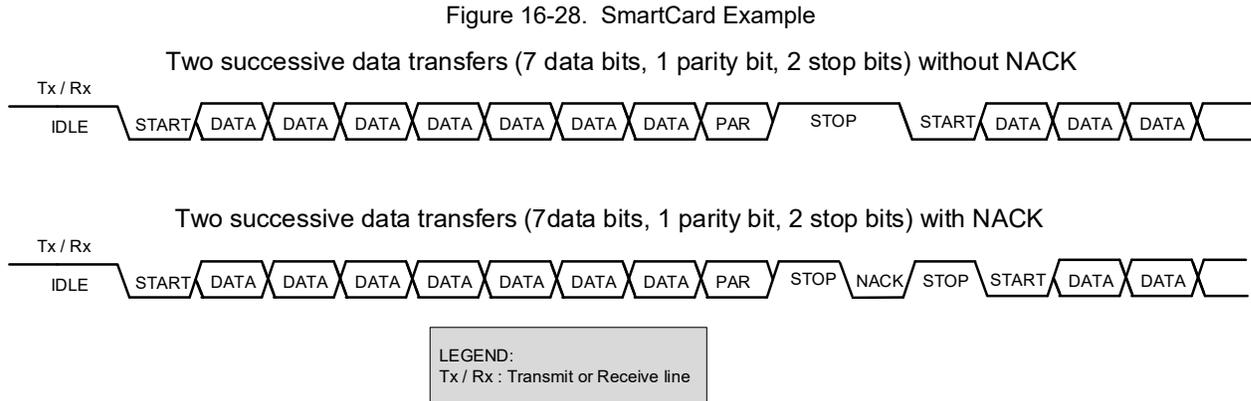
### 16.4.3.3 SmartCard (ISO7816)

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 the master (reader) function is supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. The 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 16-28 illustrates the SmartCard protocol.



The communication Baud rate while using SmartCard is given as:

$$\text{Baud rate} = \text{Fscbclk}/\text{Oversample}$$

#### 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 ModusToolbox Software does all this automatically with the help of GUIs. For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

1. Configure the SCB as UART interface by writing '10b' 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 4 described in "Enabling and Initializing the UART" on page 151.

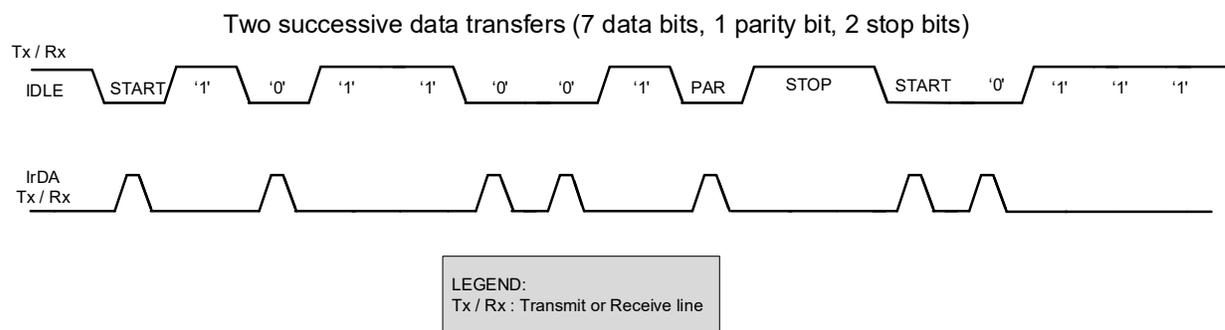
### 16.4.3.4 Infrared Data Association (IrDA)

The SCB supports the IrDA protocol for data rates of up to 115.2 kbps using the UART interface. It supports only the basic physical layer of 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 ( $\leq 115.2$  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. In addition, the PSoC 4 MCU SCB supports a low-power IrDA receiver mode, which allows it to detect pulses with a minimum width of  $1.41 \mu\text{s}$ .

Different communication speeds under 115.2 kbps can be achieved by configuring clk\_scb frequency. Additional allowable rates are 2.4 kbps, 9.6 kbps, 19.2 kbps, 38.4 kbps, and 57.6 kbps. Figure 16-29 shows how a UART transfer is IrDA modulated.

Figure 16-29. IrDA Example



### Configuring the SCB as a UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order; note that ModusToolbox Software does all this automatically with the help of GUIs. For more information on these registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

1. Configure the SCB as a UART interface by writing '10b' 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 the UART" on page 151](#).

### 16.4.4 Clocking and Oversampling

The UART protocol is implemented using clk\_scb as an oversampled multiple of the baud rate. For example, to implement a 100-kHz UART, clk\_scb could be set to 1 MHz and the oversample factor set to '10'. The oversampling is set using the SCB\_CTRL.OVS register field. The oversampling value is SCB\_CTRL.OVS + 1. In the UART standard sub-mode (including LIN) and the SmartCard sub-mode, the valid range for the OVS field is [7, 15].

In UART transmit IrDA sub-mode, this field indirectly specifies the oversampling. Oversampling determines the interface clock per bit cycle and the width of the pulse. This sub-mode has only one valid OVS value—16 (which is a value of 0 in the OVS field of the SCB\_CTRL register); the pulse width is roughly  $3/16$  of the bit period (for all bit rates).

In UART receive IrDA sub-mode (1.2, 2.4, 9.6, 19.2, 38.4, 57.6, and 115.2 kbps), this field indirectly specifies the oversampling. In normal transmission mode, this pulse is approximately  $3/16$  of the bit period (for all bit rates). In low-power transmission mode, this pulse is potentially smaller (down to  $1.62 \mu\text{s}$  typical and  $1.41 \mu\text{s}$  minimal) than  $3/16$  of the bit period (for less than 115.2 kbps bit rates).

Pulse widths greater than or equal to two SCB input clock cycles are guaranteed to be detected by the receiver. Pulse widths less than two clock cycles and greater than or equal to one SCB input clock cycle may be detected by the receiver. Pulse widths less than one SCB input clock cycle will not be detected by the receiver. Note that the SCB\_RX\_CTRL.MEDIAN should be set to '1' for IrDA receiver functionality.

The SCB input clock and the oversampling together determine the IrDA bit rate. Refer to the *PSoC 4100S Max: PSoC 4 Registers TRM* for more details on the OVS values for different baud rates.

## 16.4.5 Enabling and Initializing the UART

The UART must be programmed in the following order:

1. Program protocol specific information using the UART\_TX\_CTRL, UART\_RX\_CTRL, and UART\_FLOW\_CTRL registers. 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.
  - 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.
  - a. Set the trigger level.
  - b. Clear the transmitter and receiver FIFO and Shift registers.
4. Enable the block (write a '1' to the ENABLE 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).

## 16.4.6 I/O Pad Connection

### 16.4.6.1 Standard UART Mode

Figure 16-30 and Table 16-7 list the use of the I/O pads for the Standard UART mode.

Figure 16-30. Standard UART Mode I/O Pad Connections

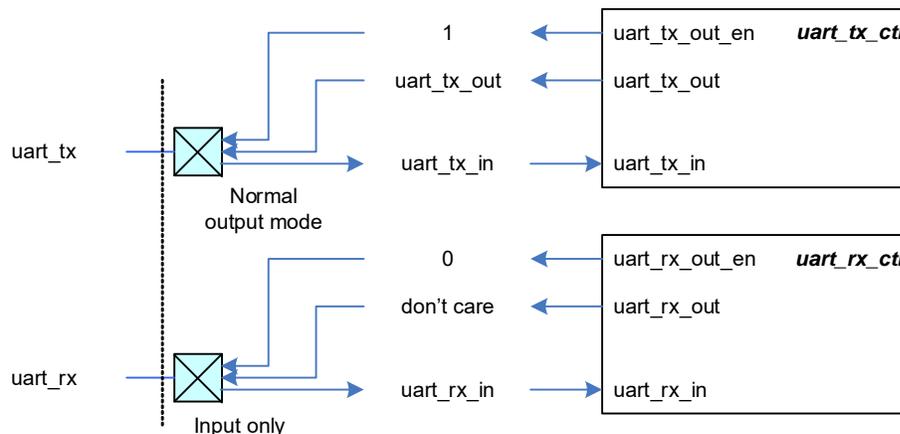


Table 16-7. UART I/O Pad Connection Usage

| I/O Pads             | Drive Mode         | On-chip I/O Signals                                     | Usage                   |
|----------------------|--------------------|---|-------------------------|
| <code>uart_tx</code> | Normal output mode | <code>uart_tx_out_en</code><br><code>uart_tx_out</code> | Transmit a data element |
| <code>uart_rx</code> | Input only         | <code>uart_rx_in</code>                                 | Receive a data element  |

### 16.4.6.2 SmartCard Mode

Figure 16-31 and Table 16-8 list the use of the I/O pads for the SmartCard mode.

Figure 16-31. SmartCard Mode I/O Pad Connections

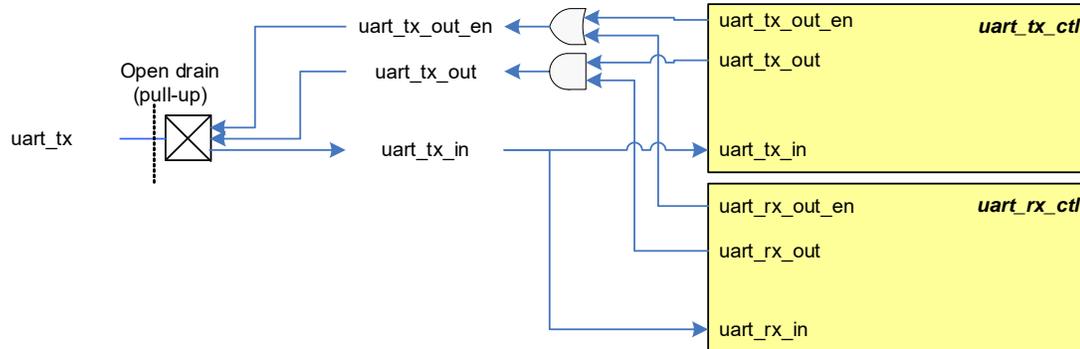


Table 16-8. SmartCard Mode I/O Pad Connections

| I/O Pads | Drive Mode              | On-chip I/O Signals           | Usage   |
|----------|-------------------------|-------------------------------|---|
| uart_tx  | Open drain with pull-up | uart_tx_in                    | Used to receive a data element.<br>Receive a negative acknowledgement of a transmitted data element |
|          |                         | uart_tx_out_en<br>uart_tx_out | Transmit a data element.<br>Transmit a negative acknowledgement to a received data element.         |

### 16.4.6.3 LIN Mode

Figure 16-32 and Table 16-9 list the use of the I/O pads for LIN mode.

Figure 16-32. LIN Mode I/O Pad Connections

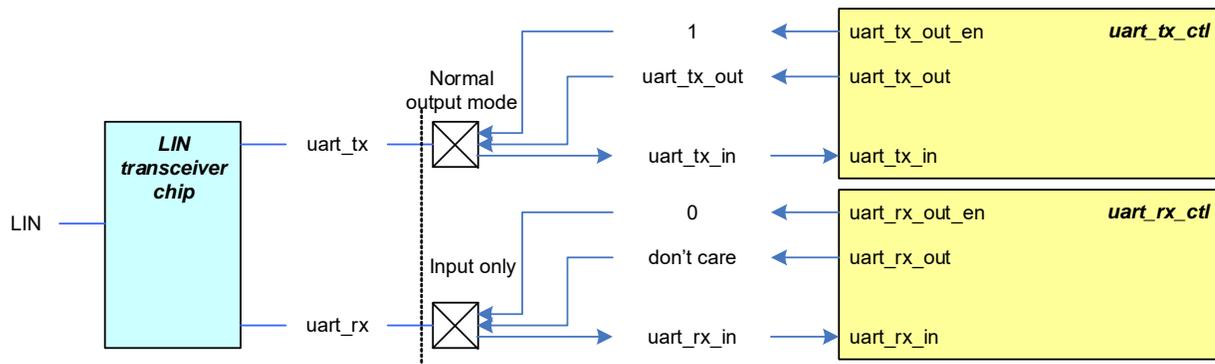


Table 16-9. LIN Mode I/O Pad Connections

| I/O Pads | Drive Mode         | On-chip I/O Signals           | Usage                    |
|----------|--------------------|-------------------------------|--------------------------|
| uart_tx  | Normal output mode | uart_tx_out_en<br>uart_tx_out | Transmit a data element. |
| uart_rx  | Input only         | uart_rx_in                    | Receive a data element.  |

### 16.4.6.4 IrDA Mode

Figure 16-33 and Table 16-10 list the use of the I/O pads for IrDA mode.

Figure 16-33. IrDA Mode I/O Pad Connections

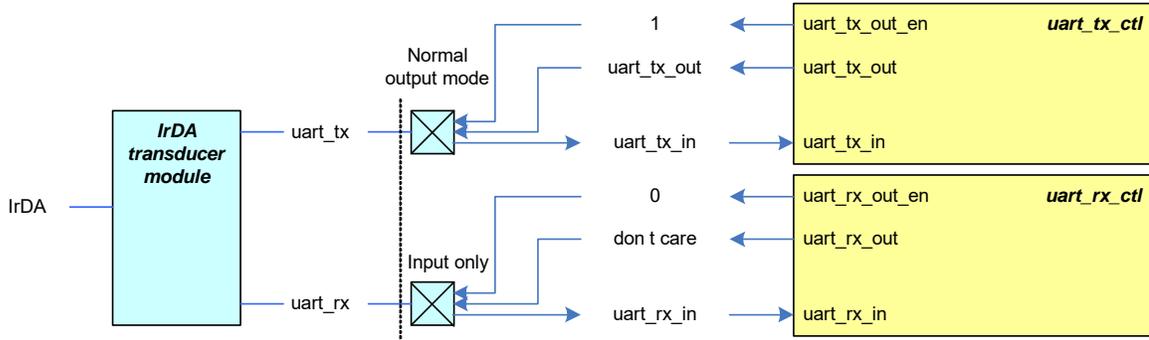


Table 16-10. IrDA Mode I/O Pad Connections

| I/O Pads | Drive Mode         | On-chip I/O Signals           | Usage                    |
|----------|--------------------|-------------------------------|--------------------------|
| uart_tx  | Normal output mode | uart_tx_out_en<br>uart_tx_out | Transmit a data element. |
| uart_rx  | Input only         | uart_rx_in                    | Receive a data element.  |

### 16.4.7 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 16-11. For more information on these registers, see the *PSoC 4100S Max: PSoC 4 Registers TRM*.

Table 16-11. UART Registers

| Register Name         | Operation   |
|-----------------------|---|
| SCB_CTRL              | Enables the SCB; selects the type of serial interface (SPI, UART, I <sup>2</sup> C)   |
| 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.   |

## 16.5 Inter Integrated Circuit (I<sup>2</sup>C)

This section explains the I<sup>2</sup>C implementation in the PSoC 4 MCU. For more information on the I<sup>2</sup>C protocol specification, refer to the I<sup>2</sup>C-bus specification available on the [NXP website](#).

### 16.5.1 Features

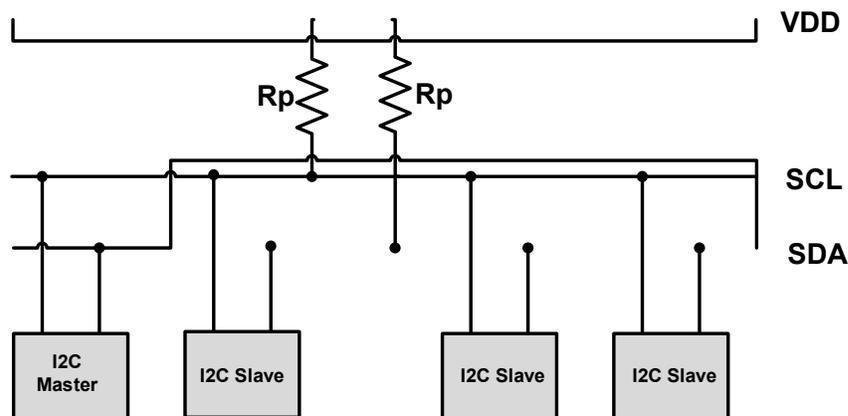
This block supports the following features:

- Master, slave, and master/slave mode
- Slow-mode (50 kbps), standard-mode (100 kbps), fast-mode (400 kbps), fast-mode plus (1000 kbps), high speed mode (3.4 Mbps)
- 7-bit slave addressing
- Clock stretching
- Collision detection
- Programmable oversampling of I<sup>2</sup>C clock signal (SCL)
- Auto ACK when RX FIFO not full, including address
- General address detection
- FIFO and EZ modes

### 16.5.2 General Description

Figure 16-34 illustrates an example of an I<sup>2</sup>C communication network.

Figure 16-34. I<sup>2</sup>C Interface Block Diagram



The standard I<sup>2</sup>C bus is a two wire interface with the following lines:

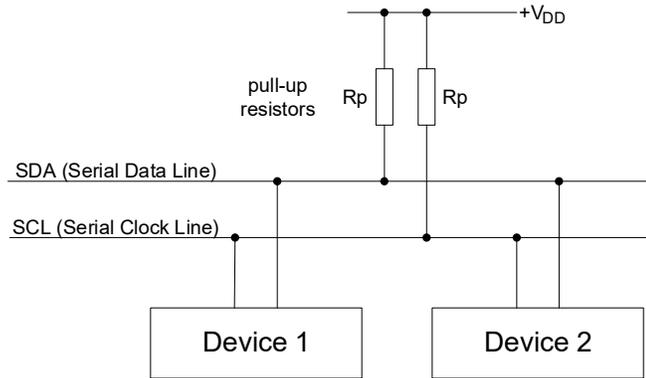
- Serial Data (SDA)
- Serial Clock (SCL)

I<sup>2</sup>C 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.

### 16.5.3 External Electrical Connections

As shown in Figure 16-35, the I<sup>2</sup>C bus requires external pull-up resistors. The pull-up resistors ( $R_p$ ) are primarily determined by the supply voltage, bus speed, and bus capacitance. For detailed information on how to calculate the optimum pull-up resistor value for your design Cypress recommends using the UM10204 I<sup>2</sup>C-bus specification and user manual Rev. 6, available from the NXP website at [www.nxp.com](http://www.nxp.com).

Figure 16-35. Connection of Devices to the I<sup>2</sup>C Bus



For most designs, the default values shown in Table 16-12 provide excellent performance without any calculations. The default values were chosen to use standard resistor values between the minimum and maximum limits.

Table 16-12. Recommended Default Pull-up Resistor Values

| Standard Mode (0 – 100 kbps) | Fast Mode (0 – 400 kbps) | Fast Mode Plus (0 – 1000 kbps) | Units    |
|------------------------------|--------------------------|--------------------------------|----------|
| 4.7 k, 5%                    | 1.74 k, 1%               | 620, 5%                        | $\Omega$ |

These values work for designs with 1.8 V to 5.0 V  $V_{DD}$ , less than 200 pF bus capacitance ( $C_B$ ), up to 25  $\mu$ A of total input leakage ( $I_{IL}$ ), up to 0.4 V output voltage level ( $V_{OL}$ ), and a max  $V_{IH}$  of  $0.7 * V_{DD}$ . Calculation of custom pull-up resistor values is required if your design does not meet the default assumptions, you use series resistors ( $R_S$ ) to limit injected noise, or you want to maximize the resistor value for low power consumption. Calculation of the ideal pull-up resistor value involves finding a value between the limits set by three equations detailed in the NXP I<sup>2</sup>C specification. These equations are:

$$R_{P_{MIN}} = (V_{DD(max)} - V_{OL(max)}) / I_{OL(min)} \quad \text{Equation 16-4}$$

$$R_{P_{MAX}} = T_R(max) / 0.8473 \times C_B(max) \quad \text{Equation 16-5}$$

$$R_{P_{MAX}} = V_{DD(min)} - (V_{IH(min)} + V_{NH(min)}) / I_{IH(max)} \quad \text{Equation 16-6}$$

Equation parameters:

- $V_{DD}$  = Nominal supply voltage for I<sup>2</sup>C bus
- $V_{OL}$  = Maximum output low voltage of bus devices
- $I_{OL}$  = Low-level output current from I<sup>2</sup>C specification
- $T_R$  = Rise time of bus from I<sup>2</sup>C specification
- $C_B$  = Capacitance of each bus line including pins and PCB traces
- $V_{IH}$  = Minimum high-level input voltage of all bus devices
- $V_{NH}$  = Minimum high-level input noise margin from I<sup>2</sup>C specification
- $I_{IH}$  = Total input leakage current of all devices on the bus

The supply voltage ( $V_{DD}$ ) limits the minimum pull-up resistor value due to bus devices maximum low output voltage ( $V_{OL}$ ) specifications. Lower pull-up resistance increases current through the pins and can therefore exceed the spec conditions of  $V_{OH}$ . Equation 26-4 is derived using Ohm's law to determine the minimum resistance that will still meet the  $V_{OL}$  specification at 3 mA for standard and fast modes, and 20 mA for fast mode plus at the given  $V_{DD}$ .

Equation 26-5 determines the maximum pull-up resistance due to bus capacitance. Total bus capacitance is comprised of all pin, wire, and trace capacitance on the bus. The higher the bus capacitance the lower the pull-up resistance required to meet the specified bus speeds rise time due to RC delays. Choosing a pull-up resistance higher than allowed can result in failing timing requirements resulting in communication errors. Most designs with five or fewer I<sup>2</sup>C devices and up to 20 centimeters of bus trace length have less than 100 pF of bus capacitance.

A secondary effect that limits the maximum pull-up resistor value is total bus leakage calculated in Equation 16-6. The primary source of leakage is I/O pins connected to the bus. If leakage is too high, the pull-ups will have difficulty maintaining an acceptable  $V_{IH}$  level causing communication errors. Most designs with five or fewer I<sup>2</sup>C devices on the bus have less than 10  $\mu$ A of total leakage current.

## 16.5.4 Terms and Definitions

Table 16-13 explains the commonly used terms in an I<sup>2</sup>C communication network.

Table 16-13. Definition of I<sup>2</sup>C 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   |
| 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  |

### 16.5.4.1 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. The SCB on the PSoC 4 MCU can and will stretch the clock.

### 16.5.4.2 Bus Arbitration

The I<sup>2</sup>C 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.

## 16.5.5 I<sup>2</sup>C Modes of Operation

I<sup>2</sup>C 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 16-14 illustrates the I<sup>2</sup>C modes of operation.

Table 16-14. I<sup>2</sup>C Modes

| Mode         | Description  |
|--------------|--|
| Slave        | Slave only operation (default)                     |
| Master       | Master only operation                              |
| Multi-master | Supports more than one master on the bus           |
| Master-slave | The SCB can change between master and slave modes. |

Table 16-15 lists some common bus events that are part of an I<sup>2</sup>C data transfer. The [Write Transfer](#) and [Read Transfer](#) sections explain the I<sup>2</sup>C bus bit format during data transfer.

Table 16-15. I<sup>2</sup>C 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 did not receive 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)  |

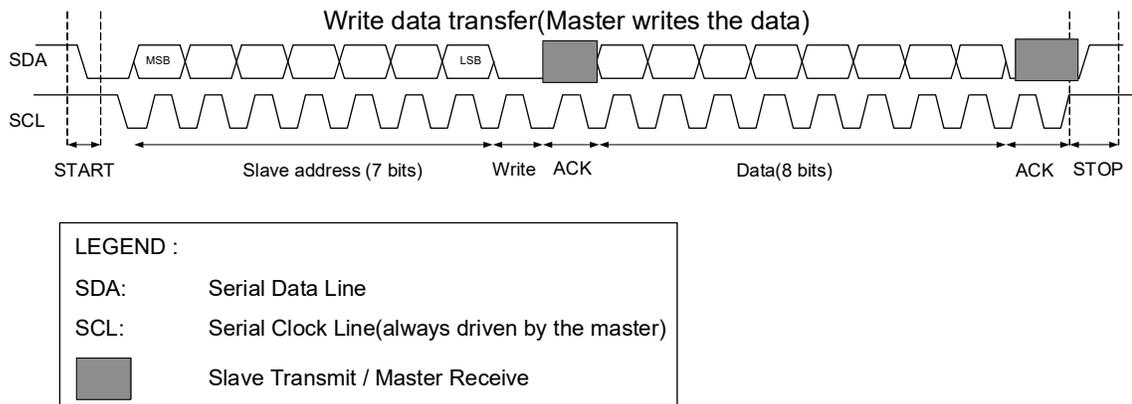
With all of these modes, there are two types of transfer - read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave.

**Note:** High speed mode requires HW detection of the following sequence: START, ADDR, NACK.

### 16.5.5.1 Write Transfer

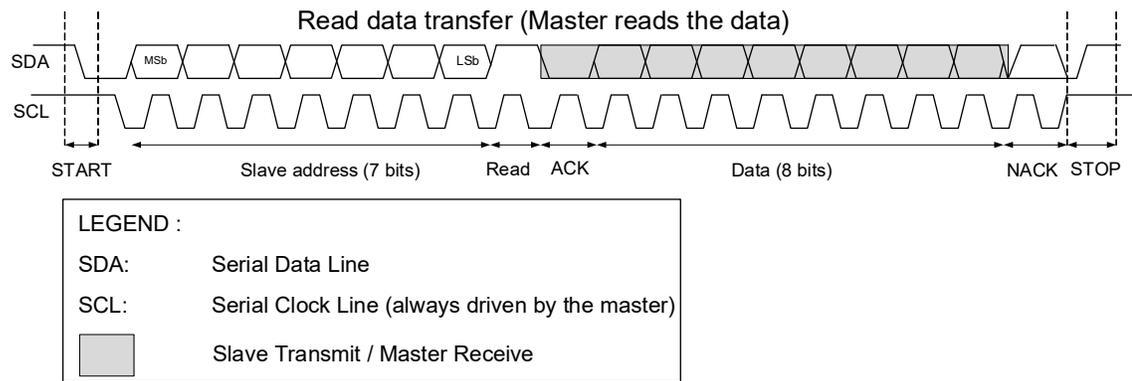
- A typical write transfer begins with the master generating a START condition on the I<sup>2</sup>C bus. The master then writes a 7-bit I<sup>2</sup>C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgment byte by pulling the data line low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

Figure 16-36. Master Write Data Transfer



### 16.5.5.2 Read Transfer

Figure 16-37. Master Read Data Transfer



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

## 16.5.6 I<sup>2</sup>C Buffer Modes

I<sup>2</sup>C can operate in two different buffered modes – FIFO and EZ modes. The buffer is used in different ways in each of the modes. The following subsections explain each of these buffered modes in detail.

### 16.5.6.1 FIFO Mode

The FIFO mode has a TX FIFO for the data being transmitted and an RX FIFO for the data being received. Each FIFO is constructed out of the SRAM buffer. The FIFOs are either 8 elements deep with 16-bit data elements or 16 elements deep with 8-bit data elements. The width of the data elements are configured using the CTRL.BYTE\_MODE bitfield of the SCB. For I<sup>2</sup>C, put the FIFO in BYTE mode because all transactions are a byte wide.

The FIFO mode operation is available only in Active and Sleep power modes, not in the Deep Sleep power mode. However, the slave address can be used to wake the device from sleep.

A write access to the transmit FIFO uses register TX\_FIFO\_WR. A read access from the receive FIFO uses register RX\_FIFO\_RD.

Transmit and receive FIFO status information is available through status registers TX\_FIFO\_STATUS and RX\_FIFO\_STATUS. When in debug mode, a read from this register behaves as a read from the SCB\_RX\_FIFO\_RD\_SILENT register; that is, data will not be removed from the FIFO.

Each FIFO has a trigger output. This trigger output can be routed through the trigger mux to various other peripheral on the device such as DMA or TCPWMs. The trigger output of the SCB is controlled through the TRIGGER\_LEVEL field in the RX\_CTRL and TX\_CTRL registers.

- For a TX FIFO a trigger is generated when the number of entries in the transmit FIFO is less than TX\_FIFO\_CTRL.TRIGGER\_LEVEL.
- For the RX FIFO a trigger is generated when the number of entries in the FIFO is greater than the RX\_FIFO\_CTRL.TRIGGER\_LEVEL.

Note that the DMA has a trigger deactivation setting. For the SCB this should be set to 16.

### Active to Deep Sleep Transition

Before going to deep sleep ensure that all active communication is complete. This can be done by checking the BUS\_BUSY bit in the I2C\_Status register.

Ensure that the TX and RX FIFOs are empty as any data will be lost during deep sleep.

Before going to deep sleep, the clock to the SCB needs to be disabled. This can be done by disabling the clock divider that supplies the clock to the SCB, see the [Clocking System chapter on page 88](#) for more information.

### Deep Sleep to Active Transition

EC\_AM = 1, EC\_OP = 0, FIFO Mode.

The following descriptions only apply to slave mode.

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 I<sup>2</sup>C slave executes clock stretching and waits until the device wakes up and ACKs the address.
- Address NACK: The I<sup>2</sup>C 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 mode.

#### Notes:

- The interrupt bit WAKE\_UP (bit 0) of the SCB\_INTR\_I2C\_EC register must be enabled for the I<sup>2</sup>C to wake up the device on slave address match while switching to the Sleep mode.
- If the device is configured in I<sup>2</sup>C slave mode, the clock to the SCB should be disabled when entering Deep-Sleep power mode; enable the clock when waking up from Deep-Sleep mode.

### 16.5.6.2 EZI2C Mode

The Easy I<sup>2</sup>C (EZI2C) protocol is a unique communication scheme built on top of the I<sup>2</sup>C protocol by Cypress. It uses a meta protocol around the standard I<sup>2</sup>C protocol to communicate to an I<sup>2</sup>C slave using indexed memory transfers. This removes the need for CPU intervention.

The EZI2C protocol defines a single memory buffer with an 8-bit address that indexes the buffer (32-entry array of 8-bit per entry is supported) located on the slave device. The EZ address is used to address these 32 locations. The CPU writes and reads to the memory buffer through the EZ\_DATA registers. These accesses are word accesses, but only the least significant byte of the word is used.

The slave interface accesses the memory buffer using the current address. At the start of a transfer (I<sup>2</sup>C START/RESTART), the base address is copied to the current address. A data element write or read operation is to the current address location. After the access, the current address is incremented by '1'.

If the current address equals the last memory buffer address (31), the current address is not incremented. Subsequent write accesses will overwrite any previously written value at the last buffer address. Subsequent read accesses will continue to provide the (same) read value at the last buffer address. The bus master should be aware of the memory buffer capacity in EZ mode.

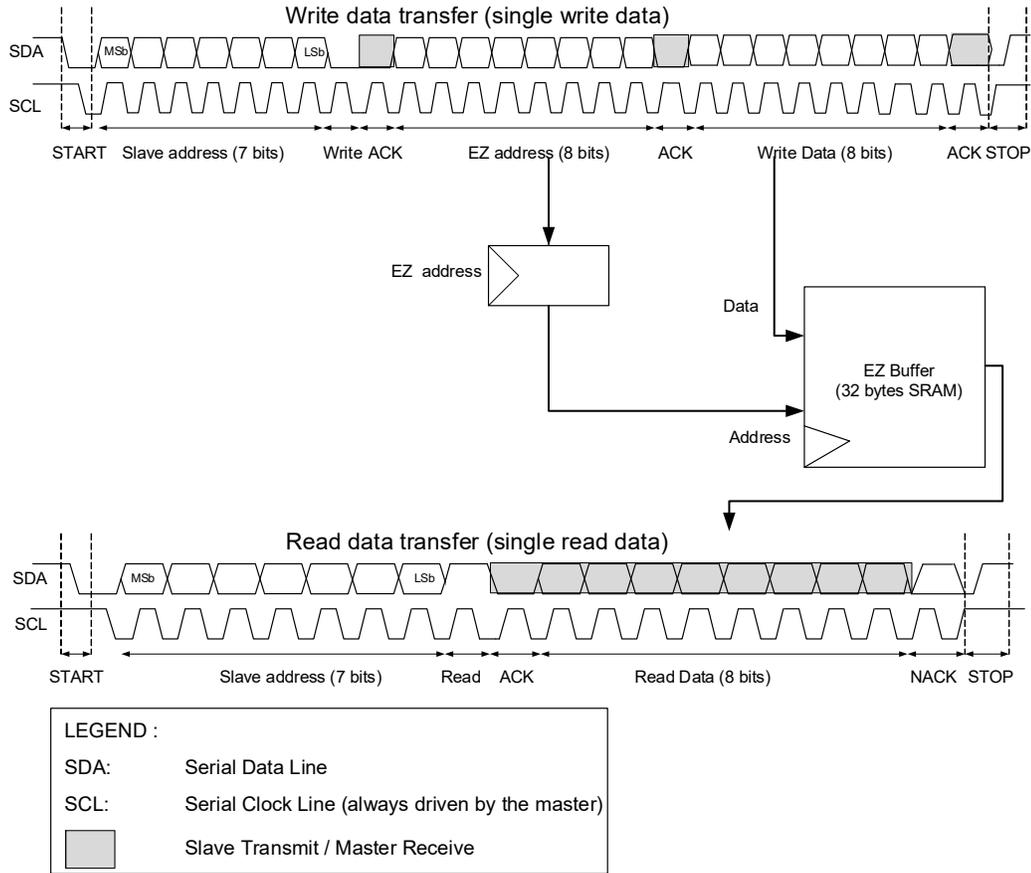
The I<sup>2</sup>C base and current addresses are provided through I2C\_STATUS. At the end of a transfer, the difference between the base and current addresses indicates how many read or write accesses were performed. The block provides interrupt cause fields to identify the end of a transfer. EZI2C can be implemented through firmware or hardware. All SCBs can implement EZI2C through a firmware implementation in both Active and Sleep power modes. The SCB can implement a hardware based EZI2C that can operate in deep sleep. This document focuses on hardware-implemented EZI2C.

EZI2C distinguishes three operation phases:

- Address phase: The master transmits an 8-bit address to the slave. This address is used as the slave base and current address.
- Write phase: The master writes 8-bit data element(s) to the slave's memory buffer. The slave's current address is set to the slave's base address. Received data elements are written to the current address memory location. After each memory write, the current address is incremented.
- Read phase: The master reads 8-bit data elements from the slave's memory buffer. The slave's current address is set to the slave's base address. Transmitted data elements are read from the current address memory location. After each memory read, the current address is incremented.

Note that a slave's base address is updated by the master and not by the CPU.

Figure 16-38. EZI2C Write and Read Data Transfer



### Active to Deep Sleep Transition

Before going to deep sleep ensure that all active communication is complete. This can be done by checking the BUS\_BUSY bit in the I2C\_Status register.

Ensure that the TX and RX FIFOs are empty as any data will be lost during deep sleep.

Before going to deep sleep the clock to the SCB needs to be disabled. This can be done by disabling the clock divider that supplies the clock to the SCB, see the [Clocking System chapter on page 88](#) for more information.

### Deep Sleep to Active Transition

EC\_AM = 1, EC\_OP = 0, EZ Mode.

The system wakes up from Sleep or Deep-Sleep system power modes when an I<sup>2</sup>C address match occurs. The fixed-function I<sup>2</sup>C block performs either of two actions after address match: Address ACK or Address NACK.

- Address ACK: The I<sup>2</sup>C slave executes clock stretching and waits until the device wakes up and ACKs the address.
- Address NACK: The I<sup>2</sup>C 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 mode.

#### Notes:

- The interrupt bit WAKE\_UP (bit 0) of the SCB\_INTR\_I2C\_EC register must be enabled for the I<sup>2</sup>C to wake up the device on slave address match while switching to the Sleep mode
- If the device is configured in I<sup>2</sup>C slave mode, the clock to the SCB should be disabled when entering Deep-Sleep power mode; enable the clock when waking up from Deep-Sleep mode.

## 16.5.7 Clocking and Oversampling

The SCB I<sup>2</sup>C supports both internally and externally clocked operation modes. Two bitfields (EC\_AM\_MODE and EC\_OP\_MODE) in the SCB\_CTRL register determine the SCB clock mode. EC\_AM\_MODE indicates whether I<sup>2</sup>C address matching is internally (0) or externally (1) clocked. I<sup>2</sup>C address matching comprises the first part of the I<sup>2</sup>C protocol. EC\_OP\_MODE indicates whether the rest of the protocol operation (besides I<sup>2</sup>C address matching) is internally (0) or externally (1) clocked. The externally clocked mode of operation is supported only in the I<sup>2</sup>C slave mode.

An internally-clocked operation uses the programmable clock dividers. For I<sup>2</sup>C, an integer clock divider must be used for both master and slave. For more information on system clocking, see the [Clocking System chapter on page 88](#).

The SCB\_CTRL bitfields EC\_AM\_MODE and EC\_OP\_MODE can be configured in the following ways.

- EC\_AM\_MODE is '0' and EC\_OP\_MODE is '0': Use this configuration when only Active mode functionality is required.
  - FIFO mode: Supported.
  - EZ mode: Supported.
- EC\_AM\_MODE is '1' and EC\_OP\_MODE is '0': Use this configuration when both Active and Deep Sleep functionality are required. This configuration relies on the externally clocked functionality for the I<sup>2</sup>C address matching and relies on the internally clocked functionality to access the memory buffer. The “hand over” from external to internal functionality relies either on an ACK/NACK or clock stretching scheme. The former may result in termination of the current transfer and relies on a master retry. The latter stretches the current transfer after a matching address is received. This mode requires the master to support either NACK generation (and retry) or clock stretching. When the I<sup>2</sup>C address is matched, INTR\_I2C\_EC.WAKE\_UP is set to '1'. The associated Deep Sleep functionality interrupt brings the system into Active power mode.
  - FIFO mode: See [“Deep Sleep to Active Transition” on page 159](#).
  - EZ mode: See [“Deep Sleep to Active Transition” on page 161](#).
- EC\_AM\_MODE is '1' and EC\_OP\_MODE is '1'. Use this mode when both Active and Deep Sleep functionality are required. When the slave is selected, INTR\_I2C\_EC.WAKE\_UP is set to '1'. The associated Deep Sleep functionality interrupt brings the system into Active power mode. When the slave is deselected, INTR\_I2C\_EC.EZ\_STOP and/or INTR\_I2C\_EC.EZ\_WRITE\_STOP are set to '1'.
  - FIFO mode: Not supported.
  - EZ mode: Supported.

An externally-clocked operation uses a clock provided by the serial interface. The externally clocked mode does not support FIFO mode. If EC\_OP\_MODE is '1', the external interface logic accesses the memory buffer on the external interface clock (I<sup>2</sup>C SCL). This allows for EZ mode functionality in Active and Deep Sleep power modes.

In Active system power mode, the memory buffer requires arbitration between external interface logic (on I<sup>2</sup>C SCL) and the CPU interface logic (on system peripheral clock). This arbitration always gives the highest priority to the external interface logic (host accesses). The external interface logic takes one serial interface clock/bit periods for the I<sup>2</sup>C. During this period, the internal logic is denied service to the memory buffer. The PSoC 4 MCU provides two programmable options to address this “denial of service”:

- If the BLOCK bitfield of SCB\_CTRL is '1': An internal logic access to the memory buffer is blocked until the memory buffer is granted and the external interface logic has completed access. For a 100-kHz I<sup>2</sup>C interface, the maximum blocking period of one serial interface bit period measures 10 μs (approximately 208 clock cycles on a 48 MHz SCB input clock). This option provides normal SCB register functionality, but the blocking time introduces additional internal bus wait states.
- If the BLOCK bitfield of SCB\_CTRL is '0': An internal logic access to the memory buffer is not blocked, but fails when it conflicts with an external interface logic access. A read access returns the value 0xFFFF:FFFF and a write access is ignored. This option does not introduce additional internal bus wait states, but an access to the memory buffer may not take effect. In this case, following failures are detected:
  - Read Failure: A read failure is easily detected, as the returned value is 0xFFFF:FFFF. This value is unique as non-failing memory buffer read accesses return an unsigned byte value in the range 0x0000:0000-0x0000:00FF.
  - Write Failure: A write failure is detected by reading back the written memory buffer location, and confirming that the read value is the same as the written value.

For both options, a conflicting internal logic access to the memory buffer sets INTR\_TX.BLOCKED field to '1' (for write access-es) and INTR\_RX.BLOCKED field to '1' (for read accesses). These fields can be used as either status fields or as interrupt cause fields (when their associated mask fields are enabled).

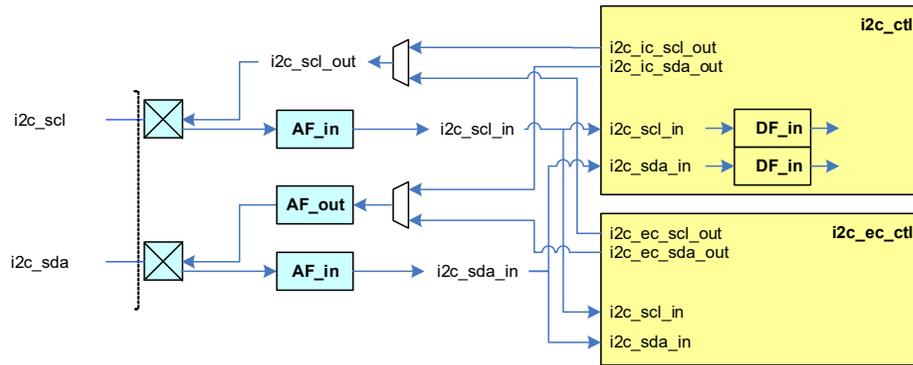
If a series of read or write accesses is performed and CTRL.BLOCKED is '0', a failure is detected by comparing the logical OR of all read values to 0xFFFF:FFFF and checking the INTR\_TX.BLOCKED and INTR\_RX.BLOCKED fields to determine whether a failure occurred for a (series of) write or read operation(s).

### 16.5.7.1 Glitch Filtering

The PSoC 4 SCB I<sup>2</sup>C has analog and digital glitch filters. Analog glitch filters are applied on the i2c\_scl\_in and i2c\_sda\_in input signals (AF\_in) to filter glitches of up to 50 ns. An analog glitch filter is also applied on the i2c\_sda\_out output signal (AF\_out). Analog glitch filters are enabled and disabled in the SCB.I2C\_CFG register. Do not change the \_TRIM bitfields; only change the \_SEL bitfields in this register.

Digital glitch filters (three input median filters) are applied on the i2c\_scl\_in and i2c\_sda\_in input signals (DF\_in). The digital glitch filter is enabled in the SCB.RX\_CTRL register via the MEDIAN bitfield.

Figure 16-39. I<sup>2</sup>C Glitch Filtering Connection



The following table lists the useful combinations of glitch filters.

Table 16-16. Glitch Filter Combinations

| AF_in | AF_out | DF_in | Comments  |
|-------|--------|-------|---|
| 0     | 0      | 1     | Used when operating in internally-clocked mode and in master in fast-mode plus (1-MHz speed mode) |
| 1     | 0      | 0     | Used when operating in internally-clocked mode (EC_OP_MODE is '0')                                |
| 1     | 1      | 0     | Used when operating in externally-clocked mode (EC_OP_MODE is '1'). Only slave mode.              |

When operating in EC\_OP\_MODE = 1, the 100-kHz, 400-kHz, 1000-kHz, and high speed modes require the following settings for AF\_out:

| AF_in | AF_out | DF_in |  |
|-------|--------|-------|--|
| 1     | 1      | 0     | 100-kHz mode: I2C_CFG.SDA_OUT_FILT_SEL = 3<br>400-kHz mode: I2C_CFG.SDA_OUT_FILT_SEL = 3<br>1000-kHz mode: I2C_CFG.SDA_OUT_FILT_SEL = 1<br>1.7-Mbps or 3.4-Mbps mode: I2C_CFG.SDA_OUT_FILT_SEL = 1 |

In High Speed mode after the SCL is '1', HW changes the glitch filter from 50 ns to 10 ns. After the next STOP, HW changes the glitch filter from 10 ns to 50 ns.

### 16.5.7.2 Oversampling and Bit Rate

#### Internally-clocked Master

The PSoC 4 implements the I<sup>2</sup>C clock as an oversampled multiple of the SCB input clock. In master mode, the block determines the I<sup>2</sup>C frequency. Routing delays on the PCB, on the chip, and the SCB (including analog and digital glitch filters) all contribute to the signal interface timing. In master mode, the block operates off `clk_scb` and uses programmable oversampling factors for the SCL high (1) and low (0) times. For high and low phase oversampling, see `I2C_CTRL.LOW_PHASE_OVS` and `I2C_CTRL.HIGH_PHASE_OVS` registers. For simple manipulation of the oversampling factor, see the `SCB_CTRL.OVS` register. In high speed mode, see `I2C_CTRL_HS` register for high and low phase oversampling.

Table 16-17. I<sup>2</sup>C Frequency and Oversampling Requirements in I<sup>2</sup>C Master Mode

| AF_in | AF_out | DF_in | Mode     | Supported Frequency | LOW_PHASE_OVS | HIGH_PHASE_OVS | clk_scb Frequency |
|-------|--------|-------|----------|---------------------|---------------|----------------|-------------------|
| 0     | 0      | 1     | 100 kHz  | [62, 100] kHz       | [9, 15]       | [9, 15]        | [1.98-3.2] MHz    |
|       |        |       | 400 kHz  | [264, 400] kHz      | [13, 5]       | [7, 15]        | [8.45-10] MHz     |
|       |        |       | 1000 kHz | [447, 1000] kHz     | [8, 15]       | [5, 15]        | [14.32-25.8] MHz  |
|       |        |       | 1.7 Mbps | [617, 1700] kHz     | [8, 15]       | [4, 15]        | [20.79-39.2] MHz  |
|       |        |       | 3.4 Mbps | [1263, 3400] kHz    | [8, 15]       | [4, 15]        | [43.8-48] MHz     |
| 1     | 0      | 0     | 100 kHz  | [48, 100] kHz       | [7, 15]       | [7, 15]        | [1.55-3.2] MHz    |
|       |        |       | 400 kHz  | [244, 400] kHz      | [12, 15]      | [7, 15]        | [7.82-10] MHz     |
|       |        |       | 1000 kHz | [495, 1000] kHz     | [6, 15]       | [9, 15]        | [16.15-25.29] MHz |
|       |        |       | 1.7 Mbps | [516, 1700] kHz     | [6, 15]       | [3, 15]        | [17.36-39.2] MHz  |
|       |        |       | 3.4 Mbps | [1206, 3400] kHz    | [8, 15]       | [4, 15]        | [41.83-48] MHz    |

Table 16-17 assumes worst-case conditions on the I<sup>2</sup>C bus. The following equations can be used to determine the settings for your own system. This will involve measuring the rise and fall times on SCL and SDA lines in your system.

$$t_{\text{CLK\_SCB(Min)}} = (t_{\text{LOW}} + t_{\text{F}}) / \text{LOW\_PHASE\_OVS}$$

If `clk_scb` is any faster than this, the  $t_{\text{LOW}}$  of the I<sup>2</sup>C specification will be violated.  $t_{\text{F}}$  needs to be measured in your system.

$$t_{\text{CLK\_SCB(Max)}} = (t_{\text{VD}} - t_{\text{RF}} - 100 \text{ ns}) / 3 \text{ (When analog filter is enabled and digital disabled)}$$

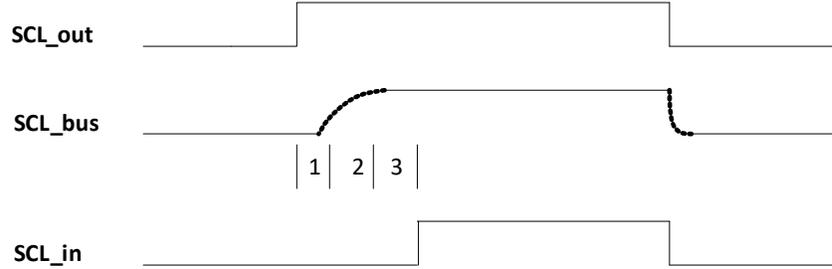
$$t_{\text{CLK\_SCB(Max)}} = (t_{\text{VD}} - t_{\text{RF}}) / 4 \text{ (When analog filter is disabled and analog filter is enabled)}$$

$t_{\text{RF}}$  is the maximum of either the rise or fall time. If `clk_scb` is slower than this frequency,  $t_{\text{VD}}$  will be violated.

#### I<sup>2</sup>C Master Clock Synchronization

The `HIGH_PHASE_OVS` counter does not start counting until the SCB detects that the SCL line is high. This is not the same as when the SCB sets the SCL high. The differences are explained by three delays:

1. Delay from SCB to I/O pin
2. I<sup>2</sup>C bus  $t_{\text{R}}$
3. Input delay (filters and synchronization)

Figure 16-40. I<sup>2</sup>C SCL Turnaround Path


If the above three delays combined are greater than one `clk_scb` cycle, then the high phase of the SCL will be extended. This may cause the actual data rate on the I<sup>2</sup>C bus to be slower than expected. This can be avoided by:

- Decreasing the pull-up resistor, or decreasing the bus capacitance to reduce  $t_R$ .
- Reducing the `I2C_CTRL.HIGH_PHASE_OVS` value.

### Internally-clocked Slave

In slave mode, the I<sup>2</sup>C frequency is determined by the incoming I<sup>2</sup>C SCL signal. To ensure proper operation, `clk_scb` must be significantly higher than the I<sup>2</sup>C bus frequency. Unlike master mode, this mode does not use programmable oversampling factors.

 Table 16-18. SCB Input Clock Requirements in I<sup>2</sup>C Slave Mode

| AF_in | AF_out | DF_in | Mode     | clk_scb Frequency Range |
|-------|--------|-------|----------|-------------------------|
| 0     | 0      | 1     | 100 kHz  | [1.98-12.8] MHz         |
|       |        |       | 400 kHz  | [8.45-17.14] MHz        |
|       |        |       | 1000 kHz | [14.32-44.77] MHz       |
|       |        |       | 1.7 Mbps | [20.38-48] MHz          |
|       |        |       | 3.4 Mbps | [42.94-48] MHz          |
| 1     | 0      | 0     | 100 kHz  | [1.55-12.8] MHz         |
|       |        |       | 400 kHz  | [7.82-15.38] MHz        |
|       |        |       | 1000 kHz | [15.84-89.0] MHz        |
|       |        |       | 1.7 Mbps | [17.02-48] MHz          |
|       |        |       | 3.4 Mbps | [41.01-48] MHz          |

$$t_{CLK\_SCB(Max)} = (t_{VD} - t_{RF} - 100 \text{ ns}) / 3 \text{ (When analog filter is enabled and digital disabled)}$$

$$t_{CLK\_SCB(Max)} = (t_{VD} - t_{RF}) / 4 \text{ (When analog filter is disabled and analog filter is enabled)}$$

$t_{RF}$  is the maximum of either the rise or fall time. If `clk_scb` is slower than this frequency,  $t_{VD}$  will be violated.

The minimum period of `clk_scb` is determined by one of the following equations:

$$t_{CLK\_SCB(MIN)} = (t_{SU;DAT(min)} + t_{RF}) / 16$$

or

$$t_{CLK\_SCB(MIN)} = (0.6 * t_F - 50 \text{ ns}) / 2 \text{ (When analog filter is enabled and digital disabled)}$$

$$t_{CLK\_SCB(MIN)} = (0.6 * t_F) / 3 \text{ (When analog filter is disabled and digital enabled)}$$

The result that yields the largest period from the two sets of equations above should be used to set the minimum period of `clk_scb`.

## Master-Slave

To configure the I<sup>2</sup>C for master-slave mode, write '1' to the MASTER\_MODE(bit-31) and SLAVE\_MODE(bit-30) of the I2C\_CTRL register. When the I<sup>2</sup>C initializes a transfer, it is a master and when it is addressed by another master it is a slave. In this mode, when the SCB is acting as a master device, the block determines the I<sup>2</sup>C frequency. When the SCB is acting as a slave device, the block does not determine the I<sup>2</sup>C frequency. Instead, the incoming I<sup>2</sup>C SCL signal does.

To guarantee operation in both master and slave modes, choose clock frequencies that work for both master and slave using the tables above.

**Note** The I<sup>2</sup>C cannot support wakeup from deepsleep power modes in fast-plus mode (1 MHz), when configured in master-slave mode.

## 16.5.8 Enabling and Initializing the I<sup>2</sup>C

The following section describes the method to configure the I<sup>2</sup>C block for standard (non-EZ) mode and EZI2C mode.

### 16.5.8.1 Configuring for I<sup>2</sup>C FIFO Mode

The I<sup>2</sup>C interface must be programmed in the following order.

1. Program protocol specific information using the SCB\_I2C\_CTRL register. This includes selecting master - slave functionality.
2. Program the generic transmitter and receiver information using the SCB\_TX\_CTRL and SCB\_RX\_CTRL registers.
3. Set the SCB\_CTRL.BYTE\_MODE to '1' to enable the byte mode.
4. Program the SCB\_CTRL register to enable the I<sup>2</sup>C block and select the I<sup>2</sup>C mode. For a complete description of the I<sup>2</sup>C registers, see the *PSoC 4100S Max: PSoc 4 Registers TRM*.

### 16.5.8.2 Configuring for EZ Mode

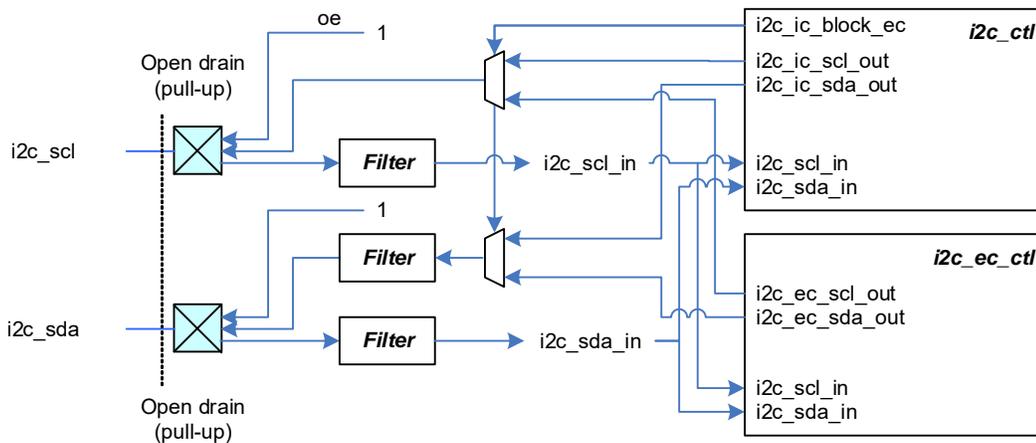
To configure the I<sup>2</sup>C block for EZ mode, set the following I<sup>2</sup>C register bits:

1. Select the EZI2C mode by writing '1' to the EZ\_MODE bit (bit 10) of the SCB\_CTRL register.
2. Set the S\_READY\_ADDR\_ACK (bit 12) and S\_READY\_DATA\_ACK (bit 13) bits of the SCB\_I2C\_CTRL register.

**Note:** For all modes clk\_scb must also be configured. For information on configuring a peripheral clock and connecting it to the SCB consult the [Clocking System chapter on page 88](#).

The GPIO must also be connected to the SCB; see the following section for more details.

## 16.5.9 I/O Pad Connections

 Figure 16-41. I<sup>2</sup>C I/O Pad Connections

 Table 16-19. I<sup>2</sup>C I/O Pad Descriptions

| I/O Pads | Drive Mode                       | On-chip I/O Signals | Usage            |
|----------|----------------------------------|---------------------|------------------|
| i2c_scl  | Open drain with external pull-up | i2c_scl_in          | Receive a clock  |
|          |                                  | i2c_scl_out         | Transmit a clock |
| i2c_sda  | Open drain with external pull-up | i2c_sda_in          | Receive data     |
|          |                                  | i2c_sda_out         | Transmit data    |

When configuring the I<sup>2</sup>C SDA/SCL lines, the following sequence must be followed. If this sequence is not followed, the I<sup>2</sup>C lines may initially have overshoot and undershoot.

1. Set SCB\_CTRL\_MODE to '0'.
2. Configure HSIOM for SCL and SDA to connect to the SCB.
3. Set TX\_CTRL.OPEN\_DRAIN to '1'.
4. Configure I<sup>2</sup>C pins for high-impedance drive mode.
5. Configure SCB for I<sup>2</sup>C
6. Enable SCB
7. Configure I<sup>2</sup>C pins for Open Drain Drives Low.

## 16.5.10 I<sup>2</sup>C Registers

The I<sup>2</sup>C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in [Table 16-20](#).

Table 16-20. I<sup>2</sup>C Registers

| Register               | Function  |
|------------------------|---|
| SCB_CTRL               | Enables the SCB block and selects the type of serial interface (SPI, UART, I <sup>2</sup> C). 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_I2C_STRETCH_CTRL   | Specifies the stretch threshold, which is SCL turnaround delay in number of clk_scb cycles.   |
| SCB_I2C_STRETCH_STATUS | Indicates the turnaround count, stretch detection, and synchronization status.  |
| SCB_I2C_CTRL_HS        | Enables the I <sup>2</sup> C high speed mode.   |
| 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 I<sup>2</sup>C register bits are available in the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

## 16.6 SCB Interrupts

SCB supports interrupt generation on various events. The interrupts generated by the SCB block vary depending on the mode of operation.

Table 16-21. SCB Interrupts

| Interrupt        | Functionality   | Active/Deep Sleep | Registers   |
|------------------|---|-------------------|---|
| interrupt_master | I <sup>2</sup> C master and SPI master functionality    | Active            | INTR_M,<br>INTR_M_SET,<br>INTR_M_MASK,<br>INTR_M_MASKED     |
| interrupt_slave  | I <sup>2</sup> C slave and SPI slave functionality      | Active            | INTR_S,<br>INTR_S_SET,<br>INTR_S_MASK,<br>INTR_S_MASKED     |
| interrupt_tx     | UART transmitter and TX FIFO functionality              | Active            | INTR_TX,<br>INTR_TX_SET,<br>INTR_TX_MASK,<br>INTR_TX_MASKED |
| interrupt_rx     | UART receiver and RX FIFO functionality                 | Active            | INTR_RX,<br>INTR_RX_SET,<br>INTR_RX_MASK,<br>INTR_RX_MASKED |
| interrupt_i2c_ec | Externally clocked I <sup>2</sup> C slave functionality | Deep Sleep        | INTR_I2C_EC,<br>INTR_I2C_EC_MASK,<br>INTR_I2C_EC_MASKED     |
| interrupt_spi_ec | Externally clocked SPI slave functionality              | Deep Sleep        | INTR_ISPI_EC,<br>INTR_SPI_EC_MASK,<br>INTR_SPI_EC_MASKED    |

**Note:** To avoid being triggered by events from previous transactions, whenever the firmware enables an interrupt mask register bit, it should clear the interrupt request register in advance.

**Note:** If the DMA is used to read data out of RX FIFO, the NOT\_EMPTY interrupt may never trigger. This can occur when clk\_peri (clocking DMA) is running much faster than the clock to the SCB. As a workaround to this issue, set the RX\_FIFO\_CTRL.TRIGGER\_LEVEL to '1' (not 0); this will allow the interrupt to fire.

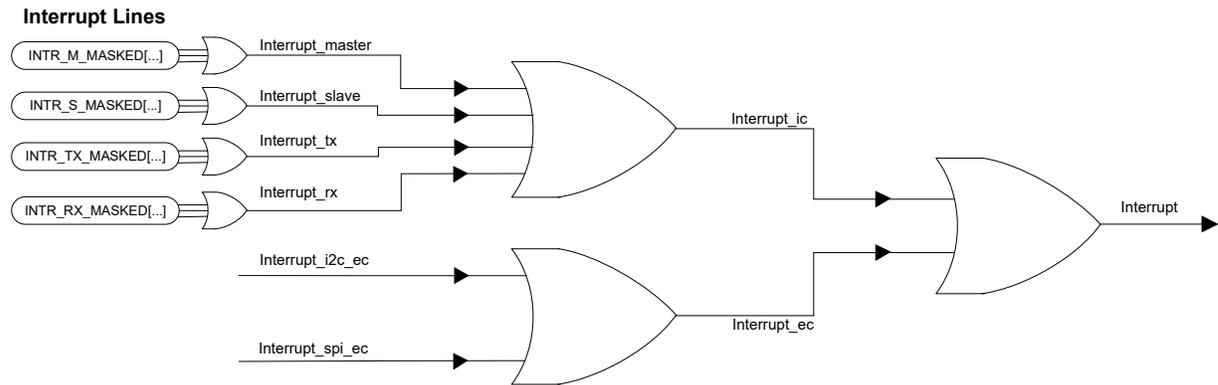
The following register definitions correspond to the SCB interrupts:

- **INTR\_M:** This register provides the instantaneous status of the interrupt sources. A write of '1' to a bit will clear the interrupt.
- **INTR\_M\_SET:** A write of '1' into this register will set the interrupt.
- **INTR\_M\_MASK:** The bit in this register masks the interrupt sources. Only the interrupt sources with their masks enabled can trigger the interrupt.
- **INTR\_M\_MASKED:** This register provides the instantaneous value of the interrupts after they are masked. It provides logical and corresponding request and mask bits. This is used to understand which interrupt triggered the event.

**Note:** While registers corresponding to INTR\_M are used here, these definitions can be used for INTR\_S, INTR\_TX, INTR\_RX, INTR\_I2C\_EC, and INTR\_SPI\_EC.

Figure 16-42 shows the physical interrupt lines. All the interrupts are OR'd together to make one interrupt source that is the OR of all six individual interrupts. All the externally-clocked interrupts make one interrupt line called `interrupt_ec`, which is the OR'd signal of `interrupt_i2c_ec` and `interrupt_spi_ec`. All the internally-clocked interrupts make one interrupt line called `interrupt_ic`, which is the OR'd signal of `interrupt_master`, `interrupt_slave`, `interrupt_tx`, and `interrupt_rx`. The Active functionality interrupts are generated synchronously to `clk_peri` while the Deep Sleep functionality interrupts are generated asynchronously to `clk_peri`.

Figure 16-42. Interrupt Lines



### 16.6.1 SPI Interrupts

The SPI interrupts can be classified as master interrupts, slave interrupts, TX interrupts, RX interrupts, and externally clocked (EC) mode interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, 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 SCB also provides an interrupt cause register (SCB\_INTR\_CAUSE) that can be used to determine interrupt source. The interrupt registers are cleared by writing '1' to the corresponding bitfield. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX\_FIFO\_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on interrupt registers, see the *PSoC 4100S Max: PSoC 4 Registers TRM*. The SPI supports interrupts on the following events:

- SPI Master Interrupts
  - SPI master transfer done – All data from the TX FIFO are sent. This interrupt source triggers later than TX\_FIFO\_EMPTY by the amount of time it takes to transmit a single data element. TX\_FIFO\_EMPTY triggers when the last data element from the TX FIFO goes to the shifter register. However, SPI Done triggers after this data element is transmitted. This means SPI Done will be asserted one SCLK clock cycle earlier than the completion of data element reception.
- SPI Slave Interrupts
  - SPI Bus Error – Slave deselected at an unexpected time in the SPI transfer. The firmware may decide to clear the TX and RX FIFOs for this error.
  - SPI slave deselected after any EZSPI transfer occurred.
  - SPI slave deselected after a write EZSPI transfer occurred.
- SPI TX
  - TX FIFO has less entries than the value specified by TRIGGER\_LEVEL in SCB\_TX\_FIFO\_CTRL.
  - TX FIFO not full – At least one data element can be written into the TX FIFO.
  - TX FIFO empty – The TX FIFO is empty.
  - TX FIFO overflow – Firmware attempts to write to a full TX FIFO.
  - TX FIFO underflow – Hardware attempts to read from an empty TX FIFO. This happens when the SCB is ready to transfer data and EMPTY is '1'.
  - TX FIFO trigger – Less entries in the TX FIFO than the value specified by TX\_FIFO\_CTRL.TRIGGER\_LEVEL.

- SPI RX
  - RX FIFO has more entries than the value specified by TRIGGER\_LEVEL in SCB\_RX\_FIFO\_CTRL.
  - RX FIFO full - RX FIFO is full.
  - RX FIFO not empty - RX FIFO is not empty. At least one data element is available in the RX FIFO to be read.
  - RX FIFO overflow - Hardware attempt to write to a full RX FIFO.
  - RX FIFO underflow - Firmware attempts to read from an empty RX FIFO.
  - RX FIFO trigger - More entries in the RX FIFO than the value specified by RX\_FIFO\_CTRL.TRIGGER\_LEVEL.
- SPI Externally Clocked
  - Wake up request on slave select – Active on incoming slave request (with address match). Only set when EC\_AM is '1'.
  - SPI STOP detection at the end of each transfer – Activated at the end of every transfer (I2C STOP). Only set for a slave request with an address match, in EZ mode, when EC\_OP is '1'.
  - SPI STOP detection at the end of a write transfer – Activated at the end of a write transfer (I2C STOP). This event is an indication that a buffer memory location has been written to. For EZ mode, a transfer that only writes the base address does not activate this event. Only set for a slave request with an address match, in EZ mode, when EC\_OP is '1'.
  - SPI STOP detection at the end of a read transfer – Activated at the end of a read transfer (I2C STOP). This event is an indication that a buffer memory location has been read from. Only set for a slave request with an address match, in EZ mode when EC\_OP is '1'.

Figure 16-43 and Figure 16-44 show how each of the interrupts are triggered. Figure 16-43 shows the TX buffer and the corresponding interrupts while Figure 16-44 shows all the corresponding interrupts for the RX buffer. The FIFO has 32 bytes split into 16 bytes for TX and 16 bytes for RX instead of the 8 bytes shown below.

Figure 16-43. TX Interrupt Source Operation

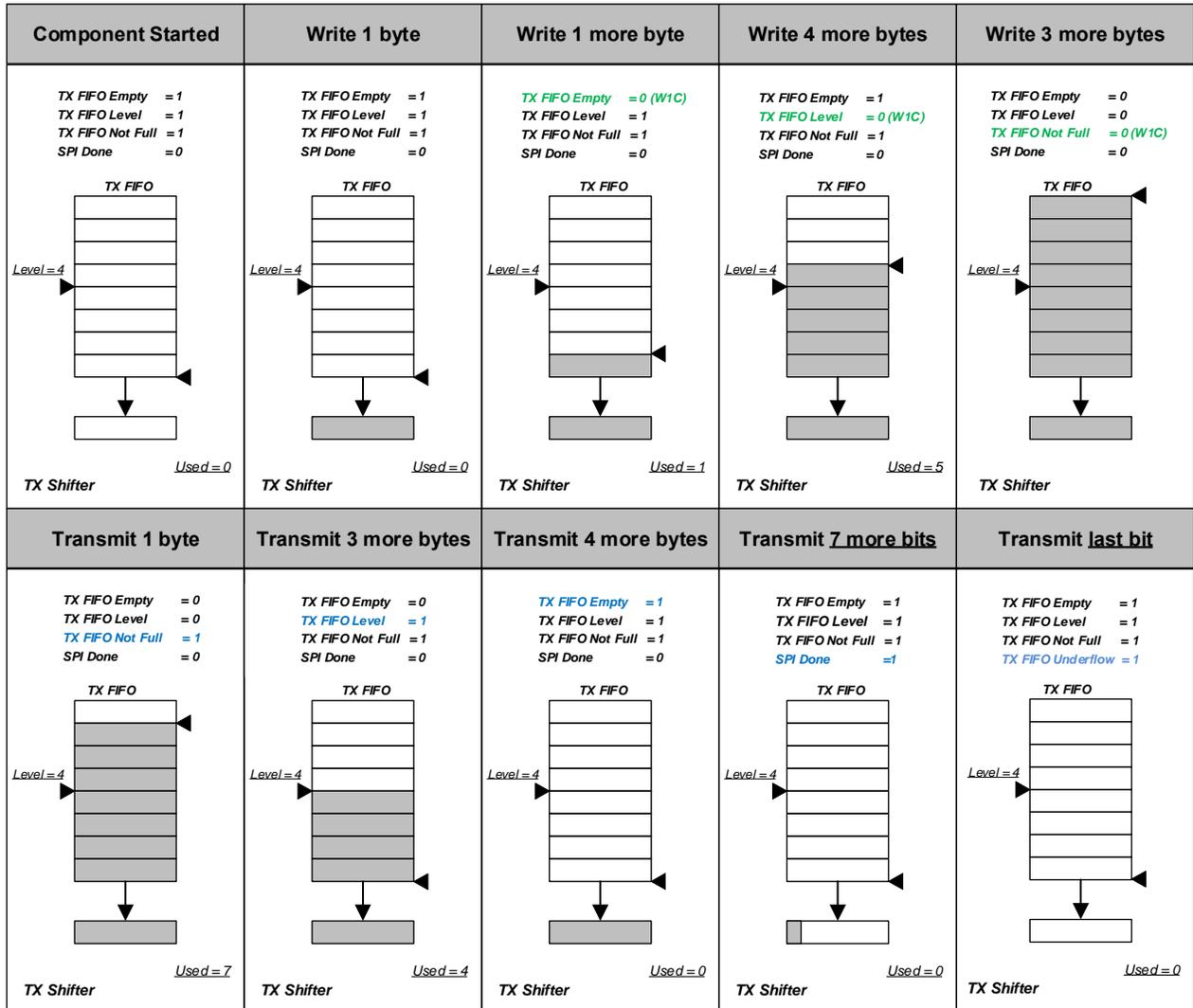
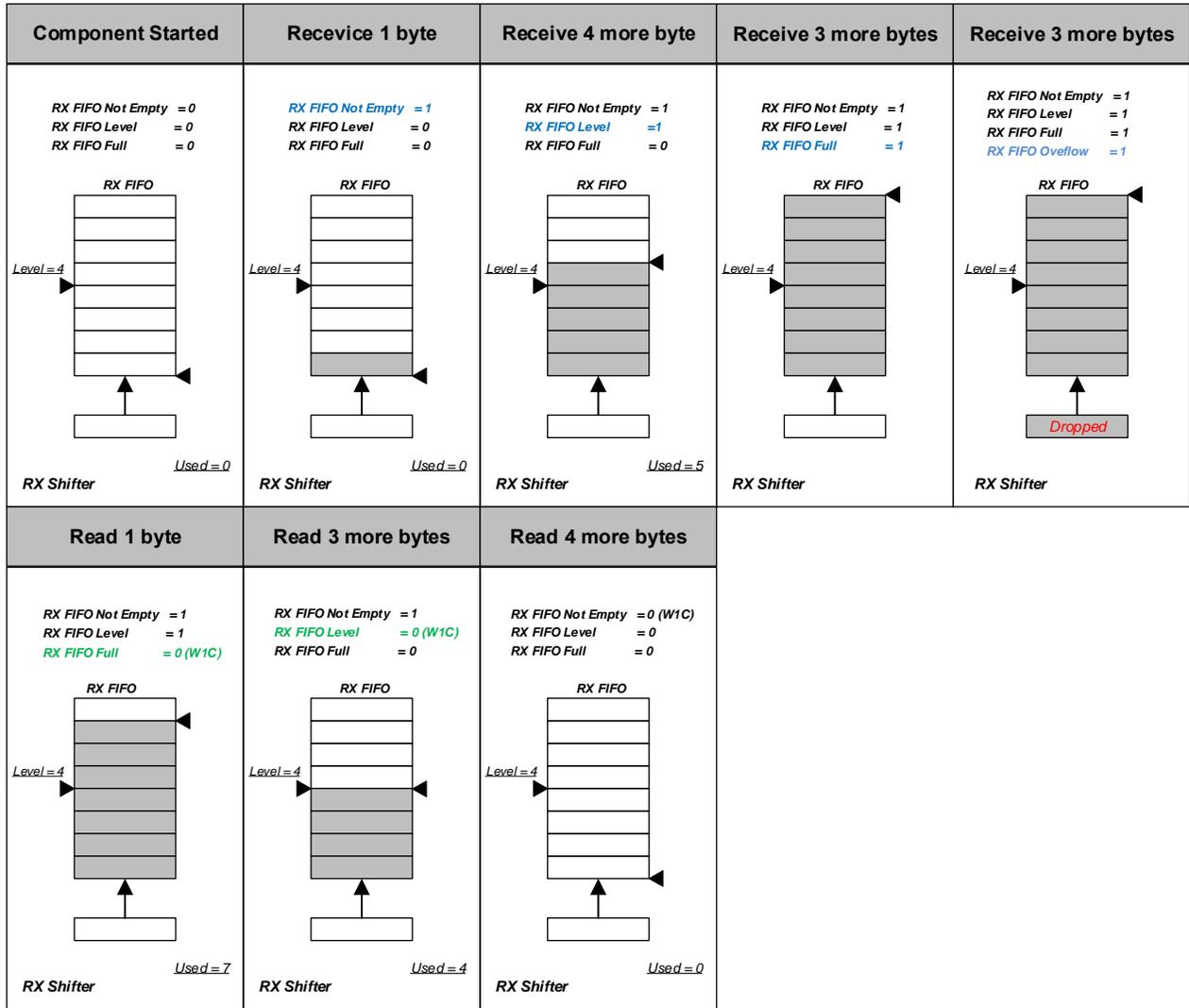


Figure 16-44. RX Interrupt Source Operation



## 16.6.2 UART Interrupts

The UART interrupts can be classified as TX interrupts and RX interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, 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 SCB also provides an interrupt cause register (SCB\_INTR\_CAUSE) that can be used to determine interrupt source. The interrupt registers are cleared by writing '1' to the corresponding bitfield. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX\_FIFO\_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on interrupt registers, see the [PSoC 4100S Max: PSoC 4 Registers TRM](#). The UART block generates interrupts on the following events:

- UART TX
  - TX FIFO has fewer entries than the value specified by TRIGGER\_LEVEL in SCB\_TX\_FIFO\_CTRL.
  - TX FIFO not full – TX FIFO is not full. At least one data element can be written into the TX FIFO.
  - TX FIFO empty – The TX FIFO is empty.
  - TX FIFO overflow – Firmware attempts to write to a full TX FIFO.
  - TX FIFO underflow – Hardware attempts to read from an empty TX FIFO. This happens when the SCB is ready to transfer data and EMPTY is '1'.
  - TX NACK – UART transmitter receives a negative acknowledgment in SmartCard mode.
  - TX done – This happens when the UART completes transferring all data in the TX FIFO and the last stop field is transmitted (both TX FIFO and transmit shifter register are empty).
  - TX lost arbitration – The value driven on the TX line is not the same as the value observed on the RX line. This condition event is useful when transmitter and receiver share a TX/RX line. This is the case in LIN or SmartCard modes.
- UART RX
  - RX FIFO has more entries than the value specified by TRIGGER\_LEVEL in SCB\_RX\_FIFO\_CTRL.
  - RX FIFO full – RX FIFO is full. Note that received data frames are lost when the RX FIFO is full.
  - RX FIFO not empty – RX FIFO is not empty.
  - RX FIFO overflow – Hardware attempts to write to a full RX FIFO.
  - RX FIFO underflow – Firmware attempts to read from an empty RX FIFO.
  - Frame error in received data frame – UART frame error in received data frame. This can be either a start of stop bit error:
    - Start bit error:** After the beginning of a start bit period is detected (RX line changes from 1 to 0), the middle of the start bit period is sampled erroneously (RX line is '1'). **Note:** A start bit error is detected before a data frame is received.
    - Stop bit error:** The RX line is sampled as '0', but a '1' was expected. A stop bit error may result in failure to receive successive data frames. **Note:** A stop bit error is detected after a data frame is received.
  - Parity error in received data frame – If UART\_RX\_CTL.DROP\_ON\_PARITY\_ERROR is '1', the received frame is dropped. If UART\_RX\_CTL.DROP\_ON\_PARITY\_ERROR is '0', the received frame is sent to the RX FIFO. In SmartCard sub mode, negatively acknowledged data frames generate a parity error. Note that firmware can only identify the erroneous data frame in the RX FIFO if it is fast enough to read the data frame before the hardware writes a next data frame into the RX FIFO.
  - LIN baud rate detection is completed – The receiver software uses the UART\_RX\_STATUS.BR\_COUNTER value to set the clk\_scb to guarantee successful receipt of the first LIN data frame (Protected Identifier Field) after the synchronization byte.
  - LIN break detection is successful – The line is '0' for UART\_RX\_CTRL.BREAK\_WIDTH + 1 bit period. Can occur at any time to address unanticipated break fields; that is, "break-in-data" is supported. This feature is supported for the UART standard and LIN submodes. For the UART standard submodes, ongoing receipt of data frames is not affected; firmware is expected to take proper action. For the LIN submode, possible ongoing receipt of a data frame is stopped and the (partially) received data frame is dropped and baud rate detection is started. Set to '1', when event is detected. Write with '1' to clear bit.

Figure 16-45 and Figure 16-46 show how each of the interrupts are triggered. Figure 16-45 shows the TX buffer and the corresponding interrupts while Figure 16-46 shows all the corresponding interrupts for the RX buffer. The FIFO has 32 bytes split into 16 bytes for TX and 16 bytes for RX instead of the 8 bytes shown below.

Figure 16-45. TX Interrupt Source Operation

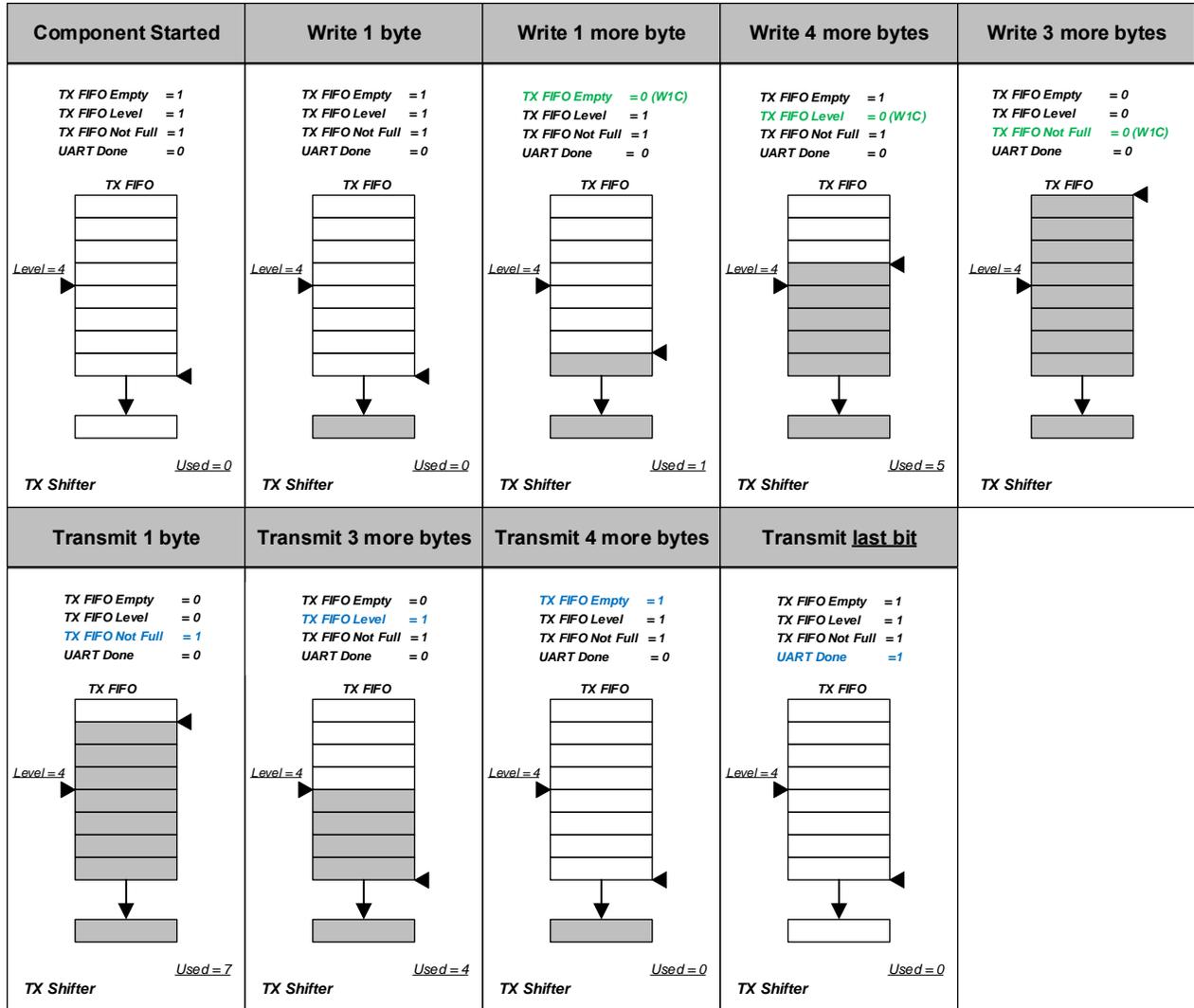
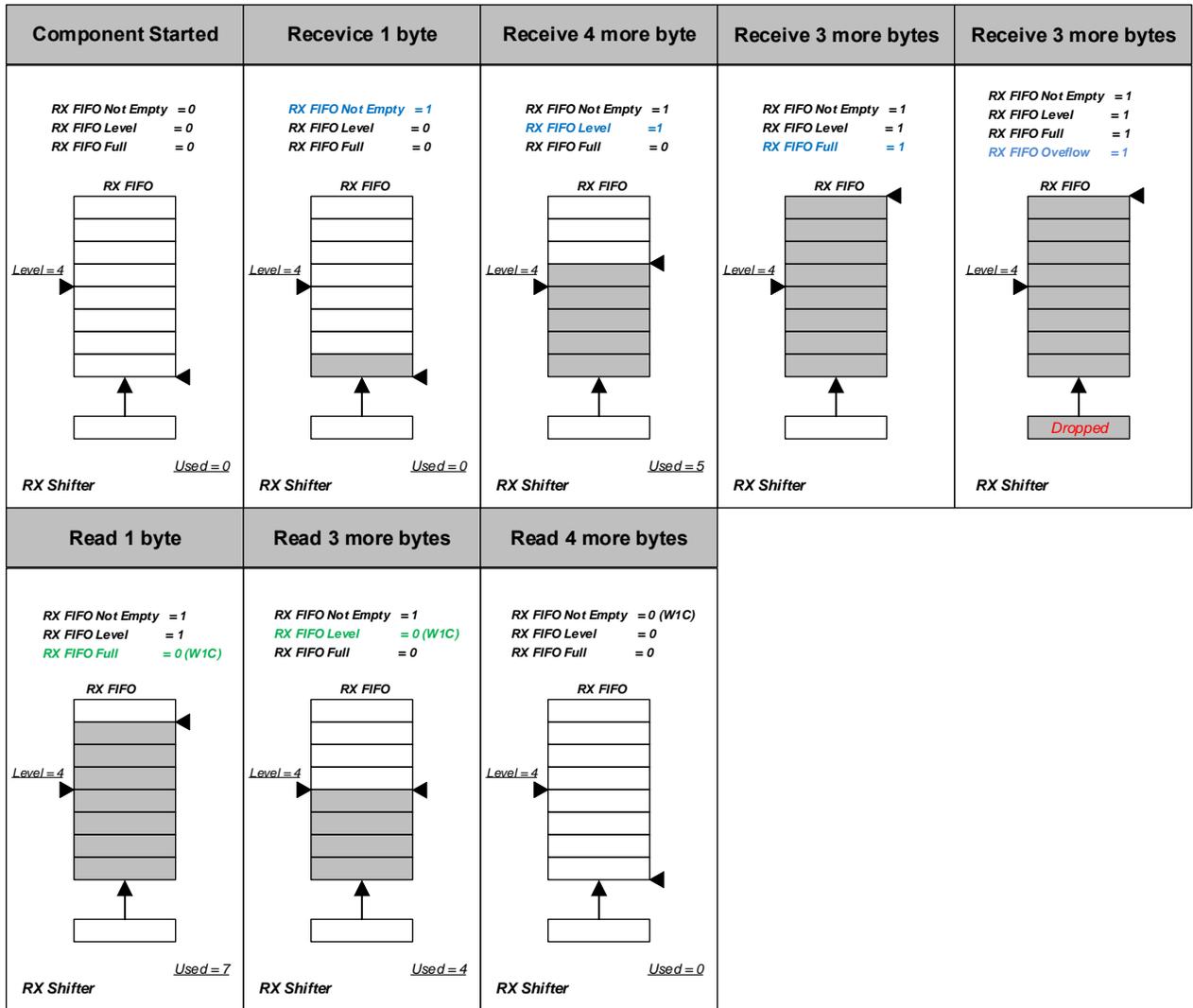


Figure 16-46. RX Interrupt Source Operation



### 16.6.3 I<sup>2</sup>C Interrupts

I<sup>2</sup>C interrupts can be classified as master interrupts, slave interrupts, TX interrupts, RX interrupts, and externally clocked (EC) mode interrupts. Each interrupt output is the logical OR of the group of all possible interrupt sources classified under the section. For example, 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 SCB also provides an interrupt cause register (SCB\_INTR\_CAUSE) that can be used to determine interrupt source. The interrupt registers are cleared by writing '1' to the corresponding bitfield. Note that certain interrupt sources are triggered again as long as the condition is met even if the interrupt source was cleared. For example, the TX\_FIFO\_EMPTY is set as long as the transmit FIFO is empty even if the interrupt source is cleared. For more information on interrupt registers, see the *PSoC 4100S Max: PSoC 4 Registers TRM*. The I<sup>2</sup>C block generates interrupts for the following conditions.

#### ■ I<sup>2</sup>C Master

- I<sup>2</sup>C master lost arbitration – The value driven by the master on the SDA line is not the same as the value observed on the SDA line.
- I<sup>2</sup>C master received NACK – When the master receives a NACK (typically after the master transmitted the slave address or TX data).
- I<sup>2</sup>C master received ACK – When the master receives an ACK (typically after the master transmitted the slave address or TX data).
- I<sup>2</sup>C master sent STOP – When the master has transmitted a STOP.
- I<sup>2</sup>C bus error – Unexpected stop/start condition is detected.
- I<sup>2</sup>C high speed enter – Entered I<sup>2</sup>C high speed mode, at time t1, SCL falling edge after “START, 8-bit master code (0000\_1XXX), NACK” sequence.
- I<sup>2</sup>C high speed exit – Exited I<sup>2</sup>C high speed mode, after STOP detection.

#### ■ I<sup>2</sup>C Slave

- I<sup>2</sup>C slave lost arbitration – The value driven on the SDA line is not the same as the value observed on the SDA line (while the SCL line is '1'). This should not occur; it represents erroneous I<sup>2</sup>C bus behavior. In case of lost arbitration, the I<sup>2</sup>C slave state machine aborts the ongoing transfer. Software may decide to clear the TX and RX FIFOs in case of this error.
- I<sup>2</sup>C slave received NACK – When the slave receives a NACK (typically after the slave transmitted TX data).
- I<sup>2</sup>C slave received ACK – When the slave receives an ACK (typically after the slave transmitted TX data).
- I<sup>2</sup>C slave received STOP – I<sup>2</sup>C STOP event for I<sup>2</sup>C (read or write) transfer intended for this slave (address matching is performed). When STOP or REPEATED START event is detected. The REPEATED START event is included in this interrupt cause such that the I<sup>2</sup>C transfers separated by a REPEATED START can be distinguished and potentially treated separately by the firmware. Note that the second I<sup>2</sup>C transfer (after a REPEATED START) may be to a different slave address.  
The event is detected on any I<sup>2</sup>C transfer intended for this slave. Note that an I<sup>2</sup>C address intended for the slave (address matches) will result in an I2C\_STOP event independent of whether the I<sup>2</sup>C address is ACK'd or NACK'd.
- I<sup>2</sup>C slave received START – When START or REPEATED START event is detected. In the case of an externally-clocked address matching (CTRL.EC\_AM\_MODE is '1') and clock stretching is performed (until the internally-clocked logic takes over) (I2C\_CTRL.S\_NOT\_READY\_ADDR\_NACK is '0'), this field is not set. Firmware should use INTR\_S\_EC.WAKE\_UP, INTR\_S.I2C\_ADDR\_MATCH, and INTR\_S.I2C\_GENERAL.
- I<sup>2</sup>C slave address matched – I<sup>2</sup>C slave matching address received. If CTRL.ADDR\_ACCEPT, the received address (including the R/W bit) is available in the RX FIFO. In the case of externally-clocked address matching (CTRL.EC\_AM\_MODE is '1') and internally-clocked operation (CTRL.EC\_OP\_MODE is '0'), this field is set when the event is detected.
- I<sup>2</sup>C bus error – Unexpected STOP/START condition is detected
- I<sup>2</sup>C restart – When repeated start event is detected
- I<sup>2</sup>C high speed enter – Entered I<sup>2</sup>C high speed mode, at time t1, SCL falling edge after “START, 8-bit master code (0000\_1XXX), NACK” sequence.
- I<sup>2</sup>C high speed exit – Exited I<sup>2</sup>C high speed mode, after STOP detection.

- I<sup>2</sup>C TX
  - TX trigger – TX FIFO has fewer entries than the value specified by TRIGGER\_LEVEL in SCB\_TX\_FIFO\_CTRL.
  - TX FIFO not full – At least one data element can be written into the TX FIFO.
  - TX FIFO empty – The TX FIFO is empty.
  - TX FIFO overflow – Firmware attempts to write to a full TX FIFO.
  - TX FIFO underflow – Hardware attempts to read from an empty TX FIFO.
- I<sup>2</sup>C RX
  - RX FIFO has more entries than the value specified by TRIGGER\_LEVEL in SCB\_RX\_FIFO\_CTRL.
  - RX FIFO is full – The RX FIFO is full.
  - RX FIFO is not empty – At least one data element is available in the RX FIFO to be read.
  - RX FIFO overflow – Hardware attempts to write to a full RX FIFO.
  - RX FIFO underflow – Firmware attempts to read from an empty RX FIFO.
- I<sup>2</sup>C Externally Clocked
  - Wake up request on address match – Active on incoming slave request (with address match). Only set when EC\_AM is '1'.
  - I<sup>2</sup>C STOP detection at the end of each transfer – Only set for a slave request with an address match, in EZ mode, when EC\_OP is '1'.
  - I<sup>2</sup>C STOP detection at the end of a write transfer – Activated at the end of a write transfer (I<sup>2</sup>C STOP). This event is an indication that a buffer memory location has been written to. For EZ mode, a transfer that only writes the base address does not activate this event. Only set for a slave request with an address match, in EZ mode, when EC\_OP is '1'.
  - I<sup>2</sup>C STOP detection at the end of a read transfer – Activated at the end of a read transfer (I<sup>2</sup>C STOP). This event is an indication that a buffer memory location has been read from. Only set for a slave request with an address match, in EZ mode, when EC\_OP is '1'.

# 17. CAN FD Controller



## 17.1 Overview

The Controller Area Network Flexible Data-rate (CAN FD) controller complies with the ISO11898-1 (CAN specification Rev. 2.0 parts A and B). In addition, it supports the Time-Triggered CAN (TTCAN) protocol defined in ISO 11898-4.

All message handling functions are implemented by the RX and TX handlers. The RX handler manages message acceptance filtering, transfer of received messages from the CAN core to a message RAM, and receive message status information. The TX handler transfers transmit messages from the message RAM to the CAN core and provides transmit status information.

A separate clock is provided to the CAN FD controller: CAN FD peripheral clock (canfd.clock\_can[0]) for CAN operation. Acceptance filtering is implemented by a combination of up to 192 filter elements, where each can be configured as a range, as a bit mask, or as a dedicated ID filter.

The CAN FD controller functions only in Active and Sleep power modes. In DeepSleep mode, it is not functional but is fully retained except the Shared Time Stamp (TS) counter.

### 17.1.1 Features

The CAN FD controller has the following features:

- Flexible data-rate (FD) (ISO 11898-1: 2015)
  - Up to 64 data bytes per message
  - Maximum 8 Mbps supported
- Time-Triggered (TT) communication on CAN (ISO 11898-4: 2004)
  - TTCAN protocol level 1 and level 2 completely in hardware
- AUTOSAR support
- Acceptance filtering
- Two configurable receive FIFOs
- Up to 64 dedicated receive buffers
- Up to 32 dedicated transmit buffers
- Configurable transmit FIFO
- Configurable transmit queue
- Configurable transmit event FIFO
- Programmable loop-back test mode
- Power-down support
- Shared message RAM
- Receive FIFO top pointer logic
  - Enables DMA access on the FIFO
- DMA for debug message and received FIFOs
- Shared time stamp counter

**Note:** Refer to the device datasheet to find the supported number of M\_TTCAN groups, M\_TTCAN channels in each group, and total message RAM allocated to each group.

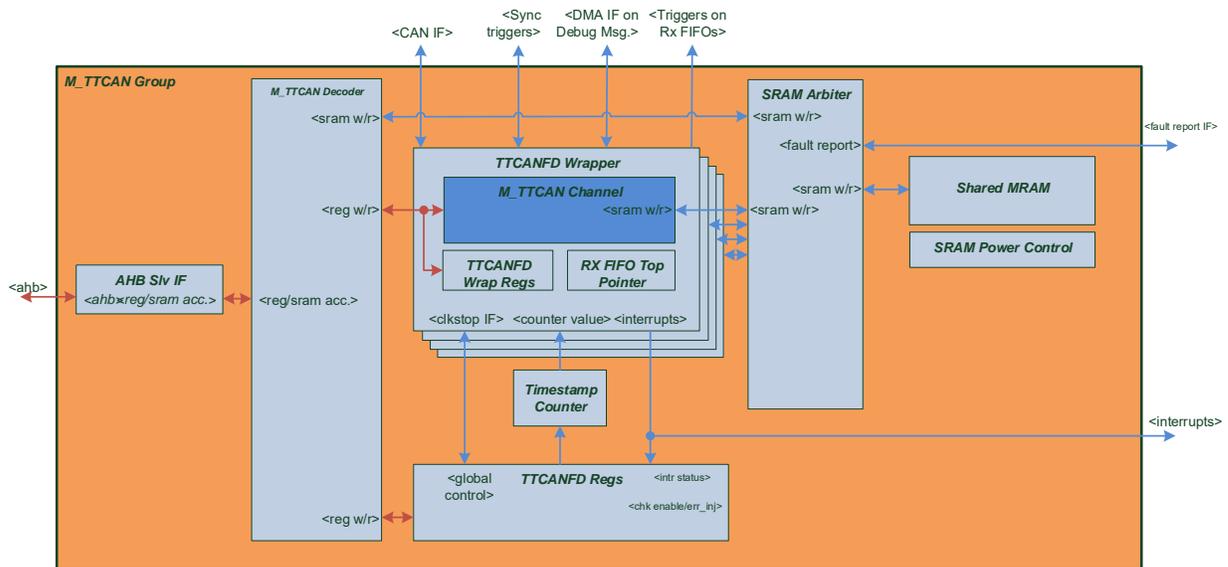
### 17.1.2 Features Not Supported

- Asynchronous serial communication (ASC)
- Interrupt of Bit Error Corrected (CANFDx\_CHy\_IR.BEC) in M\_TTCAN
  - This bit is fixed at '0'.

## 17.2 Configuration

### 17.2.1 Block Diagram

Figure 17-1. M\_TTCAN Block Diagram



### 17.2.2 Clock Sources

The CAN FD peripheral clock (canfd.clock\_can[0]) is used for the CAN (or CAN FD) operation. For the CAN FD operation, it is recommended to use 20 MHz or 40 MHz for frequency clock. See the [Clocking System chapter on page 88](#) for more details about clock configuration.

### 17.2.3 Interrupt Lines

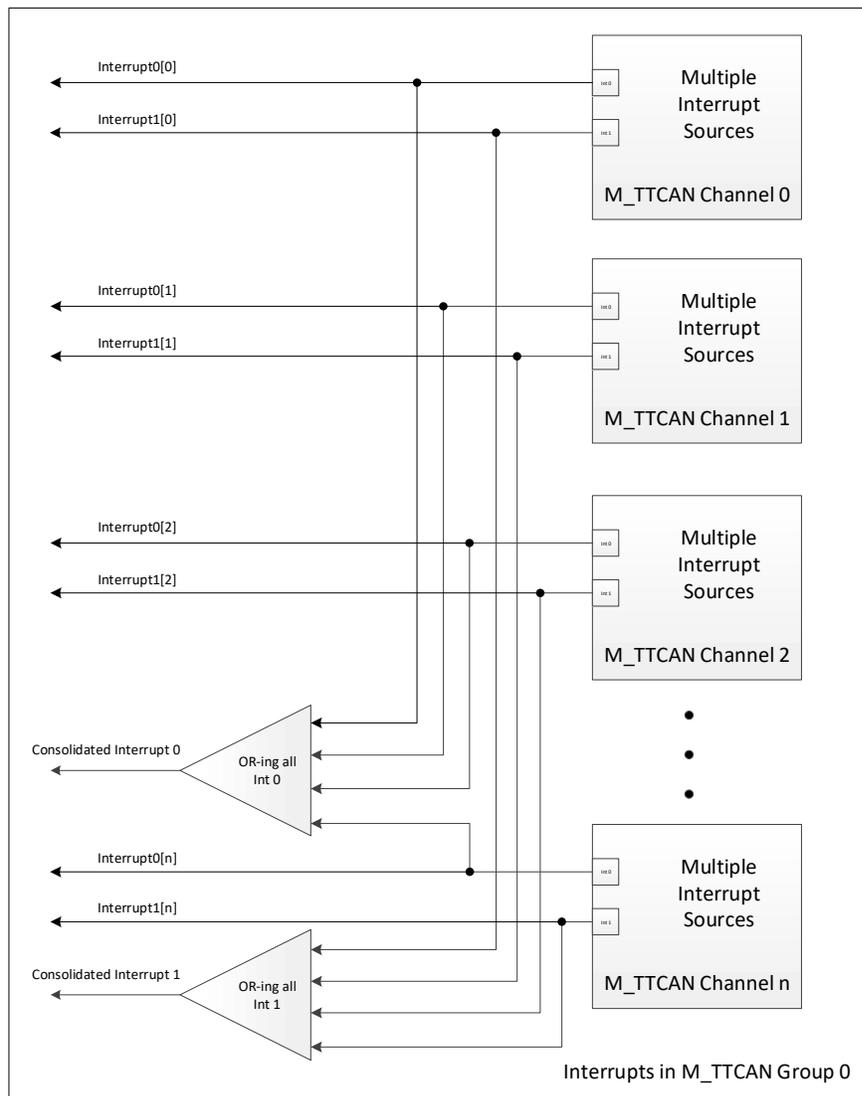
The two kind of interrupts from the M\_TTCAN group are as follows:

- Interrupt0 and interrupt1 from each M\_TTCAN channel within the M\_TTCAN group
- Consolidated interrupt0 and consolidated interrupt1 for one M\_TTCAN group

Each M\_TTCAN channel provides two interrupt lines: interrupt0 and interrupt1. Interrupts from any source within the M\_TTCAN channel can be routed either to interrupt0 or interrupt1 by using CANFDx\_CHy\_ILS and CANFDx\_CHy\_TTILS registers. By default, all interrupts are routed to interrupt0. By programming Enable Interrupt Line 0 (CANFDx\_CHy\_ILE.EINT0) and Enable Interrupt Line 1 (CANFDx\_CHy\_ILE.EINT1), the interrupt lines can be enabled or disabled separately for each interrupt source.

In the PSoC device, one device may contain multiple M\_TTCAN channels in one M\_TTCAN instance. Therefore, Interrupt line 0 and Interrupt line 1 from each M\_TTCAN channel are routed to a common interrupt0 and interrupt1. Common interrupt0 and interrupt1 are ORed of all interrupt0 and interrupt1 coming from all present channels within one M\_TTCAN group. Interrupt cause registers CANFDx\_INTR0\_CAUSE and CANFDx\_INTR1\_CAUSE provide information about the active interrupt causing channel from a particular group.

Figure 17-2. Interrupts in M\_TTCAN Group



## 17.3 Functional Description

### 17.3.1 Operation Modes

#### 17.3.1.1 Software Initialization

This refers to setting or resetting the initialization bit (CANFDx\_CHy\_CCCR.INIT). The CANFDx\_CHy\_CCCR.INIT bit is set

- either by software or hardware reset
- when an uncorrected bit error is detected in message RAM
- by going Bus Off

While the CANFDx\_CHy\_CCCR.INIT is set:

- message transfer from and to the CAN bus is stopped
- the status of the CAN bus output canfd.tcan\_tx is recessive (high)
- the protocol error counters are unchanged

Setting CANFDx\_CHy\_CCCR.INIT does not change any configuration register.

Resetting CANFDx\_CHy\_CCCR.INIT finishes the software initialization. The CAN FD controller then synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (Bus Idle) before it can take part in bus activities and start the message transfer.

#### **Access/Set/Reset Properties of Registers Affected by Configuration Change Enable (CANFDx\_CHy\_CCCR.CCE)**

Access to the configuration registers is only enabled when both bits CANFDx\_CHy\_CCCR.INIT and CANFDx\_CHy\_CCCR.CCE are set (write-protected). CANFDx\_CHy\_CCCR.CCE can only be set/reset while CANFDx\_CHy\_CCCR.INIT is 1. CANFDx\_CHy\_CCCR.CCE is automatically reset when CANFDx\_CHy\_CCCR.INIT is reset.

The following registers are reset when CANFDx\_CHy\_CCCR.CCE is set

- CANFDx\_CHy\_HPMS - High Priority Message Status
- CANFDx\_CHy\_RXF0S - RX FIFO 0 Status
- CANFDx\_CHy\_RXF1S - RX FIFO 1 Status
- CANFDx\_CHy\_TXFQS - TX FIFO/Queue Status
- CANFDx\_CHy\_TXBRP - TX Buffer Request Pending
- CANFDx\_CHy\_TXBTO - TX Buffer Transmission Occurred
- CANFDx\_CHy\_TXBCF - TX Buffer Cancellation Finished
- CANFDx\_CHy\_TXEFS - TX Event FIFO Status
- CANFDx\_CHy\_TTOST - TT Operation Status
- CANFDx\_CHy\_TTLGT - TT Local and Global Time, only Global Time CANFDx\_CHy\_TTLGT.GT is reset
- CANFDx\_CHy\_TTCTC - TT Cycle Time and Count
- CANFDx\_CHy\_TTCSM - TT Cycle Sync Mark

In addition

- Timeout Counter value (CANFDx\_CHy\_TOCV.TOC[15:0]) is preset to the value configured by the Timeout Period (CANFDx\_CHy\_TOCC.TOP[15:0]) when CANFDx\_CHy\_CCCR.CCE is set.
- State machines of TX and RX handlers are held in idle state while CANFDx\_CHy\_CCCR.CCE is 1.

The following registers are only writable while CANFDx\_CHy\_CCCR.CCE is 0.

- CANFDx\_CHy\_TXBAR - TX Buffer Add Request
- CANFDx\_CHy\_TXBCR - TX Buffer Cancellation Request

Test Mode Enable (CANFDx\_CHy\_CCCR.TEST) and Bus Monitoring mode (CANFDx\_CHy\_CCCR.MON) can only be set by the CPU while CANFDx\_CHy\_CCCR.INIT is 1 and CANFDx\_CHy\_CCCR.CCE is 1. Both bits may be reset at any time.

Disable Automatic Retransmission (CANFDx\_CHy\_CCCR.DAR) can only be set/reset while CANFDx\_CHy\_CCCR.INIT is 1 and CANFDx\_CHy\_CCCR.CCE is 1.

## Message RAM Initialization

Each message RAM word should be reset by writing 0x00000000 before configuration of the CAN FD controller. This prevents message RAM bit errors when reading uninitialized words, and also avoids unexpected filter element configurations in message RAM.

### 17.3.1.2 Normal Operation

The M\_TTCAN's default operating mode after hardware reset is event-driven CAN communication without time triggers (CANFDx\_CHy\_TTOCF.OM = 00). Both CANFDx\_CHy\_CCCR.INIT and CANFDx\_CHy\_CCCR.CCE must be set before the TT operation mode is changed.

When M\_TTCAN is initialized and CANFDx\_CHy\_CCCR.INIT is reset to zero, M\_TTCAN synchronizes itself to the CAN bus and is ready for communication.

After passing the acceptance filtering, received messages including Message ID and DLC are stored into a dedicated RX buffer or into RX FIFO 0 or RX FIFO 1.

For messages to be transmitted, dedicated TX buffers and a TX FIFO/TX queue can be initialized or updated. Automated transmission on reception of remote frames is not implemented.

### 17.3.1.3 CAN FD Operation

The two variants in CAN FD frame transmission are:

- CAN FD frame without bit rate switching
- CAN FD frame where the control, data, and CRC fields are transmitted with a higher bit rate than the beginning and end of the frame

The previously reserved bit in CAN frames with 11-bit identifiers and 29-bit identifiers will now be decoded as FDF bit.

- FDF = recessive signifies a CAN FD frame
- FDF = dominant signifies a classic CAN frame

In a CAN FD frame, the two bits following FDF, reserved bits, and bit rate switch (BRS) decide whether the bit rate inside the CAN FD frame is switched. A CAN FD bit rate switch signified by res is dominant and BRS is recessive. The coding of res as recessive is reserved for future expansion of the protocol. If the M\_TTCAN receives a frame with FDF and res as recessive, it will signal a Protocol Exception Event by setting the CANFDx\_CHy\_PSR.PXE bit. When Protocol Exception Handling is enabled (CANFDx\_CHy\_CCCR.PXHD = 0), it causes the operation state to change from Receiver (CANFDx\_CHy\_PSR.ACT = 10) to Integrating (CANFDx\_CHy\_PSR.ACT = 00) at the next sample point. If Protocol Exception Handling is disabled (CANFDx\_CHy\_CCCR.PXHD = 1), the M\_TTCAN will treat a recessive res bit as a form error and respond with an error frame.

CAN FD operation is enabled by programming CANFDx\_CHy\_CCCR.FDOE. If CANFDx\_CHy\_CCCR.FDOE is '1', transmission and reception of CAN FD frames is enabled. Transmission and reception of classic CAN frames is always possible. Whether a CAN FD frame or a classic CAN frame is transmitted can be configured via the FDF bit in the respective TX buffer element. With CANFDx\_CHy\_CCCR.FDOE as '0', received frames are interpreted as classic CAN frames, which leads to the transmission of an error frame when receiving a CAN FD frame. When CAN FD operation is disabled, no CAN FD frames are transmitted even if the FDF bit of a TX buffer element is set. CANFDx\_CHy\_CCCR.FDOE and CANFDx\_CHy\_CCCR.BRSE can only be changed while CANFDx\_CHy\_CCCR.INIT and CANFDx\_CHy\_CCCR.CCE are both set.

With CANFDx\_CHy\_CCCR.FDOE as '0', the setting of FDF and BRS is ignored and frames are transmitted in classic CAN format. When CANFDx\_CHy\_CCCR.FDOE = 1 and CANFDx\_CHy\_CCCR.BRSE = 0, only FDF of a TX buffer element is evaluated. When CANFDx\_CHy\_CCCR.FDOE = 1 and CANFDx\_CHy\_CCCR.BRSE = 1, transmission of CAN FD frames with bit rate switching is enabled. All TX buffer elements with FDF and BRS bits set are transmitted in CAN FD format with bit rate switching.

A mode change during CAN operation is only recommended under the following conditions:

- The failure rate in the CAN FD data phase is significantly higher than in the CAN FD arbitration phase. In this case disable the CAN FD bit rate switching option for transmissions.
- During system startup, all nodes transmit classic CAN messages until it is verified that they can communicate in CAN FD format. If this is true, all nodes switch to CAN FD operation.
- Wake-up messages in CAN partial networking must be transmitted in classic CAN format.
- End-of-line programming occurs in case all nodes are not CAN FD capable. Non-CAN FD nodes are held in Silent mode until programming is completed. Then all nodes switch back to classic CAN communication.

In the CAN FD format, the coding of the DLC differs from the standard CAN format. The DLC codes 0 to 8 have the same coding as in standard CAN, codes 9 to 15, which in standard CAN have a data field of 8 bytes, are coded according to Table 17-1.

Table 17-1. Coding of DLC in CAN FD

| DLC                  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|----------------------|----|----|----|----|----|----|----|
| Number of Data Bytes | 12 | 16 | 20 | 24 | 32 | 48 | 64 |

In CAN FD frames, the bit timing will be switched inside the frame after the (BRS) bit, if this bit is recessive. Before the BRS bit, in the CAN FD arbitration phase, the nominal CAN bit timing is used as defined by the Nominal Bit Timing and Prescaler Register (CANFDx\_CHy\_NBTP). In the following CAN FD data phase, the data phase bit timing is used as defined by the Data Bit Timing and Prescaler Register (CANFDx\_CHy\_DBTP). The bit timing is switched back from the data phase timing at the CRC delimiter or when an error is detected, whichever occurs first.

The maximum configurable bit rate in the CAN FD data phase depends on the CAN FD peripheral clock frequency (canfd.clock\_can[0]). For example, with a CAN clock frequency of 20 MHz and the shortest configurable bit time of 4 tq, the bit rate in the data phase is 5 Mbit/s.

In both data frame formats, CAN FD and CAN FD with bit rate switching, the value of the bit ESI (Error Status Indicator) is determined by the transmitter's error state at the start of the transmission. If the transmitter is error passive, ESI is transmitted recessive; otherwise, it is transmitted dominant.

### 17.3.1.4 Transmitter Delay Compensation

During the data phase of a CAN FD transmission only one node is a transmitter; all others are receivers. The length of the bus line has no impact. When transmitting via pin canfd.tcan\_tx, the M\_TTCAN receives the transmitted data from its local CAN transceiver via pin canfd.tcan\_rx. The received data is delayed by the transmitter delay. In case this delay is greater than TSEG1 (time segment before sample point), a bit error is detected. To enable a data phase bit time that is even shorter than the transmitter delay, the delay compensation is introduced. Without transmitter delay compensation, the bit rate in the data phase of a CAN FD frame is limited by the transmitter delay.

#### Description

The M\_TTCAN's protocol unit has implemented a delay compensation mechanism to compensate the transmitter delay. This enables transmission with higher bit rates during the CAN FD data phase, independent of the delay of a specific CAN transceiver.

To check for bit errors during the data phase of transmitting nodes, the delayed transmit data is compared against the received data at the Secondary Sample Point (SSP). If a bit error is detected, the transmitter will react on this bit error at the next following regular sample point. During the arbitration phase the delay compensation is always disabled.

The transmitter delay compensation enables configurations where the data bit time is shorter than the transmitter delay, it is described in detail in the new ISO 11898-1:2015. It is enabled by setting bit CANFDx\_CHy\_DBTP.TDC.

The received bit is compared against the transmitted bit at the SSP. The SSP position is defined as the sum of the measured delay from the M\_TTCAN's transmit output canfd.tcan\_tx through the transceiver to the receive input RX plus the transmitter delay compensation offset as configured by CANFDx\_CHy\_TDCR.TDCO. The transmitter delay compensation offset is used to adjust the position of the SSP inside the received bit (for example, half of the bit time in the data phase). The position of the secondary sample point is rounded down to the next integer number of mtq (CAN FD peripheral clock).

CANFDx\_CHy\_PSR.TDCV shows the actual transmitter delay compensation value. CANFDx\_CHy\_PSR.TDCV is cleared when CANFDx\_CHy\_CCCR.INIT is set and is updated at each transmission of an FD frame while CANFDx\_CHy\_DBTP.TDC is set.

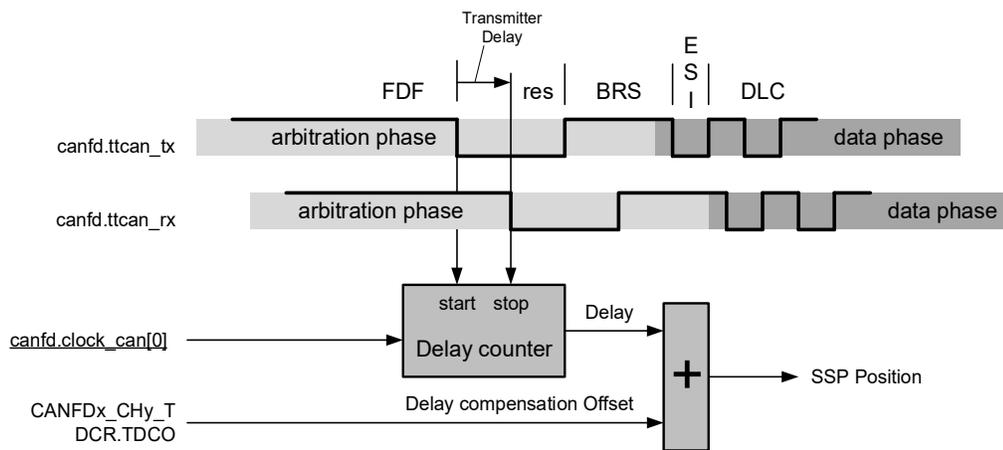
The following boundary conditions must be considered for the transmitter delay compensation implemented in the M\_TTCAN:

- The sum of the measured delay from canfd.ttcn\_tx to canfd.ttcn\_rx and the configured transmitter delay compensation offset CANFDx\_CHy\_TDCR.TDCO must be less than 6 bit times in the data phase.
- The sum of the measured delay from canfd.ttcn\_tx to canfd.ttcn\_rx and the configured transmitter delay compensation offset CANFDx\_CHy\_TDCR.TDCO should be less than or equal 127 mtq. In case this sum exceeds 127 mtq, the maximum value of 127 mtq is used for transmitter delay compensation
- The data phase ends at the sample point of the CRC delimiter that stops checking of receive bits at the SSPs.

**Transmitter Delay Compensation Measurement**

If transmitter delay compensation is enabled by programming CANFDx\_CHy\_DBTP.TDC = 1, the measurement is started within each transmitted CAN FD frame at the falling edge of bit FDF to bit res. The measurement is stopped when this edge is seen at the receive input canfd.ttcn\_rx of the transmitter. The resolution of this measurement is one mtq (minimum time quanta).

Figure 17-3. Transmitter Delay Measurement



To avoid this, a dominant glitch inside the received FDF bit ends the delay compensation measurement before the falling edge of the received res bit, resulting in an early SSP position. The use of a transmitter delay compensation filter window can be enabled by programming CANFDx\_CHy\_TDCR.TDCF. This defines a minimum value for the SSP position. Dominant edges on canfd.ttcn\_rx, that results in an earlier SSP position are ignored for transmitter delay measurement. The measurement is stopped when the SSP position is at least CANFDx\_CHy\_TDCR.TDCF and canfd.ttcn\_rx is low.

**17.3.1.5 Restricted Operation mode**

In Restricted Operation mode, the node is able to receive data and remote frames and acknowledge valid frames, but it does not send data frames, remote frames, active error frames, or overload frames. In case of an error or overload condition, it does not send dominant bits; instead it waits for the occurrence of a bus idle condition to resynchronize itself to the CAN communication. The error counters (CANFDx\_CHy\_ECR.REC and CANFDx\_CHy\_ECR.TEC) are frozen while Error Logging (CANFDx\_CHy\_ECR.CEL) is active.

The CPU can set the CAN FD controller into Restricted Operation mode by setting the Restricted Operation mode bit (CANFDx\_CHy\_CCCR.ASM). CANFDx\_CHy\_CCCR.ASM can only be set by the CPU when both CANFDx\_CHy\_CCCR.CCE and CANFDx\_CHy\_CCCR.INIT are set to '1'. CANFDx\_CHy\_CCCR.ASM can be reset by the CPU at any time.

The CAN FD controller enters Restricted Operation mode automatically when the TX handler is not able to read data from the message RAM in time. To leave this mode, the CPU should reset CANFDx\_CHy\_CCCR.ASM.

The Restricted Operation mode can be used in applications that adapt themselves to different CAN bit rates. In this case, the application tests different bit rates and leaves the mode after it has received a valid frame.

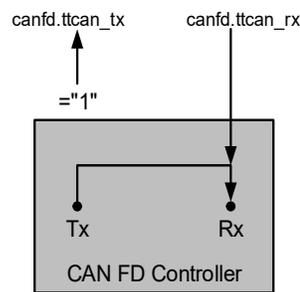
**Note:** The Restricted Operation mode must not be combined with the Loop Back mode (internal or external).

### 17.3.1.6 Bus Monitoring Mode

The M\_TTCAN is set in Bus Monitoring mode by programming CANFDx\_CHy\_CCCR.MON to '1' or when error level S3 (CANFDx\_CHy\_TTOST.EL = 11) is entered. In Bus Monitoring mode, the M\_TTCAN is able to receive valid data frames and valid remote frames, but cannot start a transmission. In this mode, it sends only recessive bits on the CAN bus, if the M\_TTCAN is required to send a dominant bit (ACK bit, overload flag, or active error flag), the bit is rerouted internally so that the M\_TTCAN monitors this dominant bit, although the CAN bus may remain in recessive state. In Bus Monitoring mode, the CANFDx\_CHy\_TXBRP register is held in reset state.

The Bus Monitoring mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits. Figure 17-4 shows the connection of signals canfd.tcan\_tx and canfd.tcan\_rx to the M\_TTCAN in Bus Monitoring mode.

Figure 17-4. Pin Control in Bus Monitoring Mode



Bus Monitoring Mode

### 17.3.1.7 Disable Automatic Retransmission

M\_TTCAN supports automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. By default, automatic retransmission is enabled. To support time-triggered communication (as described in ISO 11898-1:2015, chapter 9.2), the automatic retransmission may be disabled via CANFDx\_CHy\_CCCR.DAR.

In DAR mode, all transmissions are automatically canceled after they are started on the CAN bus. The TX Request Pending bit CANFDx\_CHy\_TXBRP.TRPx is reset after successful transmission, when a transmission has not yet started at the point of cancellation, is aborted due to lost arbitration, or when an error occurred during frame transmission.

- Successful transmission:
  - Corresponding TX Buffer Transmission Occurred bit CANFDx\_CHy\_TXBTO.TOx set
  - Corresponding TX Buffer Cancellation Finished bit CANFDx\_CHy\_TXBCF.CFx not set
- Successful transmission in spite of cancellation:
  - Corresponding TX Buffer Transmission Occurred bit CANFDx\_CHy\_TXBTO.TOx set
  - Corresponding TX Buffer Cancellation Finished bit CANFDx\_CHy\_TXBCF.CFx set
- Arbitration lost or frame transmission disturbed:
  - Corresponding TX Buffer Transmission Occurred bit CANFDx\_CHy\_TXBTO.TOx not set
  - Corresponding TX Buffer Cancellation Finished bit CANFDx\_CHy\_TXBCF.CFx set

In successful frame transmissions, and if storage of TX events is enabled, a TX Event FIFO element is written with Event Type ET = 10 (transmission despite cancellation).

### 17.3.1.8 Power Down (Sleep Mode)

The M\_TTCAN channel can be set into power down mode via Clock Stop Request (CANFDx\_CTL.STOP\_REQ). As long as clock stop request is active, STOP\_REQ bit is read as one.

When all pending transmission requests have completed, the M\_TTCAN waits until bus idle state is detected. Then the M\_TTCAN sets CANFDx\_CHy\_CCCR.INIT to one to prevent any further CAN transfers. Now the M\_TTCAN acknowledges that it is ready for power down by setting Clock Stop Acknowledge (CANFDx\_STATUS.STOP\_ACK). Upon receiving acknowledgment from channel, hardware automatically switches off the clock to the respective channel.

To leave power down mode, the application must reset CANFDx\_CTL.STOP\_REQ. The M\_TTCAN will acknowledge this by resetting CANFDx\_STATUS.STOP\_ACK. Afterwards, the application can restart CAN communication by resetting bit CANFDx\_CHy\_CCCR.INIT.

When the clock stop request is triggered through CANFDx\_CTL.STOP\_REQ, it must not be cleared before CANFDx\_STATUS.STOP\_ACK bit is set.

**Note:** Do not use the TTCAN CANFDx\_CHy\_CCCR.CSR register for the power down control, instead use CANFDx\_CTL.STOP\_REQ. Similarly, use of CANFDx\_CHy\_CCCR.CSA should be avoided, instead use CANFDx\_STAUTS.STOP\_ACK.

### 17.3.1.9 Test Mode

To enable write access to CANFDx\_CHy\_TEST register, Test Mode Enable bit (CANFDx\_CHy\_CCCR.TEST) must be set to one. This allows the configuration of the test modes and test functions.

Four output functions are available for the CAN transmit pin canfd.ttcn\_tx by programming CANFDx\_CHy\_TEST.TX. Apart from its default function of serial data output, it can drive the CAN Sample Point signal to monitor the M\_TTCAN's bit timing; it can also drive constant dominant or recessive values. The actual value at the canfd.ttcn\_rx pin can be read from CANFDx\_CHy\_TEST.RX. Both functions can be used to check the CAN bus physical layer.

Due to the synchronization mechanism between CAN clock and host clock domain, there may be a delay of several host clock periods between writing to CANFDx\_CHy\_TEST.TX until the new configuration is visible at output pin canfd.ttcn\_tx. This applies also when reading input pin canfd.ttcn\_rx via CANFDx\_CHy\_TEST.RX.

**Note:** Test modes should be used for production tests or self-test only. The software control for pin canfd.ttcn\_tx interferes with all CAN protocol functions. It is not recommended to use test modes for application.

#### External Loop Back Mode

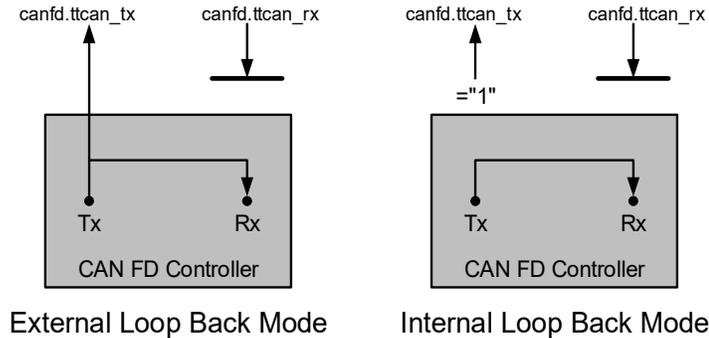
The M\_TTCAN can be set in External Loop Back mode by programming CANFDx\_CHy\_TEST.LBCK to one. In Loop Back mode, the M\_TTCAN treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into an RX buffer or an RX FIFO. [Figure 17-5](#) shows the connection of signals canfd.ttcn\_tx and canfd.ttcn\_rx to the M\_TTCAN in External Loop Back mode.

This mode is provided for hardware self-test. To be independent from external stimulation, the M\_TTCAN ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back mode. In this mode the M\_TTCAN performs an internal feedback from its TX output to its RX input. The actual value of the canfd.ttcn\_rx input pin is disregarded by the M\_TTCAN. The transmitted messages can be monitored at the canfd.ttcn\_tx pin.

#### Internal Loop Back Mode

Internal Loop Back mode is entered by programming the CANFDx\_CHy\_TEST.LBCK and CANFDx\_CHy\_CCCR.MON bits to one. This mode can be used for a "Hot Selftest", meaning the M\_TTCAN can be tested without affecting a running CAN system connected to the canfd.ttcn\_tx and canfd.ttcn\_rx pins. In this mode, canfd.ttcn\_rx pin is disconnected from the M\_TTCAN and canfd.ttcn\_tx pin is held recessive. [Figure 17-5](#) shows the connection of canfd.ttcn\_tx and canfd.ttcn\_rx to the M\_TTCAN in Internal Loop Back mode.

Figure 17-5. Pin Control in Loop Back Modes



### 17.3.1.10 Application Watchdog

The application watchdog is served by reading the `CANFDx_CHy_TTOST` register. When the application watchdog is not served in time, `CANFDx_CHy_TTOST.AWE` bit is set, all TTCAN communication is stopped, and the `M_TTCAN` is set into Bus Monitoring mode.

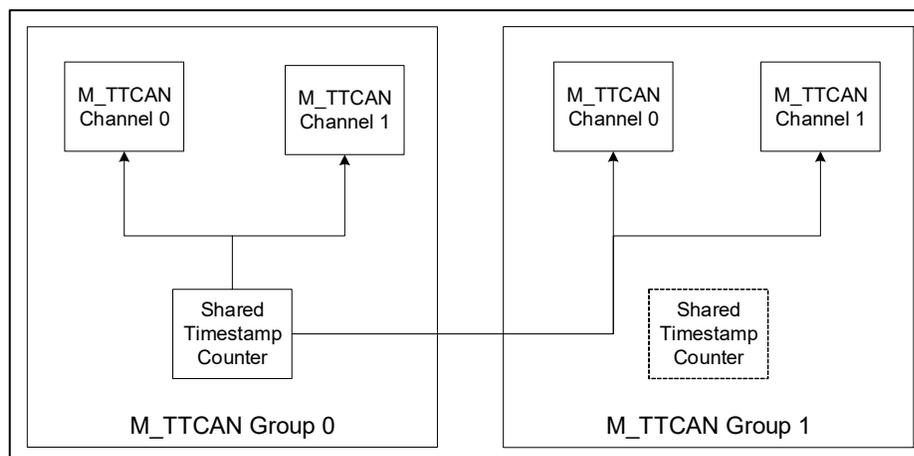
The TT application watchdog can be disabled by programming the Application Watchdog Limit `CANFDx_CHy_TTOCF.AWL` to `0x00`. The TT application watchdog should not be disabled in a TTCAN application program.

### 17.3.2 Timestamp Generation

The `M_TTCAN` channel uses a 16-bit counter to record when messages are sent or received. This allows the application software to know the order in which events occurred.

To keep event ordering across multiple `M_TTCAN` channels, a global timestamp counter is implemented, which must be selected by setting '10' to `CANFDx_CHy_TSCC.TSS[1:0]`. This global timestamp counter is shared among all `M_TTCAN` groups present in the device. For instance, if the device contains two `M_TTCAN` groups, timestamp counter is shared among all the channels present in both the groups.

Figure 17-6. Timestamp Connection Between Two `M_TTCAN` Group (depiction)



The timestamp counter is configured through the `CANFDx_TS_CTL` register. The `CANFDx_TS_CTL.ENABLED` bit will enable the counter. Upon enabling, it will start incrementing according to the `CANFDx_TS_CTL.PRESCALE [15:0]`. The application can read the counter value through the `CANFDx_TS_CNT` register. Write access to the `CANFDx_TS_CNT` register will clear the `CANFDx_TS_CNT`.

When the timestamp counter is enabled, internal counter for prescaler counts with every cycle of CLK\_SYS; when the counter value reaches the prescaler value, the timestamp counter increments by one and internal prescaler counter is cleared. When CANFDx\_TS\_CTL.PRESCALE changes, CANFDx\_TS\_CNT should be written to reset them. This can make the internal prescaler counter follow a new value of CANFDx\_TS\_CTL.PRESCALE immediately.

The shared timestamp counter is a wrap-around counter. When the counter wraps around, CANFDx\_CHy\_IR.TSW for all M\_TTCAN channels will be raised.

On start of frame reception/transmission, the timestamp counter value is captured and stored into the timestamp section of an RX buffer/RX FIFO (RXTS [15:0]) or TX Event FIFO (TXTS [15:0]) element.

**Note:** The counter value CANFDx\_TS\_CNT is not retained in DeepSleep mode whereas the CANFDx\_TS\_CTL is retained.

### 17.3.3 Timeout Counter

To signal timeout conditions for RX FIFO 0, RX FIFO 1, and the TX Event FIFO, the M\_TTCAN supplies a 16-bit Timeout Counter. It operates as down-counter and uses the same prescaler controlled by CANFDx\_CHy\_TSCC.TCP as the timestamp Counter. A prescaler CANFDx\_CHy\_TSCC.TCP should be configured to clock the timeout counter in multiples of CAN bit times (1...16). The timeout counter is configured via register CANFDx\_CHy\_TOCC. The actual counter value can be read from CANFDx\_CHy\_TOCV.TOC.

The timeout counter can only be started while CANFDx\_CHy\_CCCR.INIT = 0. It is stopped when CANFDx\_CHy\_CCCR.INIT = 1; for example, when the M\_TTCAN enters Bus\_Off state.

The operation mode is selected by CANFDx\_CHy\_TOCC.TOS. When operating in Continuous mode, the counter starts when CANFDx\_CHy\_CCCR.INIT is reset. A write to CANFDx\_CHy\_TOCV presets the counter to the value configured by CANFDx\_CHy\_TOCC.TOP and continues down-counting.

When the timeout counter is controlled by one of the FIFOs, an empty FIFO presets the counter to the value configured by CANFDx\_CHy\_TOCC.TOP. Down-counting is started when the first FIFO element is stored. Writing to CANFDx\_CHy\_TOCV has no effect.

When the counter reaches zero, interrupt flag CANFDx\_CHy\_IR.TOO is set. In Continuous mode, the counter is immediately restarted at CANFDx\_CHy\_TOCC.TOP.

**Note:** The clock signal for the timeout counter is derived from the CAN Core's sample point signal. Therefore, the time the Timeout Counter is decremented may vary due to the synchronization/resynchronization mechanism of the CAN Core. If the bit rate switch feature in CAN FD is used, the timeout counter is clocked differently in arbitration and data field.

### 17.3.4 RX Handling

The RX handler controls acceptance filtering, transfer of received messages to the RX buffers or to one of the two RX FIFOs, as well as RX FIFO's Put and Get Indices.

#### 17.3.4.1 Acceptance Filtering

The M\_TTCAN offers the possibility to configure two sets of acceptance filters, one for standard identifiers and one for extended identifiers. These filters can be assigned to an RX buffer or to RX FIFO 0,1. For acceptance filtering each list of filters is executed from element #0 until the first matching element. Acceptance filtering stops at the first matching element. The following filter elements are not evaluated for this message.

The main features are:

- Each filter element can be configured as
  - Range filter (from - to)
  - Filter for one or two dedicated IDs
  - Classic bit mask filter
- Each filter element is configurable for acceptance or rejection filtering
- Each filter element can be enabled/disabled individually
- Filters are checked sequentially; execution stops with the first matching filter element

Related configuration registers are:

- Global Filter Configuration (CANFDx\_CHy\_GFC)
- Standard ID Filter Configuration (CANFDx\_CHy\_SIDFC)
- Extended ID Filter Configuration (CANFDx\_CHy\_XIDFC)
- Extended ID AND Mask (CANFDx\_CHy\_XIDAM)

Depending on the configuration of the filter element (SFEC/EFEC) a match triggers one of the following actions:

- Store received frame in FIFO 0 or FIFO 1
- Store received frame in RX buffer
- Store received frame in RX buffer and generate pulse at filter event pin
- Reject received frame
- Set High-Priority Message interrupt flag CANFDx\_CHy\_IR.HPM
- Set High-Priority Message interrupt flag CANFDx\_CHy\_IR.HPM and store received frame in FIFO 0 or FIFO 1

Acceptance filtering is started after the complete identifier is received. After acceptance filtering has completed, if a matching RX buffer or RX FIFO is found, the message handler starts writing the received message data in portions of 32 bits to the matching RX buffer or RX FIFO. If the CAN protocol controller has detected an error condition (such as CRC error), this message is discarded with the following impact on the affected RX buffer or RX FIFO:

- RX Buffer

New data flag of the matching RX buffer is not set, but RX buffer is (partly) overwritten with received data. For error type, see CANFDx\_CHy\_PSR.LEC and CANFDx\_CHy\_PSR.DLEC, respectively.

- RX FIFO

Put index of matching RX FIFO is not updated, but related RX FIFO element is (partly) overwritten with received data. For error type, see CANFDx\_CHy\_PSR.LEC and CANFDx\_CHy\_PSR.DLEC, respectively. If the matching RX FIFO is operated in overwrite mode, the boundary conditions described in [RX FIFO Overwrite Mode on page 194](#) should be considered.

**Note:** When an accepted message is written to one of the two RX FIFOs, or into an RX buffer, the unmodified received identifier is stored independent of the filter(s) used. The result of the acceptance filter process depends on the sequence of configured filter elements.

### Range Filter

The filter matches for all received frames with Message IDs in the range defined by SF1ID/SF2ID resp. EF1ID/EF2ID.

The two possibilities when range filtering is used with extended frames are:

- EFT = 00: The Message ID of received frames is ANDed with the CANFDx\_CHy\_XIDAM before the range filter is applied
- EFT = 11: The CANFDx\_CHy\_XIDAM is not used for range filtering

### Filter for Specific IDs

A filter element can be configured to filter one or two specific Message IDs. To filter a specific Message ID, the filter element should be configured with SF1ID = SF2ID and EF1ID = EF2ID, respectively.

### Classic Bit Mask Filter

Classic bit mask filtering is intended to filter groups of Message IDs by masking single bits of a received Message ID. With classic bit mask filtering, SF1ID/EF1ID is used as Message ID filter, while SF2ID/EF2ID is used as filter mask.

A zero bit at the filter mask will mask the corresponding bit position of the configured ID filter; for example, the value of the received Message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received Message ID where the corresponding mask bits are one are relevant for acceptance filtering.

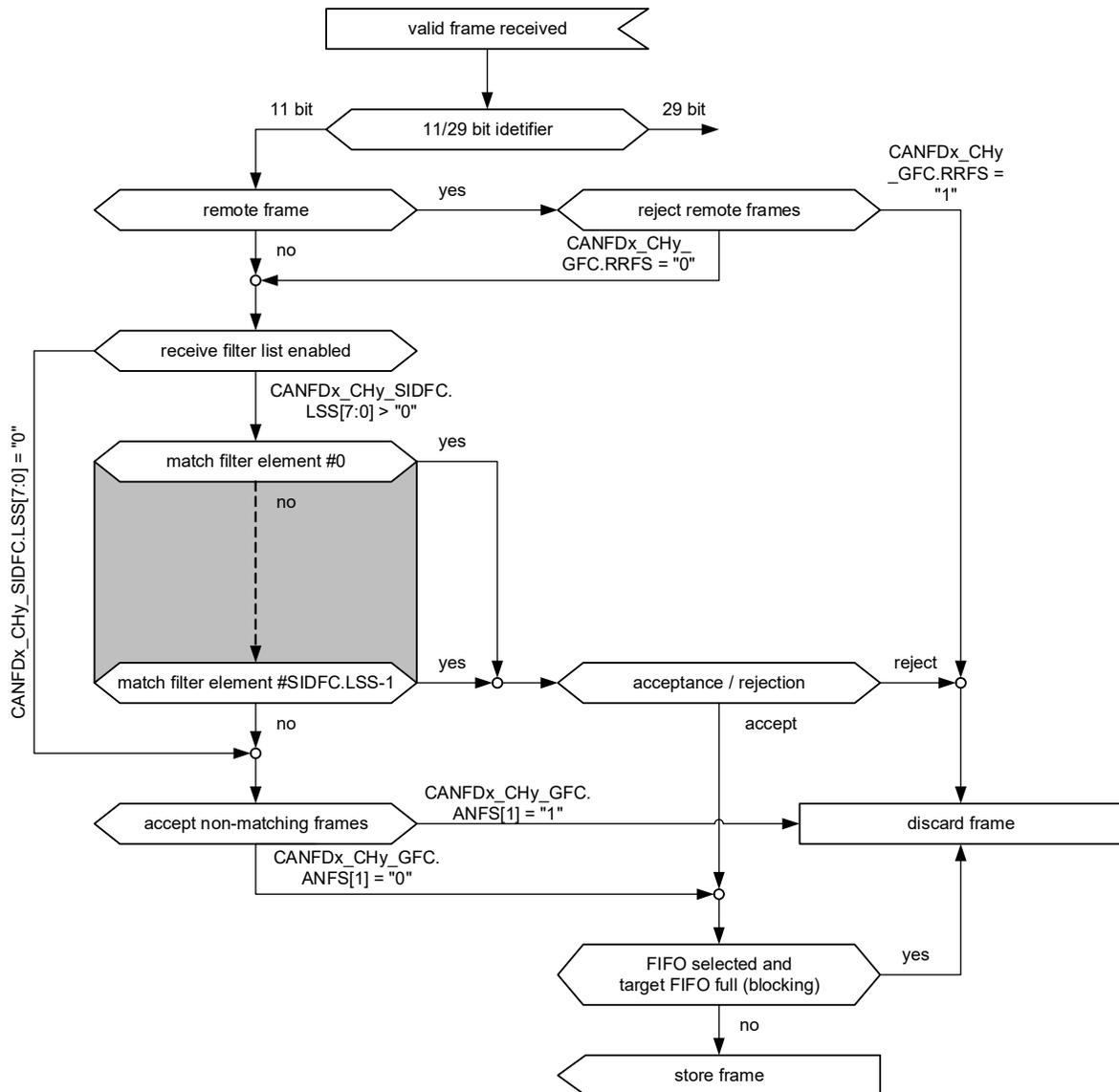
In case all mask bits are one, a match occurs only when the received Message ID and the Message ID filter are identical. If all mask bits are zero, all Message IDs match.

### Standard Message ID Filtering

[Figure 17-7](#) shows the flow for standard Message ID (11-bit Identifier) filtering. The Standard Message ID Filter element is described in [Standard Message ID Filter Element on page 212](#).

Controlled by the CANFDx\_CHy\_GFC and CANFDx\_CHy\_SIDFC Message IDs, the Remote Transmission Request bit (RTR) and the Identifier Extension bit (IDE) of received frames are compared against the list of configured filter elements.

Figure 17-7. Standard Message ID Filter



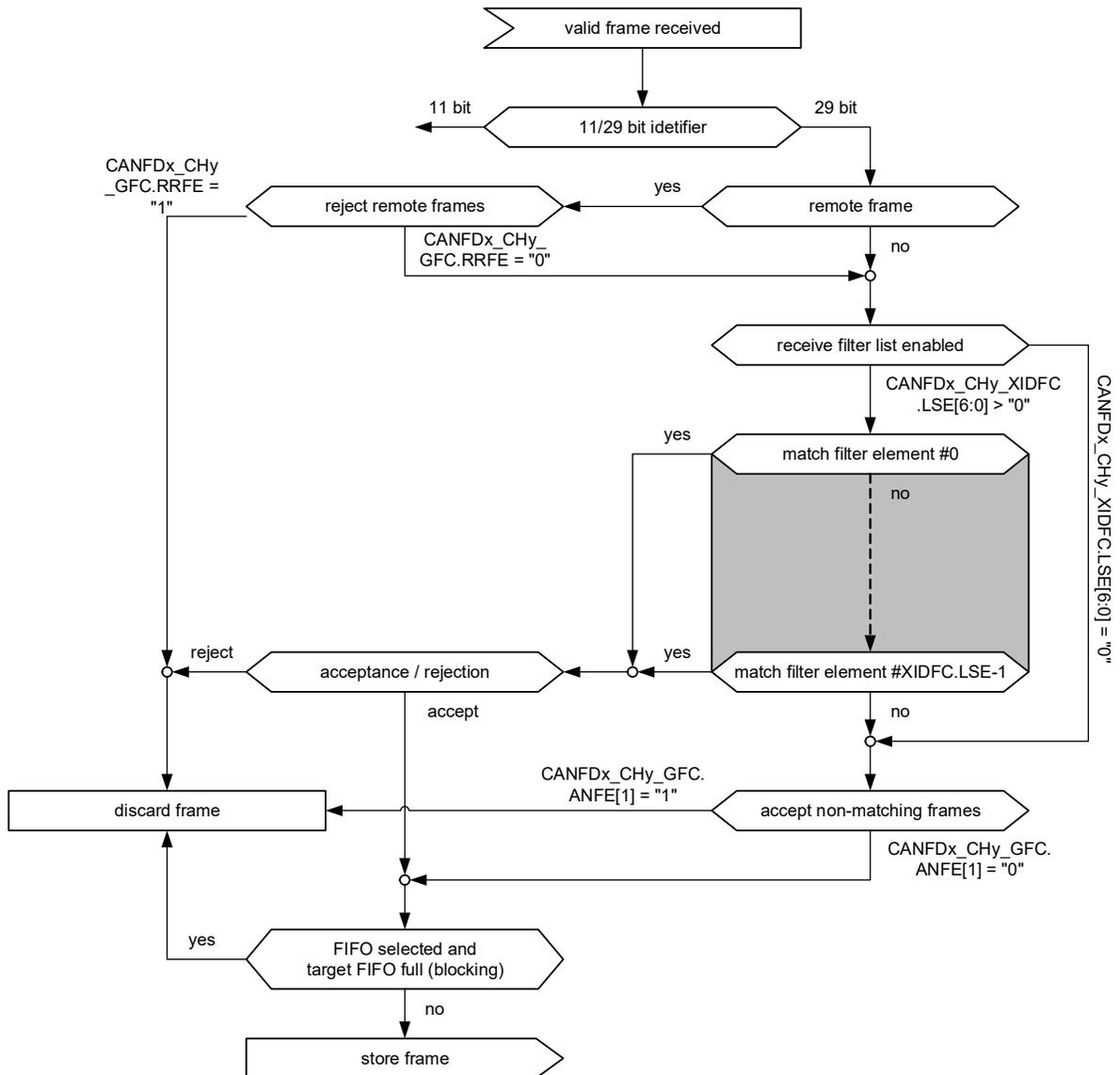
### Extended Message ID Filtering

Figure 17-8 shows the flow for extended Message ID (29-bit Identifier) filtering. The Extended Message ID Filter element is described in [Extended Message ID Filter Element on page 213](#).

Controlled by the CANFDx\_CHy\_GFC and CANFDx\_CHy\_XIDFC Message IDs, the RTR bit, and IDE bit of received frames are compared against the list of configured filter elements.

The Extended ID AND Mask XIDAM[28:0] is ANDed with the received identifier before the filter list is executed.

Figure 17-8. Extended Message ID Filter Path



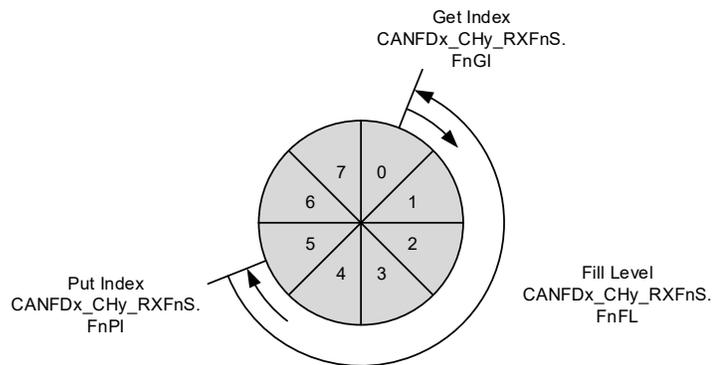
### 17.3.4.2 RX FIFOs

RX FIFO 0 and RX FIFO 1 can be configured to hold up to 64 elements each. The two RX FIFOs are configured via the CANFDx\_CHy\_RXF0C and CANFDx\_CHy\_RXF1C registers.

Received messages that pass acceptance filtering are transferred to the RX FIFO as configured by the matching filter element. For a description of the filter mechanisms available for RX FIFO 0 and RX FIFO 1, see [Acceptance Filtering on page 189](#). The RX FIFO element is described in [RX Buffer and FIFO Element on page 206](#).

To avoid an RX FIFO overflow, the RX FIFO watermark can be used. When the RX FIFO fill level reaches the RX FIFO watermark configured by CANFDx\_CHy\_RXFnC.FnWM, interrupt flag CANFDx\_CHy\_IR.RFnW is set. When the RX FIFO Put Index reaches the RX FIFO Get Index, an RX FIFO Full condition is signaled by CANFDx\_CHy\_RXFnS.FnF. In addition, interrupt flag CANFDx\_CHy\_IR.RFnF is set. The FIFO watermark interrupt flags can be used to trigger the DMA. DMA request for FIFO will remain set until the respective trigger is cleared by software. Software can clear the trigger by clearing the watermark flag.

Figure 17-9. RX FIFO Status



When reading from an RX FIFO, RX FIFO Get Index CANFDx\_CHy\_RXFnS.FnGI x FIFO Element Size has to be added to the corresponding RX FIFO start address CANFDx\_CHy\_RXFnC.FnSA. RX FIFO Top pointer logic is added to the CAN FD controller to make reading faster. See [RX FIFO Top Pointer on page 195](#).

Table 17-2. RX Buffer/FIFO Element size

| CANFDx_CHy_RXESC.RBDS[2:0]<br>CANFDx_CHy_RXESC.FnDS[2:0] | Data Field<br>[bytes] | FIFO Element Size<br>[RAM words] |
|--|-----------------------|----------------------------------|
| 000  | 8                     | 4                                |
| 001  | 12                    | 5                                |
| 010  | 16                    | 6                                |
| 011  | 20                    | 7                                |
| 100  | 24                    | 8                                |
| 101  | 32                    | 10                               |
| 110  | 48                    | 14                               |
| 111  | 64                    | 18                               |

### RX FIFO Blocking Mode

The RX FIFO blocking mode is configured by  $CANFDx\_CHy\_RXFnC.FnOM = 0$ . This is the default operation mode for RX FIFOs.

When an RX FIFO full condition is reached ( $CANFDx\_CHy\_RXFnS.FnPI = CANFDx\_CHy\_RXFnS.FnGI$ ), no further messages are written to the corresponding RX FIFO until at least one message is read and the RX FIFO Get Index is incremented. An RX FIFO full condition is signaled by  $CANFDx\_CHy\_RXFnS.FnF = 1$ . In addition, the interrupt flag  $CANFDx\_CHy\_IR.RFnF$  is set.

If a message is received while the corresponding RX FIFO is full, this message is discarded and the message lost condition is signaled by  $CANFDx\_CHy\_RXFnS.RFnL = 1$ . In addition, the interrupt flag  $CANFDx\_CHy\_IR.RFnL$  is set.

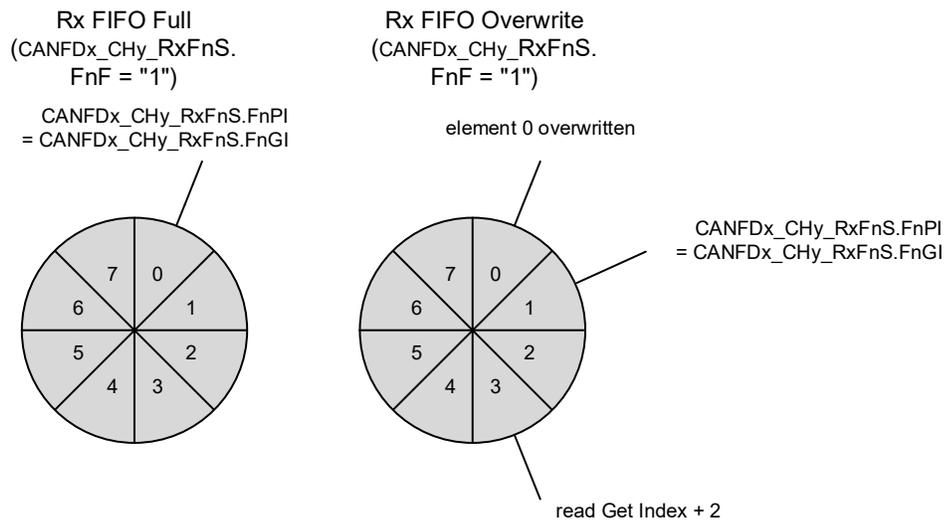
### RX FIFO Overwrite Mode

The RX FIFO overwrite mode is configured by  $CANFDx\_CHy\_RXFnC.FnOM = 1$ .

When an RX FIFO full condition ( $CANFDx\_CHy\_RXFnS.FnPI = CANFDx\_CHy\_RXFnS.FnGI$ ) is signaled by  $CANFDx\_CHy\_RXFnS.FnF = 1$ , the next message accepted for the FIFO will overwrite the oldest FIFO message. Put and Get indices are both incremented by one.

When an RX FIFO is operated in overwrite mode and an RX FIFO full condition is signaled, reading of the RX FIFO elements should start at least at Get Index + 1. This is because a received message may be written to the message RAM (Put Index) while the CPU is reading from the message RAM (Get Index). In this case, inconsistent data may be read from the respective RX FIFO element. Adding an offset to the Get Index when reading from the RX FIFO avoids this problem. The offset depends on how fast the CPU accesses the RX FIFO. Figure 17-10 shows an offset of two with respect to the Get Index when reading the RX FIFO. In this case, the two messages stored in element 1 and 2 are lost.

Figure 17-10. RX FIFO Overflow Handling



After reading from the RX FIFO, the number of the last element read must be written to the RX FIFO Acknowledge Index  $CANFDx\_CHy\_RXFnA.FnA$ . This increments the Get Index to that element number. If the Put Index is not incremented to this RX FIFO element, the RX FIFO full condition is reset ( $CANFDx\_CHy\_RXFnS.FnF = 0$ ).

### RX FIFO Top Pointer

M\_TTCAN supports two receive FIFOs. Reading from these FIFOs requires application to go through following steps:

- Retrieve read pointer
- Calculate correct message RAM address
- Read the data from message RAM
- Update the read pointer

To avoid all these steps, RX FIFO Top Pointer logic has been integrated in the CAN FD controller. It provides a single MMIO location (CANFDx\_CHy\_RXFTOPn\_DATA; n = 0,1) to read the data from. Using such hardware logic has the following benefits:

- Higher performance data access
- Less bus traffic
- Reduced CPU load
- Reduced power
- Enables DMA access to FIFO

This logic is enabled when CANFDx\_CHy\_RXFTOP\_CTL.FnTPE is set. Setting this bit enables the logic to set the FIFO top address (FnTA) and internal message word counter. Receive FIFO in the top status register (CANFDx\_CHy\_RXFTOPn\_STAT) shows the respective FIFO top address and CANFDx\_CHy\_RXFTOPn\_DATA provides the data located at the top address. Refer to register definitions for more details on both registers.

If CANFDx\_CHy\_RXFTOPn\_DATA is read, the top pointer logic also updates the RX FIFO Acknowledge Index (CANFDx\_CHy\_RXFnA.FnA) in TTCAN channel.

**Note:** Top pointer logic is disabled when the channel is being configured (CANFDx\_CHy\_CCCR.CCE = 1). Reading CANFDx\_CHy\_RXFTOPn\_DATA while the logic is disabled will return the invalid data.

#### 17.3.4.3 Dedicated RX Buffers

The M\_TTCAN supports up to 64 dedicated RX buffers. The start address of the dedicated RX buffer section is configured via CANFDx\_CHy\_RXBC.RBSA.

For each RX buffer, a Standard or Extended Message ID Filter Element with SFEC/EFEC = 111 and SFID2/EFID2[10:9] = 00 must be configured (s (see [17.4.5 Standard Message ID Filter Element](#) and [17.4.6 Extended Message ID Filter Element](#)).

After a received message is accepted by a filter element, the message is stored into the RX buffer in the message RAM referenced by the filter element. The format is the same as for an RX FIFO element. In addition, the flag CANFDx\_CHy\_IR.DRX (message stored in a dedicated RX buffer) in the interrupt register is set.

Table 17-3. Example Filter Configuration for RX Buffers

| Filter Element | SFID1[10:0]<br>EFID1[28:0] | SFID2[10:9]<br>EFID2[10:9] | SFID2[5:0]<br>EFID2[5:0] |
|----------------|----------------------------|----------------------------|--------------------------|
| 0              | ID message 1               | 00                         | 00 0000                  |
| 1              | ID message 2               | 00                         | 00 0001                  |
| 2              | ID message 3               | 00                         | 00 0010                  |

After the last word of a matching received message is written to the message RAM, the respective New Data flag in CANFDx\_CHy\_NDAT1 and CANFDx\_CHy\_NDAT2 registers is set. As long as the New Data flag is set, the respective RX buffer is locked against updates from received matching frames. The New Data flags should be reset by the host by writing a '1' to the respective bit position.

While an RX buffer's New Data flag is set, a Message ID Filter Element referencing the specific RX buffer will not match, causing the acceptance filtering to continue. The following Message ID Filter Elements may cause the received message to be stored into another RX buffer, or into an RX FIFO, or the message may be rejected, depending on filter configuration.

- Reset interrupt flag CANFDx\_CHy\_IR.DRX
- Read New Data registers
- Read messages from message RAM
- Reset New Data flags of processed messages

#### 17.3.4.4 Debug on CAN Support

Debug messages are stored into RX buffers; three consecutive RX buffers (for example, #61, #62, and #63) should be used to store debug messages A, B, and C. The format is the same for RX buffer and RX FIFO elements.

To filter debug messages Standard/Extended Filter Elements with SFEC/EFEC = "111" should be set up. Messages that match these filter elements are stored into the RX buffers addressed by SFID2/EFID2[5:0].

After message C is stored, the DMA request is activated and the three messages can be read from the message RAM under DMA control. The RAM words holding the debug messages will not be changed by the M\_TTCAN while DMA request is activated. The behavior is similar to that of an RX buffer with its New Data flag set.

After the DMA transfer is completed, an acknowledge from DMA resets the DMA request. Now the M\_TTCAN is prepared to receive the next set of debug messages.

#### Filtering Debug Messages

Debug messages are filtered by configuring one Standard/Extended Message ID filter element for each of the three debug messages. To enable a filter element to filter debug messages, SFEC/EFEC should be programmed to "111". In this case the SFID1/SFID2 and EFID1/EFID2 fields have a different meaning (see [Standard Message ID Filter Element on page 212](#) and [Extended Message ID Filter Element on page 213](#)). While SFID2/EFID2[10:9] controls the debug message handling state machine, SFID2/EFID2[5:0] controls the storage location of a received debug message.

When a debug message is stored, neither the respective New Data flag nor CANFDx\_CHy\_IR.DRX are set. The reception of debug messages can be monitored via CANFDx\_CHy\_RXF1S.DMS.

Table 17-4. Example Filter Configuration for Debug Message

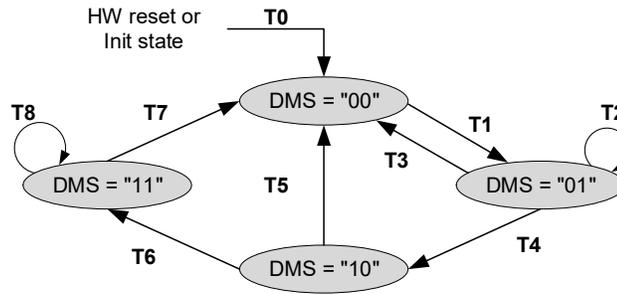
| Filter Element | SFID1[10:0]<br>EFID1[28:0] | SFID2[10:9]<br>EFID2[10:9] | SFID2[5:0]<br>EFID2[5:0] |
|----------------|----------------------------|----------------------------|--------------------------|
| 0              | ID debug message A         | 01                         | 11 1101                  |
| 1              | ID debug message B         | 10                         | 11 1110                  |
| 2              | ID debug message C         | 11                         | 11 1111                  |

#### Debug Message Handling

The debug message handling state machine assures that debug messages are stored to three consecutive RX buffers in the correct order. If there are missing messages, the process is restarted. The DMA request is activated only when all three debug messages A, B, and C are received in correct order.

The status of the debug message handling state machine is signaled via CANFDx\_CHy\_RXF1S.DMS.

Figure 17-11. Debug Message Handling State Machine



T0: reset DMA request, enable reception of debug messages A, B, and C  
 T1: reception of debug message A  
 T2: reception of debug message A  
 T3: reception of debug message C  
 T4: reception of debug message B  
 T5: reception of debug messages A, B  
 T6: reception of debug message C  
 T7: DMA transfer completed  
 T8: reception of debug message A, B, C (message rejected)

### 17.3.5 TX Handling

The TX handler handles transmission requests for the dedicated TX buffers, TX FIFO, and TX Queue. It controls the transfer of transmit messages to the CAN Core, the Put and Get Indices, and the TX Event FIFO. Up to 32 TX buffers can be set up for message transmission. The CAN mode for transmission (Classic CAN or CAN FD) can be configured separately for each TX buffer element. The TX buffer element is described in [TX Buffer Element on page 208](#). [Table 17-5](#) describes the possible configurations for frame transmission.

Table 17-5. Possible Configuration for Frame Transmission

| CANFDx_CHy_CCCR |      | TX Buffer Element |         | Frame Transmission            |
|-----------------|------|-------------------|---------|-------------------------------|
| BRSE            | FDOE | fdf               | BRS     |                               |
| ignored         | 0    | ignored           | ignored | Classic CAN                   |
| 0               | 1    | 0                 | ignored | Classic CAN                   |
| 0               | 1    | 1                 | ignored | FD without bit rate switching |
| 1               | 1    | 0                 | ignored | Classic CAN                   |
| 1               | 1    | 1                 | 0       | FD without bit rate switching |
| 1               | 1    | 1                 | 1       | FD with bit rate switching    |

The TX handler starts a TX scan to check for the highest priority pending TX request (TX buffer with lowest Message ID) when the TX Buffer Request Pending register (CANFDx\_CHy\_TXBRP) is updated, or when a transmission is started.

### 17.3.5.1 Transmit Pause

The transmit pause feature is intended for use in CAN systems where the CAN message identifiers are (permanently) assigned to specific values and cannot be changed easily. These message identifiers may have a higher CAN arbitration priority than other defined messages, while in a specific application their relative arbitration priority should be inverse. This may lead to a case where one Electronic Control Unit (ECU) sends a burst of CAN messages that cause another ECU's CAN messages to be delayed because the other messages have a lower CAN arbitration priority.

For example, if CAN ECU-1 has the transmit pause feature enabled and is requested by the application software to transmit four messages, it will, after the first successful message transmission, wait for two nominal bit times of bus idle before it is allowed to start the next requested message. If there are other ECUs with pending messages, those messages are started in the idle time, they will not need to arbitrate with the next message of ECU-1. After having received a message, ECU-1 is allowed to start its next transmission as soon as the received message releases the CAN bus.

The transmit pause feature is controlled by the Transmit Pause bit (CANFDx\_CHy\_CCCR.TXP). If the bit is set, the M\_TTCAN controller will, each time it has successfully transmitted a message, pause for two nominal bit times before starting the next transmission. This enables other CAN nodes in the network to transmit messages even if their messages have lower prior identifiers. Default is transmit pause disabled (CANFDx\_CHy\_CCCR.TXP = 0).

This feature loses burst transmissions coming from a single node and protects against “babbling idiot” scenarios where the application program erroneously requests too many transmissions.

### 17.3.5.2 Dedicated TX Buffers

Dedicated TX buffers are intended for message transmission under complete control of the CPU. Each dedicated TX buffer is configured with a specific Message ID. If multiple TX buffers are configured with the same Message ID, the TX buffer with the lowest buffer number is transmitted first.

If the data section is updated, a transmission is requested by an Add Request via CANFDx\_CHy\_TXBAR.ARn. The requested messages arbitrate internally with messages from an optional TX FIFO or TX Queue and externally with messages on the CAN bus, and are sent out according to their Message ID.

#### Addressing Dedicated TX Buffers

A dedicated TX buffer allocates an Element Size of 32-bit words in the message RAM as shown in the [Table 17-6](#). Therefore, the start address of a dedicated TX buffer in the message RAM is calculated by

(transmit buffer index (0 to 31) × Element Size) + TX Buffers Start Address (CANFDx\_CHy\_TXBC.TBSA[15:2]).

Table 17-6. TX Buffer/FIFO/Queue Element Size

| CANFDx_CHy_TXESC.TBDS[2:0] | Data Field [bytes] | Element Size [RAM Words] |
|----------------------------|--------------------|--------------------------|
| 000                        | 8                  | 4                        |
| 001                        | 12                 | 5                        |
| 010                        | 16                 | 6                        |
| 011                        | 20                 | 7                        |
| 100                        | 24                 | 8                        |
| 101                        | 32                 | 10                       |
| 110                        | 48                 | 14                       |
| 111                        | 64                 | 18                       |

### 17.3.5.3 TX FIFO

TX FIFO operation is configured by programming CANFDx\_CHy\_TXBC.TFQM to '0'. Messages stored in the TX FIFO are transmitted starting with the message referenced by the Get Index CANFDx\_CHy\_TXFQS.TFGI. After each transmission, the Get Index is incremented cyclically until the TX FIFO is empty. The TX FIFO enables transmission of messages with the same Message ID from different TX buffers in the order these messages are written to the TX FIFO. The M\_TTCAN calculates the TX FIFO Free Level CANFDx\_CHy\_TXFQS.TFFL as difference between Get and Put Index. It indicates the number of available (free) TX FIFO elements.

New transmit messages must be written to the TX FIFO starting with the TX buffer referenced by the Put Index CANFDx\_CHy\_TXFQS.TFQPI. An Add Request increments the Put Index to the next free TX FIFO element. When the Put Index reaches the Get Index, TX FIFO Full (CANFDx\_CHy\_TXFQS.TFQF = 1) is signaled. In this case no further messages should be written to the TX FIFO until the next message is transmitted and the Get Index is incremented.

When a single message is added to the TX FIFO, the transmission is requested by writing a '1' to the CANFDx\_CHy\_TXBAR bit related to the TX buffer referenced by the TX FIFO's Put Index.

When multiple (n) messages are added to the TX FIFO, they are written to n consecutive TX buffers starting with the Put Index. The transmissions are then requested via CANFDx\_CHy\_TXBAR. The Put Index is then cyclically incremented by n. The number of requested TX buffers should not exceed the number of free TX buffers as indicated by the TX FIFO Free Level.

When a transmission request for the TX buffer referenced by the Get Index is canceled, the Get Index is incremented to the next TX buffer with pending transmission request and the TX FIFO Free Level is recalculated. When transmission cancellation is applied to any other TX buffer, the Get Index and the FIFO Free Level remain unchanged.

A TX FIFO element allocates Element Size 32-bit words in the message RAM as shown in [Table 17-6](#). Therefore, the start address of the next available (free) TX FIFO buffer is calculated by adding TX FIFO/Queue Put Index CANFDx\_CHy\_TXFQS.TFQPI (0...31) • Element Size to the TX buffer Start Address CANFDx\_CHy\_TXBC.TBSA.

### 17.3.5.4 TX Queue

TX Queue operation is configured by programming CANFDx\_CHy\_TXBC.TFQM to '1'. Messages stored in the TX Queue are transmitted starting with the message with the lowest Message ID (highest priority). If multiple queue buffers are configured with the same Message ID, the queue buffer with the lowest buffer number is transmitted first.

New messages must be written to the TX buffer referenced by the Put Index CANFDx\_CHy\_TXFQS.TFQPI. An Add Request cyclically increments the Put Index to the next free TX buffer. If the TX Queue is full (CANFDx\_CHy\_TXFQS.TFQF = 1), the Put Index is not valid and no further message should be written to the TX Queue until at least one of the requested messages is sent out or a pending transmission request is canceled.

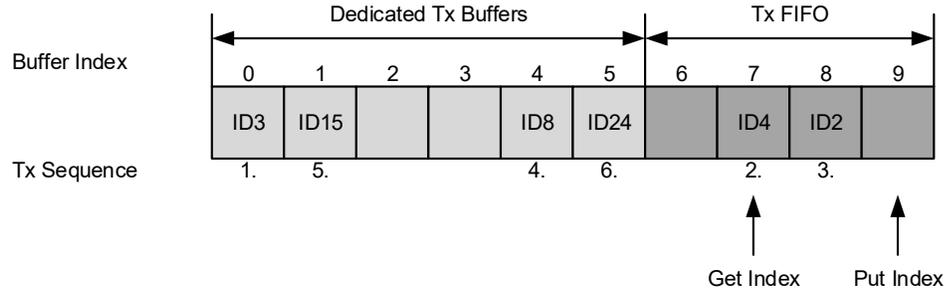
The application may use the CANFDx\_CHy\_TXBRP register instead of the Put Index and may place messages to any TX buffer without pending transmission request.

A TX Queue buffer allocates element size of 32-bit words in the message RAM as shown in [Table 17-6](#). Therefore, the start address of the next available (free) TX Queue buffer is calculated by adding TX FIFO/Queue Put Index CANFDx\_CHy\_TXFQS.TFQPI (0...31) × Element Size to the TX buffer Start Address CANFDx\_CHy\_TXBC.TBSA.

### 17.3.5.5 Mixed Dedicated TX Buffers/TX FIFO

In this case, the TX Buffers section in the message RAM is subdivided into a set of dedicated TX buffers and a TX FIFO. The number of dedicated TX buffers is configured by CANFDx\_CHy\_TXBC.NDTB. The number of TX buffers assigned to the TX FIFO is configured by CANFDx\_CHy\_TXBC.TFQS. If CANFDx\_CHy\_TXBC.TFQS is programmed to zero, only the dedicated TX buffers are used.

Figure 17-12. Example of Mixed Configuration Dedicated TX Buffers/TX FIFO



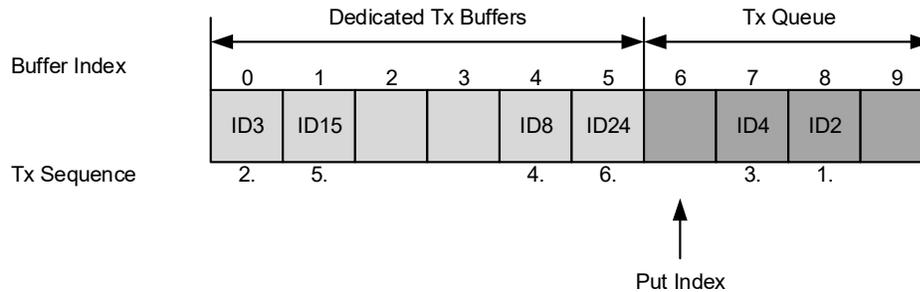
TX prioritization:

- Scan dedicated TX buffers and oldest pending TX FIFO buffer (referenced by CANFDx\_CHy\_TXFS.TFGI)
- Buffer with the lowest Message ID gets highest priority and is transmitted next

### 17.3.5.6 Mixed Dedicated TX Buffers/TX Queue

In this case the TX Buffers section in the message RAM is subdivided into a set of dedicated TX buffers and a TX Queue. The number of dedicated TX buffers is configured by CANFDx\_CHy\_TXBC.NDTB. The number of TX Queue buffers is configured by CANFDx\_CHy\_TXBC.TFQS. In case CANFDx\_CHy\_TXBC.TFQS is programmed to zero, only dedicated TX buffers are used.

Figure 17-13. Example of Mixed Configuration Dedicated TX Buffers/TX Queue



TX prioritization:

- Scan all TX buffers with activated transmission request
- TX buffer with the lowest Message ID gets highest priority and is transmitted next

### 17.3.5.7 Transmit Cancellation

The M\_TTCAN supports transmit cancellation. This feature is especially intended for gateway applications and AUTOSAR-based applications. To cancel a requested transmission from a dedicated TX buffer or a TX Queue buffer the host must write a '1' to the corresponding bit position (number of TX buffers) of the CANFDx\_CHy\_TXBCR register. Transmit cancellation is not intended for TX FIFO operation.

Successful cancellation is signaled by setting the corresponding bit of the CANFDx\_CHy\_TXBCF register to '1'.

In case a transmit cancellation is requested while a transmission from a TX buffer is ongoing, the corresponding CANFDx\_CHy\_TXBRP bit remains set as long as the transmission is in progress. If the transmission was successful, the corresponding CANFDx\_CHy\_TXBTO and CANFDx\_CHy\_TXBCF bits are set. If the transmission was not successful, it is not repeated and only the corresponding CANFDx\_CHy\_TXBCF bit is set.

**Note:** If a pending transmission is canceled immediately before this transmission is started, there follows a short time window where no transmission is started even if another message is also pending in this node. This may enable another node to transmit a message, which may have a lower priority than the second message in this node.

### 17.3.5.8 TX Event Handling

To support TX event handling, the M\_TTCAN has implemented a TX Event FIFO. After the M\_TTCAN has transmitted a message on the CAN bus, the Message ID and timestamp are stored in a TX Event FIFO element. To link a TX event to a TX Event FIFO element, the Message Marker from the transmitted TX buffer is copied into the TX Event FIFO element.

The TX Event FIFO can be configured to a maximum of 32 elements. The TX Event FIFO element is described in [TX Event FIFO Element on page 210](#).

The purpose of the TX Event FIFO is to decouple handling transmit status information from transmit message handling; that is, a TX buffer holds only the message to be transmitted, while the transmit status is stored separately in the TX Event FIFO. This has the advantage, especially when operating a dynamically managed transmit queue, that a TX buffer can be used for a new message immediately after successful transmission. There is no need to save transmit status information from a TX buffer before overwriting that TX buffer.

When a TX Event FIFO full condition is signaled by CANFDx\_CHy\_IR.TEFF, no further elements are written to the TX Event FIFO until at least one element is read out and the TX Event FIFO Get Index is incremented. In case a TX event occurs while the TX Event FIFO is full, this event is discarded and interrupt flag CANFDx\_CHy\_IR.TEFL is set.

To avoid a TX Event FIFO overflow, the TX Event FIFO watermark can be used. When the TX Event FIFO fill level reaches the TX Event FIFO watermark configured by CANFDx\_CHy\_TXEFC.EFWM, interrupt flag CANFDx\_CHy\_IR.TEFW is set.

When reading from the TX Event FIFO, the TX Event FIFO Get Index CANFDx\_CHy\_TXEFS.EFGI must be added twice to the TX Event FIFO start address CANFDx\_CHy\_TXEFC.EFSA.

## 17.3.6 FIFO Acknowledge Handling

The Get indices of RX FIFO 0, RX FIFO 1, and the TX Event FIFO are controlled by the corresponding FIFO Acknowledge Index.

When RX FIFO top pointer hardware logic is used, it updates the RX FIFO Acknowledge Index. After CANFDx\_CHy\_RXFTOPn\_DATA is read, the Acknowledge Index (CANFDx\_CHy\_RXFnA.FnA) is updated automatically, which will eventually set the FIFO Get Index to the FIFO Acknowledge Index plus one and thereby updates the FIFO Fill Level.

When the application does not use RX FIFO top pointer logic, the Acknowledge Index must be updated. This can be done using one of the following two use cases:

- When only a single element is read from the FIFO (the one being pointed to by the Get Index), the Get Index value is written to the FIFO Acknowledge Index.
- When a sequence of elements is read from the FIFO, it is sufficient to write the FIFO Acknowledge Index only once at the end of that read sequence (value is the index of the last element read), to update the FIFO's Get Index.

Because the CPU has free access to the M\_TTCAN's message RAM, take care when reading FIFO elements in an arbitrary order (Get Index not considered). This may be useful when reading a high-priority message from one of the two RX FIFOs. In this case the FIFO Acknowledge Index should not be written because this will set the Get Index to a wrong position and alter the FIFO's Fill Level. Some older FIFO elements are lost.

**Note:** The application must ensure that a valid value is written to the FIFO Acknowledge Index. The M\_TTCAN does not check for erroneous values.

### 17.3.7 Configuring the CAN Bit Timing

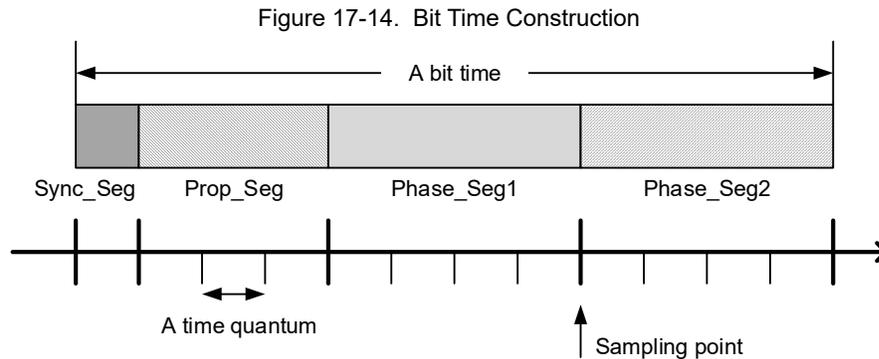
Each node in the CAN network has its own clock generator (usually a quartz oscillator). The time parameter of the bit time can be configured individually for each CAN node. Even if each CAN node's oscillator has a different period, a common bit rate can be generated.

The oscillator frequencies vary slightly because of changes in temperature or voltage, or deterioration of components. As long as the frequencies vary only within the tolerance range of the oscillators, the CAN nodes can compensate for the different bit rates by resynchronizing to the bit stream.

#### 17.3.7.1 CAN Bit Timing

The CAN FD operation defines two bit times – nominal bit time and data bit time. The nominal bit time is for the arbitration phase. The data bit time has an equal or shorter length and can be used to accelerate the data phase (see [CAN FD Operation on page 183](#)).

The basic construction of a bit time is shared with both the nominal and data bit times. The bit time can be divided into four segments according to the CAN specifications (see [Figure 17-14](#)): the synchronization segment (Sync\_Seg), the propagation time segment (Prop\_Seg), the phase buffer segment 1 (Phase\_Seg1), and the phase buffer segment 2 (Phase\_Seg2). The sample point at which the bus level is read and interpreted as the value of that respective bit, is located at the end of Phase\_Seg1.



Each segment consists of a programmable number of time quanta, which is a multiple of the time quantum that is defined by `canfd.clock_can[0]` and a prescaler. The values and prescalers used to define these parameters differ for the nominal and data bit times, and are configured by `CANFDx_CHy_NBTP` (Nominal Bit Timing and Prescaler Register) and `CANFDx_CHy_DBTP` (Data Bit Timing and Prescaler Register) as shown in [Table 17-7](#).

Table 17-7. Bit Time Parameters

| Parameter                                      | Description   |
|--|---|
| Time quantum<br>tq (nominal) and<br>tqd (data) | Time quantum. Derived by multiplying the basic unit time quanta (the CAN FD peripheral clock period) with the respective prescaler.<br>The time quantum is configured by the CAN FD controller as<br>nominal: $tq = (CANFDx\_CHy\_NBTP.NBRP[8:0] + 1) \times canfd.clock\_can[0]$ period<br>data: $tqd = (CANFDx\_CHy\_DBTP.DBRP[4:0] + 1) \times canfd.clock\_can[0]$ period |
| Sync_Seg                                       | Sync_Seg is fixed to one time quantum as defined by the CAN specifications and is not configurable (inherently built into the CAN FD controller).<br>nominal: 1 tq<br>data: 1 tqd   |
| Prop_Seg                                       | Prop_Seg is the part of the bit time that is used to compensate for the physical delay times within the network. The CAN FD controller configures the sum of Prop_Seg and Phase_Seg1 with a single parameter:<br>nominal: $Prop\_Seg + Phase\_Seg1 = CANFDx\_CHy\_NBTP.NTSEG1[7:0] + 1$<br>data: $Prop\_Seg + Phase\_Seg1 = CANFDx\_CHy\_DBTP.DTSEG1[4:0] + 1$                |
| Phase_Seg1                                     | Phase_Seg1 is used to compensate for edge phase errors before the sampling point. Can be lengthened by the resynchronization jump width.<br>The sum of Prop_Seg and Phase_Seg1 is configured by the CAN FD controller as<br>nominal: $CANFDx\_CHy\_NBTP.NTSEG1[7:0] + 1$<br>data: $CANFDx\_CHy\_DBTP.DTSEG1[4:0] + 1$   |
| Phase_Seg2                                     | Phase_Seg2 is used to compensate for edge phase errors after the sampling point. Can be shortened by the resynchronization jump width.<br>Phase_Seg2 is configured by the CAN FD controller as<br>nominal: $CANFDx\_CHy\_NBTP.NTSEG2[6:0] + 1$<br>data: $CANFDx\_CHy\_DBTP.DTSEG2[3:0] + 1$   |
| SJW  | Resynchronization Jump Width. Used to adjust the length of Phase_Seg1 and Phase_Seg2. SJW will not be longer than either Phase_Seg1 or Phase_Seg2.<br>SJW is configured by the CAN FD controller as<br>nominal: $CANFDx\_CHy\_NBTP.NSJW[6:0] + 1$<br>data: $CANFDx\_CHy\_DBTP.DSJW[3:0] + 1$  |

These relations result in the following equations for the nominal and data bit times:

Nominal Bit time

$$= [Sync\_Seg + Prop\_Seg + Phase\_Seg1 + Phase\_Seg2] \times tq$$

$$= [1 + (CANFDx\_CHy\_NBTP.NTSEG1[7:0] + 1) + (CANFDx\_CHy\_NBTP.NTSEG2[6:0] + 1)] \times [(CANFDx\_CHy\_NBTP.NBRP[8:0] + 1) \times canfd.clock\_can[y] \text{ period}]$$

Data Bit time

$$= [1 + (CANFDx\_CHy\_DBTP.DTSEG1[4:0] + 1) + (CANFDx\_CHy\_DBTP.DTSEG2[3:0] + 1)] \times [(CANFDx\_CHy\_DBTP.DBRP[4:0] + 1) \times canfd.clock\_can[0] \text{ period}]$$

**Note:** The Information Processing Time (IPT) of the CAN FD controller is zero; this means that the data for the next bit is available at the first CAN clock edge after the sample point. Therefore, the IPT does not have to be accounted for when configuring Phase\_Seg2, which is the maximum of Phase\_Seg1 and the IPT.

### 17.3.7.2 CAN Bit Rates

The bit rate is the inverse of bit time; therefore, the nominal bit rate is:

$$1 / \{ [1 + (\text{CANFDx\_CHy\_NBTP.NTSEG1}[7:0] + 1) + (\text{CANFDx\_CHy\_NBTP.NTSEG2}[6:0] + 1)] \times \{ (\text{CANFDx\_CHy\_NBTP.NBRP}[8:0] + 1) \times \text{canfd.clock\_can}[0] \text{ period} \} \}$$

and the data bit rate is:

$$1 / \{ [1 + (\text{CANFDx\_CHy\_DBTP.DTSEG1}[4:0] + 1) + (\text{CANFDx\_CHy\_DBTP.DTSEG2}[3:0] + 1)] \times \{ (\text{CANFDx\_CHy\_DBTP.DBRP}[4:0] + 1) \times \text{canfd.clock\_can}[0] \text{ period} \} \}$$

The above formulae indicate that the bit rates of the CAN FD controller depends on the CAN FD peripheral clock (canfd.clock\_can[0]) period, and the range each parameter can be configured to. The following tables list examples of the configurable bit rates at varying CAN clock frequencies. Empty boxes indicate that the desired bit rate cannot be configured at the specified input CAN clock frequency.

Figure 17-15. Example Configuration for Nominal Bit Rates

| CAN clock frequency | 8MHz                        |                          | 10MHz                        |                          | 16MHz                                |                          | 20MHz                                 |                          | 32MHz   |                              | 40MHz  |                              |
|---------------------|-----------------------------|--------------------------|------------------------------|--------------------------|--------------------------------------|--------------------------|---------------------------------------|--------------------------|---|------------------------------|--|------------------------------|
| configuration       | Number of tqs per bit time  | CANFDx_CHy_NBTP.NBRP + 1 | Number of tqs per bit time   | CANFDx_CHy_NBTP.NBRP + 1 | Number of tqs per bit time           | CANFDx_CHy_NBTP.NBRP + 1 | Number of tqs per bit time            | CANFDx_CHy_NBTP.NBRP + 1 | Number of tqs per bit time                    | CANFDx_CHy_NBTP.NBRP + 1     | Number of tqs per bit time                     | CANFDx_CHy_NBTP.NBRP + 1     |
| nominal bit rate    |                             |                          |                              |                          |                                      |                          |                                       |                          |   |                              |  |                              |
| 125Kbps             | 64tq<br>32tq<br>16tq<br>8tq | 1<br>2<br>4<br>8         | 80tq<br>40tq<br>20tq<br>10tq | 1<br>2<br>4<br>8         | 128tq<br>64tq<br>32tq<br>16tq<br>8tq | 1<br>2<br>4<br>8<br>16   | 160tq<br>80tq<br>40tq<br>20tq<br>10tq | 1<br>2<br>4<br>8<br>16   | 256tq<br>128tq<br>64tq<br>32tq<br>16tq<br>8tq | 1<br>2<br>4<br>8<br>16<br>32 | 320tq<br>160tq<br>80tq<br>40tq<br>20tq<br>10tq | 1<br>2<br>4<br>8<br>16<br>32 |
| 250Kbps             | 32tq<br>16tq<br>8tq         | 1<br>2<br>4              | 40tq<br>20tq<br>10tq         | 1<br>2<br>4              | 64tq<br>32tq<br>16tq<br>8tq          | 1<br>2<br>4<br>8         | 80tq<br>40tq<br>20tq<br>10tq          | 1<br>2<br>4<br>8         | 128tq<br>64tq<br>32tq<br>16tq<br>8tq          | 1<br>2<br>4<br>8<br>16       | 160tq<br>80tq<br>40tq<br>20tq<br>10tq          | 1<br>2<br>4<br>8<br>16       |
| 500Kbps             | 16tq<br>8tq                 | 1<br>2                   | 20tq<br>10tq                 | 1<br>2                   | 32tq<br>16tq<br>8tq                  | 1<br>2<br>4              | 40tq<br>20tq<br>10tq                  | 1<br>2<br>4              | 64tq<br>32tq<br>16tq<br>8tq                   | 1<br>2<br>4<br>8             | 80tq<br>40tq<br>20tq<br>10tq                   | 1<br>2<br>4<br>8             |
| 1Mbps               | 8tq                         | 1                        | 10tq                         | 1                        | 16tq<br>8tq                          | 1<br>2                   | 20tq<br>10tq                          | 1<br>2                   | 32tq<br>16tq<br>8tq                           | 1<br>2<br>4                  | 40tq<br>20tq<br>10tq                           | 1<br>2<br>4                  |

Figure 17-16. Example Configuration for Data Bit Rates

| CAN clock frequency | 8MHz                        |                     | 10MHz                       |                     | 16MHz                       |                     | 20MHz                       |                     | 32MHz                       |                     | 40MHz                       |                     |
|---------------------|-----------------------------|---------------------|-----------------------------|---------------------|-----------------------------|---------------------|-----------------------------|---------------------|-----------------------------|---------------------|-----------------------------|---------------------|
| configuration       | Number of tqds per bit time |                     |
| data bit rate       | CANFDx_CHy_DBRP + 1         | CANFDx_CHy_DBRP + 1 |
| 500Kbps             | 16tqd<br>8tqd               | 1<br>2              | 20tqd<br>10tqd              | 1<br>2              | 32tqd<br>16tqd<br>8tqd      | 1<br>2<br>4         | 40tqd<br>20tqd<br>10tqd     | 1<br>2<br>4         | 32tqd<br>16tqd<br>8tqd      | 2<br>4<br>8         | 40tqd<br>20tqd<br>10tqd     | 2<br>4<br>8         |
| 1Mbps               | 8tqd                        | 1                   | 10tqd                       | 1                   | 16tqd<br>8tqd               | 1<br>2              | 20tqd<br>10tqd              | 1<br>2              | 32tqd<br>16tqd<br>8tqd      | 1<br>2<br>4         | 40tqd<br>20tqd<br>10tqd     | 1<br>2<br>4         |
| 2Mbps               | -                           | -                   | -                           | -                   | 8tqd                        | 1                   | 10tqd                       | 1                   | 16tqd<br>8tqd               | 1<br>2              | 20tqd<br>10tqd              | 1<br>2              |
| 4Mbps               | -                           | -                   | -                           | -                   | -                           | -                   | -                           | -                   | 8tqd                        | 1                   | 10tqd                       | 1                   |
| 5Mbps               | -                           | -                   | -                           | -                   | -                           | -                   | -                           | -                   | -                           | -                   | 8tqd                        | 1                   |

**Note:** The user must configure the CAN bit timings to comply with the corresponding CAN standards to ensure proper communication on the CAN bus.

## 17.4 Message RAM

Message RAM (MRAM) in Traveo II family devices is shared among multiple M\_TTCAN channels present in the M\_TTCAN group. Refer to the device datasheet for the supported number of M\_TTCAN groups, M\_TTCAN channels in each group, and the total message RAM allocated to each group. Each M\_TTCAN channel in the group can configure its required message RAM according to application requirements.

The message RAM stores RX/TX messages and filter configurations.

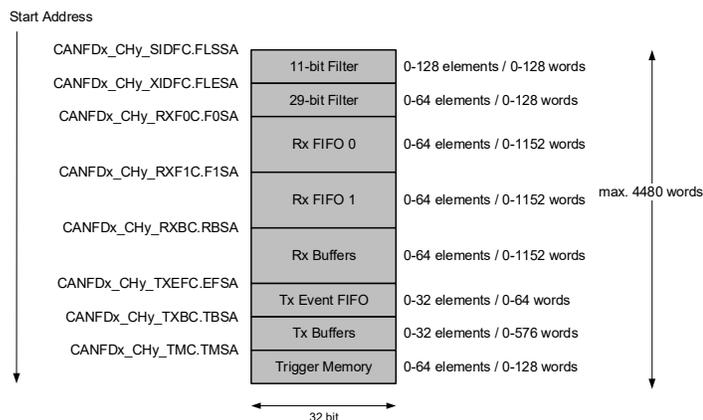
**Notes:**

- The message RAM should be made zero before configuration of the CAN FD controller to prevent bit errors when reading uninitialized words, and to avoid unexpected filter element configurations in the message RAM.
- Unused message RAM cannot be used for general purposes.

### 17.4.1 Message RAM Configuration

The message RAM has a width of 32 bits. The CAN FD controller can be configured to allocate up to 4480 words in the message RAM (note that the number of words that can be used will be limited by the size of the actual message RAM). It is not necessary to configure each of the sections listed in Figure 17-17, nor is there any restriction with respect to the sequence of the sections.

Figure 17-17. Message RAM Configuration



The CAN FD controller addresses the message RAM in 32-bit words, not single bytes. The configurable start addresses are 32-bit word addresses – only bits 15 to 2 are evaluated, the two least significant bits are ignored.

**Notes:**

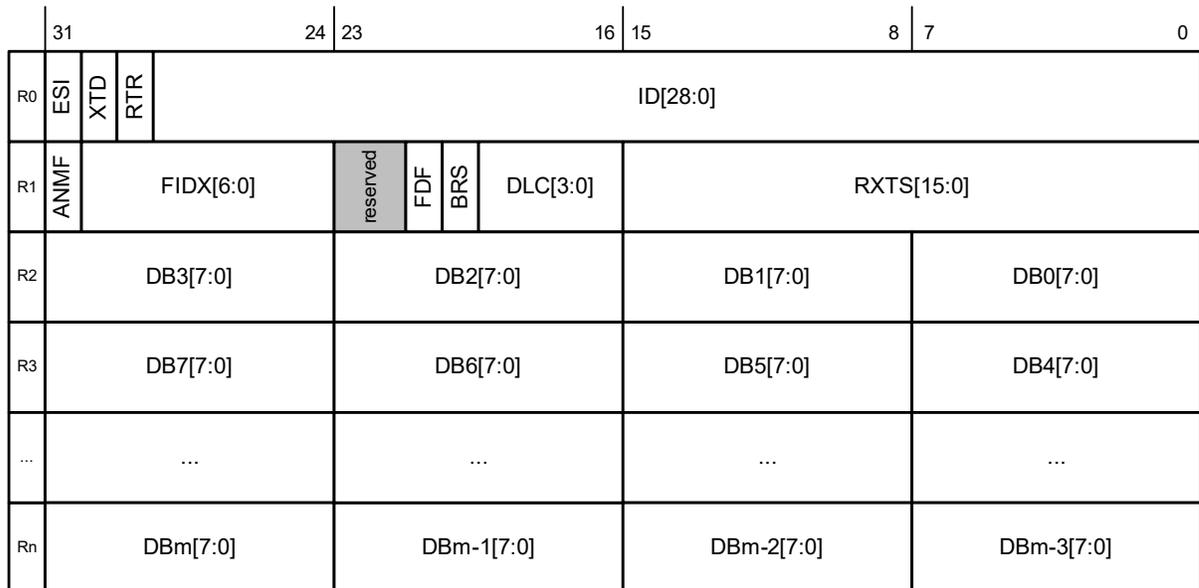
- The CAN FD controller does not check for erroneous configuration of the message RAM. The configuration of the start addresses of different sections and the number of elements of each section should be done carefully to avoid falsification or loss of data.
- Message RAM is accessible by both M\_TTCAN and CPU. Dynamic round-robin scheme is implemented to allocate access.

### 17.4.2 RX Buffer and FIFO Element

An RX buffer and FIFO element is a block of 32-bit words, which holds the data and status of a received frame that was stored in the message RAM.

Up to 64 RX buffers and two RX FIFOs can be configured in the message RAM. Each RX FIFO section can be configured to store up to 64 received messages. The structure of an RX buffer and FIFO element is shown in Figure 17-18. The element size can be configured to store CAN FD messages with up to 64 bytes data field via register CANFDx\_CHy\_RXESC (RX buffer/FIFO element Size Configuration).

Figure 17-18. RX Buffer and FIFO



R0 [bit31] ESI: Error State Indicator

| Bit | Description                         |
|-----|-------------------------------------|
| 0   | Transmitting node is error active.  |
| 1   | Transmitting node is error passive. |

R0 [bit30] XTD: Extended Identifier

Signals to the CPU whether the received frame has a standard or extended identifier.

| Bit | Description                 |
|-----|-----------------------------|
| 0   | 11-bit standard identifier. |
| 1   | 29-bit extended identifier. |

R0 [bit29] RTR: Remote Transmission Request

Signals to the CPU whether the received frame is a data frame or a remote frame.

| Bit | Description                       |
|-----|-----------------------------------|
| 0   | Received frame is a data frame.   |
| 1   | Received frame is a remote frame. |

**Note:** There are no remote frames in CAN FD format. In CAN FD frames (FDF = 1), the dominant RRS (Remote Request Substitution) bit replaces bit RTR (Remote Transmission Request).

R0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].

R1 [bit31] ANMF: Accepted Non-matching Frame

Acceptance of non-matching frames may be enabled via CANFDx\_CHy\_GFC.ANFS[1:0] (Accept Non-matching Frames Standard) and CANFDx\_CHy\_GFC.ANFE[1:0] (Accept Non-matching Frames Extended).

| Bit | Description   |
|-----|---|
| 0   | Received frame matching filter index FIDX.          |
| 1   | Received frame did not match any RX filter element. |

R1 [bit30:24] FIDX[6:0]: Filter Index

| FIDX[6:0] | Description  |
|-----------|--|
| 0-127     | Index of matching RX acceptance filter element (invalid if ANMF = 1). Range is 0 to List Size Standard/Extended minus 1 (CANFDx_CHy_SIDFC.LSS - 1 resp. CANFDx_CHy_XIDFC.LSE - 1). |

R1 [bit23:22] Reserved: Reserved Bits

When writing, always write '0'. The read value is undefined.

R1 [bit21] FDF: Extended Data Length

| Bit | Description               |
|-----|---------------------------|
| 0   | Classic CAN frame format. |
| 1   | CAN FD frame format.      |

R1 [bit20] BRS: Bit Rate Switch

| Bit | Description                                |
|-----|--|
| 0   | Frame received without bit rate switching. |
| 1   | Frame received with bit rate switching.    |

R1 [bit19:16] DLC[3:0]: Data Length Code

| DLC[3:0] | Description  |
|----------|--|
| 0-8      | Classic CAN + CAN FD: received frame has 0-8 data bytes.   |
| 9-15     | Classic CAN: received frame has 8 data bytes.<br>CAN FD: received frame has 12/16/20/24/32/48/64 data bytes. See <a href="#">Table 17-1</a> for details. |



T0 [bit31] ESI: Error State Indicator

| Bit | Description  |
|-----|--|
| 0   | ESI bit in CAN FD format depends only on error passive flag. |
| 1   | ESI bit in CAN FD format transmitted recessive.              |

**Note:** The ESI bit of the transmit buffer is ORed with the error passive flag to decide the value of the ESI bit in the transmitted FD frame. As required by the CAN FD protocol specification, an error active node may optionally transmit the ESI bit recessive, but an error passive node will always transmit the ESI bit recessive.

T0 [bit30] XTD: Extended Identifier

| Bit | Description                 |
|-----|-----------------------------|
| 0   | 11-bit standard identifier. |
| 1   | 29-bit extended identifier. |

T0 [bit29] RTR: Remote Transmission Request

| Bit | Description            |
|-----|------------------------|
| 0   | Transmit data frame.   |
| 1   | Transmit remote frame. |

**Note:** When RTR = 1, the CAN FD controller transmits a remote frame according to ISO11898-1, even if FD Operation Enable (CANFDx\_CHy\_CCCR.FDOE) enables the transmission in CAN FD format.

T0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier has to be written to ID[28:18].

T1 [bit31:24] MM[7:0]: Message Marker

Written by CPU during TX buffer configuration. Copied into TX Event FIFO element for identification of TX message status.

T1 [bit23] EFC: Event FIFO Control

| Bit | Description            |
|-----|------------------------|
| 0   | Don't store TX events. |
| 1   | Store TX events.       |

T1 [bit22] Reserved: Reserved Bit

When writing, always write '0'. The read value is undefined.

T1 [bit21] FDF: FD Format

| Bit | Description                              |
|-----|--|
| 0   | Frame transmitted in Classic CAN format. |
| 1   | Frame transmitted in CAN FD format.      |

T1 [bit20] BRS: Bit Rate Switching

| Bit | Description   |
|-----|---|
| 0   | CAN FD frames transmitted without bit rate switching. |
| 1   | CAN FD frames transmitted with bit rate switching.    |

**Note:** Bits ESI, FDF, and BRS are only evaluated when CAN FD operation is enabled CANFDx\_CHy\_CCCR.FDOE = 1. Bit BRS is only evaluated when in addition CANFDx\_CHy\_CCCR.BRSE = 1. See [Table 17-5](#) for details of bits FDF and BRS.

T1 [bit19:16] DLC[3:0]: Data Length Code

| DLC[3:0] | Description  |
|----------|--|
| 0-8      | Classic CAN + CAN FD: transmit frame has 0-8 data bytes. |
| 9-15     | Classic CAN: transmit frame has 8 data bytes.            |

CAN FD: transmit frame has 12/16/20/24/32/48/64 data bytes.

T1 [bit15:0] Reserved: Reserved Bits

When writing, always write '0'. The read value is undefined.

|               |             |               |
|---------------|-------------|---------------|
| T2 [bit31:24] | DB3[7:0]:   | Data Byte 3   |
| T2 [bit23:16] | DB2[7:0]:   | Data Byte 2   |
| T2 [bit15:8]  | DB1[7:0]:   | Data Byte 1   |
| T2 [bit7:0]   | DB0[7:0]:   | Data Byte 0   |
| T3 [bit31:24] | DB7[7:0]:   | Data Byte 7   |
| T3 [bit23:16] | DB6[7:0]:   | Data Byte 6   |
| T3 [bit15:8]  | DB5[7:0]:   | Data Byte 5   |
| T3 [bit7:0]   | DB4[7:0]:   | Data Byte 4   |
| ...           | ...         | ...           |
| Tn [bit31:24] | DBm[7:0]:   | Data Byte m   |
| Tn [bit23:16] | DBm-1[7:0]: | Data Byte m-1 |
| Tn [bit15:8]  | DBm-2[7:0]: | Data Byte m-2 |
| Tn [bit7:0]   | DBm-3[7:0]: | Data Byte m-3 |

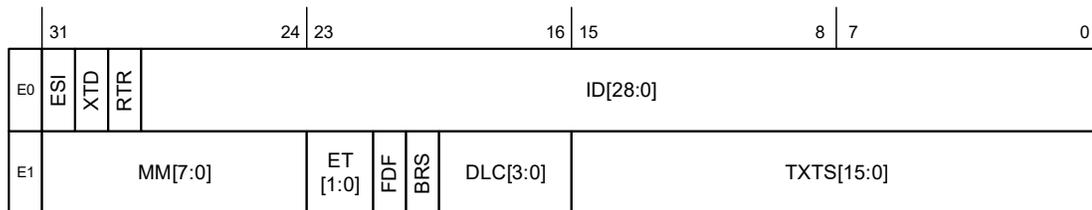
**Notes:**

- Depending on the configuration of the element size (TXESC), Tn will vary from n = 3 to 17.
- m is a function of n:  $m = (n - 1) \times 4 - 1$ .

### 17.4.4 TX Event FIFO Element

Each TX Event FIFO Element stores information about transmitted messages. By reading the TX Event FIFO, the CPU gets this information in the order the messages were transmitted. Status information about the TX Event FIFO can be obtained from register CANFDx\_CHy\_TXEFS (TX Event FIFO Status).

Figure 17-20. TX Event FIFO Element



E0 [bit31] ES!: Error State Indicator

| Bit | Description                         |
|-----|-------------------------------------|
| 0   | Transmitting node is error active.  |
| 1   | Transmitting node is error passive. |

E0 [bit30] XTD: Extended Identifier

| Bit | Description                 |
|-----|-----------------------------|
| 0   | 11-bit standard identifier. |
| 1   | 29-bit extended identifier. |

E0 [bit29] RTR: Remote Transmission Request

| Bit | Description               |
|-----|---------------------------|
| 0   | Data frame transmitted.   |
| 1   | Remote frame transmitted. |

E0 [bit28:0] ID[28:0]: Identifier

Standard or extended identifier depending on bit XTD. A standard identifier is stored into ID[28:18].

E1 [bit31:24] MM[7:0]: Message Marker

Copied from TX buffer into TX Event FIFO element for identification of TX message status.

E1 [bit23:22] ET[1:0]: Event Type

| ET[1:0] | Description   |
|---------|---|
| 00      | Reserved.   |
| 01      | TX event.   |
| 10      | Transmission in spite of cancellation.<br>Always set for transmissions in DAR mode (Disable Automatic Retransmission mode). |
| 11      | Reserved.   |

E1 [bit21] FDF: FD Format

| Bit | Description                                   |
|-----|---|
| 0   | Classic CAN frame format.                     |
| 1   | CAN FD frame format (new DLC-coding and CRC). |

E1 [bit20] BRS: Bit Rate Switching

| Bit | Description                                   |
|-----|---|
| 0   | Frame transmitted without bit rate switching. |
| 1   | Frame transmitted with bit rate switching.    |

E1 [bit19:16] DLC[3:0]: Data Length Code

| DLC[3:0] | Description   |
|----------|---|
| 0-8      | Classic CAN + CAN FD: frame with 0-8 data bytes transmitted.  |
| 9-15     | Classic CAN: frame with 8 data bytes transmitted.<br>CAN FD: frame with 12/16/20/24/32/48/64 data bytes transmitted.<br>See <a href="#">Table 17-1</a> for details. |

E1 [bit15:0] TXTS[15:0]: TX Timestamp

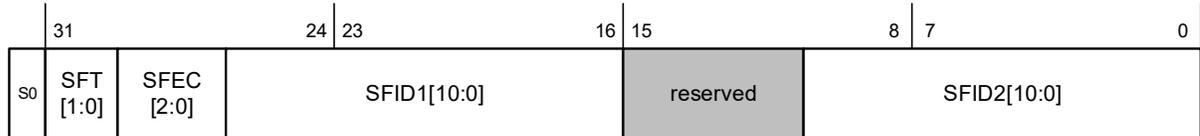
Timestamp Counter value captured on start-of-frame transmission. Resolution depending on configuration of the Shared Timestamp Counter Pre-scaler CANFDx\_TS\_CTL.PRESCALE[15:0].

### 17.4.5 Standard Message ID Filter Element

A Standard Message ID Filter Element consists of a single 32-bit word, and can be configured as a range filter, dual filter, classic bit mask filter, or filter for a single dedicated ID, for messages with 11-bit standard IDs.

Up to 128 filter elements can be configured for 11-bit standard IDs. When accessing a Standard Message ID Filter element, its address is: Filter List Standard Start Address (CANFDx\_CHy\_SIDFC.FLSSA[15:2]) + index of the filter element (0 to 127).

Figure 17-21. Standard Message ID Filter



S0 [bit31:30] SFT[1:0]: Standard Filter Type

| SFT[1:0] | Description   |
|----------|---|
| 00       | Range filter from SFID1[10:0] to SFID2[10:0] (SFID2[10:0] ≥ received ID ≥ SFID1[10:0]).   |
| 01       | Dual ID filter for SFID1[10:0] or SFID2[10:0].  |
| 10       | Classic filter: SFID1[10:0] = filter, SFID2[10:0] = mask. Only those bits of SFID1[10:0] where the corresponding SFID2[10:0] bits are 1 are relevant. |
| 11       | Filter element disabled.  |

**Note:** With SFT = 11, the filter element is disabled and the acceptance filtering continues. (same behavior as with SFEC = 000)

S0 [bit29:27] SFEC[2:0]: Standard Filter Element Configuration

All enabled filter elements are used for acceptance filtering of standard frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

If SFEC[2:0] = 100, 101, or 110 a match sets interrupt flag CANFDx\_CHy\_IR.HPM (High Priority Message) and, if enabled, an interrupt is generated. In this case register CANFDx\_CHy\_HPMS (High Priority Message Status) is updated with the status of the priority match.

| SFEC[2:0] | Description  |
|-----------|--|
| 000       | Disable filter element.  |
| 001       | Store in RX FIFO 0 if filter matches.  |
| 010       | Store in RX FIFO 1 if filter matches.  |
| 011       | Reject ID if filter matches.   |
| 100       | Set priority if filter matches.  |
| 101       | Set priority and store in RX FIFO 0 if filter matches.                                 |
| 110       | Set priority and store in RX FIFO 1 if filter matches.                                 |
| 111       | Store into dedicated RX buffer or as debug message, configuration of SFT[1:0] ignored. |

S0 [bit26:16] SFID1[10:0]: Standard Filter ID 1

This bit field has a different meaning depending on the configuration of SFEC[2:0]:

- SFEC[2:0] = 001 to 110

Set SFID1[10:0] according to the SFT[1:0] setting.

- SFEC[2:0] = 111

SFID1[10:0] defines the ID of a standard dedicated RX buffer or debug message to be stored. The received identifiers must match, no masking mechanism is used.

S0 [bit15:11] Reserved: Reserved Bits

When writing, always write '0'. The read value is undefined.

S0 [bit10:0] SFID2[10:0]: Standard Filter ID 2

This bit field has a different meaning depending on the configuration of SFEC[2:0]:

- SFEC[2:0] = 001 to 110

Set SFID2[10:0] according to the SFT[1:0] setting

- SFEC[2:0] = 111

Filter for dedicated RX buffers or for debug messages SFID2[10:9] decides whether the received message is stored into a dedicated RX buffer or treated as message A, B, or C of the debug message sequence.

| SFID2[10:9] | Description                               |
|-------------|---|
| 00          | Store message into a dedicated RX buffer. |
| 01          | Debug Message A.                          |
| 10          | Debug Message B.                          |
| 11          | Debug Message C.                          |

SFID2[8:6] are reserved bits. When writing, always write '0'. The read value is undefined.

SFID2[5:0] defines the offset to the RX buffer Start Address CANFDx\_CHy\_RXBC.RBSA[15:2] to store a matching message.

**Note:** Debug message is used to debug on CAN feature.

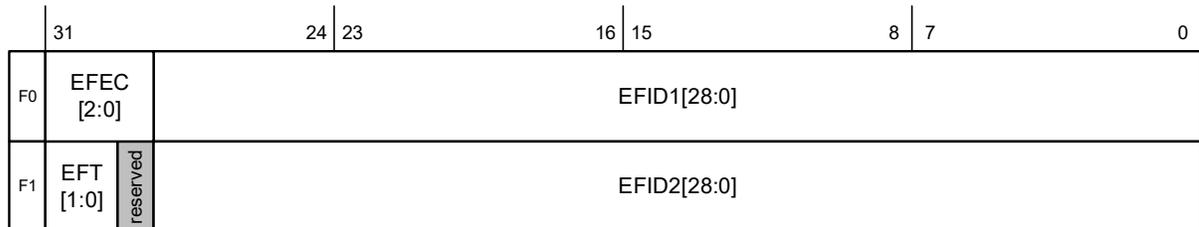
### 17.4.6 Extended Message ID Filter Element

An Extended Message ID Filter Element consists of two 32-bit words, and can be configured as a range filter, dual filter, classic bit mask filter, or filter for a single dedicated ID, for messages with 29-bit extended IDs.

Up to 64 filter elements can be configured for 29-bit extended IDs. When accessing an Extended Message ID Filter element, its address is

Filter List Extended Start Address (CANFDx\_CHy\_XIDFC.FLESA[15:2]) + 2 × index of the filter element (0 to 63).

Figure 17-22. Extended Message ID Filter



F0 [bit31:29] EFEC[2:0]: Extended Filter Element Configuration

All enabled filter elements are used for acceptance filtering of extended frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

If EFEC[2:0] = 100, 101, or 110 a match sets interrupt flag CANFDx\_CHy\_IR.HPM (High Priority Message) and, if enabled, an interrupt is generated. In this case register CANFDx\_CHy\_HPMS (High Priority Message Status) is updated with the status of the priority match.

| EFEC[2:0] | Description  |
|-----------|--|
| 000       | Disable filter element.  |
| 001       | Store in RX FIFO 0 if filter matches.  |
| 010       | Store in RX FIFO 1 if filter matches.  |
| 011       | Reject ID if filter matches.   |
| 100       | Set priority if filter matches.  |
| 101       | Set priority and store in RX FIFO 0 if filter matches.                                 |
| 110       | Set priority and store in RX FIFO 1 if filter matches.                                 |
| 111       | Store into dedicated RX buffer or as debug message, configuration of EFT[1:0] ignored. |

F0 [bit28:0] EFID1[28:0]: Extended Filter ID 1

This bit field has a different meaning depending on the configuration of EFEC[2:0].

- EFEC[2:0] = 001 to 110

Set EFID1[28:0] according to the EFT[1:0] setting.

- EFEC[2:0] = 11

EFID1[28:0] defines the ID of an extended dedicated RX buffer or debug message to be stored. The received identifiers must match, only XIDAM masking mechanism is used.

F1 [bit31:30] EFT[1:0]: Extended Filter Type

| EFT[1:0] | Description  |
|----------|--|
| 00       | Range filter from EFID1[28:0] to EFID2[28:0]<br>(EFID2[28:0] ≥ received ID ANDed with XIDAM ≥ EFID1[28:0]).  |
| 01       | Dual ID filter<br>Matches when EFID1[28:0] or EFID2[28:0] is equal to received ID ANDed with XIDAM.  |
| 10       | Classic filter: EFID1[28:0] = filter, EFID2[28:0] = mask.<br>Only those bits of EFID1[28:0] where the corresponding EFID2[28:0] bits are 1 are relevant.<br>Matches when the received ID ANDed with XIDAM is equal to EFID1[28:0] masked by EFID2[28:0]. |
| 11       | Range filter from EFID1[28:0] to EFID2[28:0]<br>(EFID2[28:0] ≥ EFID1[28:0]), XIDAM mask not applied.   |

F1 [bit29] Reserved: Reserved Bit

When writing, always write '0'. The read value is undefined.

F1 [bit28:0] EFID2[28:0]: Extended Filter ID 2

This bit field has a different meaning depending on the configuration of EFEC[2:0]:

- EFEC[2:0] = 001 to 110

Set EFID2[28:0] according to the EFT[1:0] setting

- EFEC[2:0] = 111

EFID2[28:0] is used to configure this filter for dedicated RX buffers or for debug messages

EFID2[28:11] are reserved bits. When writing, always write '0'. The read value is undefined.

EFID2[10:9] decides whether the received message is stored into a dedicated RX buffer or treated as message A, B, or C of the debug message sequence.

| EFID2[10:9] | Description                               |
|-------------|---|
| 00          | Store message into a dedicated RX buffer. |
| 01          | Debug Message A.                          |
| 10          | Debug Message B.                          |
| 11          | Debug Message C.                          |

EFID2[8:6] are reserved bits. When writing, always write '0'. The read value is undefined.

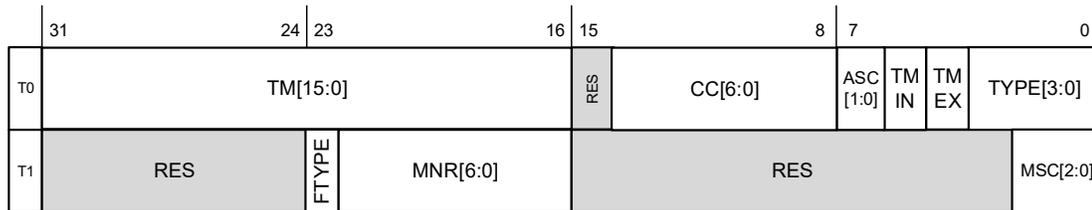
EFID2[5:0] defines the offset to the RX Buffer Start Address CANFDx\_CHy\_RXBC.RBSA[15:2] to store a matching message.

**Note:** Debug message is used to debug on CAN feature.

### 17.4.7 Trigger Memory Element

Up to 64 trigger memory elements can be configured. When accessing a trigger memory element, its address is the Trigger Memory Start Address CANFDx\_CHy\_TTTMC.TMSA plus the index of the trigger memory element (0...63).

Figure 17-23. Trigger Memory Element



T0 Bit 31:16 TM[15:0]: Time Mark

Cycle time for which the trigger becomes active.

T0 Bit 14:8 CC[6:0]: Cycle Code

Cycle count for which the trigger is valid. Ignored for trigger types Tx\_Ref\_Trigger, Tx\_Ref\_Trigger\_Gap, Watch\_Trigger, Watch\_Trigger\_Gap, and End\_of\_List.

| CC[6:0]   | Description   |
|-----------|---|
| 0b000000x | Valid for all cycles  |
| 0b000001c | Valid every second cycle at cycle count mod2 = c              |
| 0b00001cc | Valid every fourth cycle at cycle count mod4 = cc             |
| 0b0001ccc | Valid every eighth cycle at cycle count mod8 = ccc            |
| 0b001cccc | Valid every sixteenth cycle at cycle count mod16 = cccc       |
| 0b01ccccc | Valid every thirty-second cycle at cycle count mod32 = cccccc |
| 0b1cccccc | Valid every sixty-fourth cycle at cycle count mod64 = ccccccc |

T0 Bit 7:6 ASC[1:0]: Asynchronous Serial Communication

| ASC[1:0] | Description             |
|----------|-------------------------|
| 00       | No ASC operation        |
| 01       | Reserved, do not use    |
| 10       | Node is ASC receiver    |
| 11       | Node is ASC transmitter |

**Note:** ASC functionality is not supported in any Traveo II device

## T0 Bit 5 TMIN: Time Mark Event Internal

| TMIN | Description  |
|------|--|
| 0    | No Action  |
| 1    | CANFDx_CHy_TTIR.TTMI is set when trigger memory element becomes active |

## T0 Bit 4 TMEX: Time Mark Event External

| TMEX | Description  |
|------|--|
| 0    | No Action  |
| 1    | Pulse at output of Trigger Time Mark with the length of one canfd.clock_can[0] period is generated when the time mark of the trigger memory element becomes active and CANFDx_CHy_T-TOCN.TTMIE = 1 |

## T0 Bit 3:0 TYPE[3:0]: Trigger Type

| TYPE [3:0]    | Description  |
|---------------|--|
| 0000          | Tx_Ref_Trigger - valid when not in gap   |
| 0001          | Tx_Ref_Trigger_Gap - valid when in gap   |
| 0010          | Tx_Trigger_Single - starts a single transmission in an exclusive time window       |
| 0011          | Tx_Trigger_Continuous - starts continuous transmission in an exclusive time window |
| 0100          | Tx_Trigger_Arbitration - starts a transmission in an arbitrating time window       |
| 0101          | Tx_Trigger_Merged - starts a merged arbitration window                             |
| 0110          | Watch_Trigger - valid when not in gap  |
| 0111          | Watch_Trigger_Gap - valid when in gap  |
| 1000          | Rx_Trigger - check for reception   |
| 1001          | Time_Base_Trigger - only control TMIN, TMEX, and ASC                               |
| 1010 ... 1111 | End_of_List - illegal type, causes configuration error                             |

**Notes:**

- For ASC operation (ASC = 10, 11) only trigger types Rx\_Trigger and Time\_Base\_Trigger should be used.
- ASC operation is not supported in this device.

## T1 Bit 23 FTYPE: Filter Type

| FTYPE | Description                |
|-------|----------------------------|
| 0     | 11-bit standard message ID |
| 1     | 29-bit extended message ID |

## T1 Bit 22:16 MNR[6:0]: Message Number

Transmission: Trigger is valid for configured TX buffer number. Valid values are 0 to 31.

Reception: Trigger is valid for standard/extended message ID filter element number. Valid values are 0 to 63 and 0 to 127.

## T1 Bits 2:0 MSC[2:0]: Message Status Count

Counts scheduling errors for periodic messages in exclusive time windows. It has no function for arbitrating messages and in event-driven CAN communication (ISO 11898-1:2015).

**Notes:**

- The trigger memory elements should be written when the M\_TTCAN is in INIT state. Write access to the trigger memory elements outside INIT state is not allowed.
- There is an exception for TMIN and TMEX when they are defined as part of a trigger memory element of TYPE Tx\_Ref\_Trigger. In this case they become active at the time mark modified by the actual Reference Trigger Offset (CANFDx\_CHy\_TTOST.RTO).

### 17.4.8 Message RAM OFF

Message RAM can be turned off to save power by setting CANFDx\_CTL.MRAM\_OFF bit. Default value of this bit is '0' and message RAM is retained in this configuration during DeepSleep power mode.

All the M\_TTCAN channels must be powered down before setting CANFDx\_CTL.MRAM\_OFF bit. See [Power Down \(Sleep Mode\) on page 187](#) to power down the M\_TTCAN channels. When message RAM is OFF, any access to message RAM may raise Address Error (MRAM\_SIZE = 0).

After switching the message RAM on again, software needs to allow a certain power-up time before message RAM can be used, that is, before STOP\_REQ can be de-asserted. Check the RAM\_PWR\_DELAY\_CTL register to see the required time for message RAM power up process.

### 17.4.9 RAM Watchdog (RWD)

The RAM watchdog monitors the READY output of the Message RAM. A Message RAM access starts the Message RAM Watchdog Counter with the value configured by CANFDx\_CHy\_RWD.WDC. The counter is reloaded with CANFDx\_CHy\_RWD.WDC when the Message RAM signals successful completion by activating its READY output. In case there is no response from the Message RAM until the counter has counted down to zero, the counter stops and interrupt flag CANFDx\_CHy\_IR.WDI is set. The RAM Watchdog Counter is clocked by the Host clock (CLK\_SYS). Refer to the Registers TRM for more information about the CANFDx\_CHy\_RWD register.

### 17.4.10 Address Error

An address error is detected when either the M\_TTCAN channel or MCU is trying to access an out of range message RAM address (address  $\geq$  MRAM\_SIZE). This feature is added to make software debugging easier. When such an address error is detected the following will happen:

- For writes, the error is not reported back to the master
  - Writes are posted and both the AHB interface and the CAN channels ignore error signaling
- For reads from a M\_TTCAN channel, the error is reported back to the channel; this will result in the following:
  - To prevent corrupt data from being sent, channel will be shutdown (CANFDx\_CHy\_CCCR.INIT=1) immediately
  - An interrupt is raised (BEU)
- For reads from the AHB interface the address error results in a bus error

## 17.5 TTCAN Operation

### 17.5.1 Reference Message

A reference message is a data frame characterized by a specific CAN identifier. It is received and accepted by all nodes except the time master (sender of the reference message).

For Level 1, the data length must be at least one. For Level 0 and Level 2, the data length must be at least four; otherwise, the message is not accepted as a reference message. The reference message may be extended by other data up to the sum of eight CAN data bytes. All bits of the identifier except the three LSBs characterize the message as a reference message. The last three bits specify the priorities of up to eight potential time masters. Reserved bits are transmitted as logical 0 and are ignored by the receivers. The reference message is configured using the CANFDx\_CHy\_TTRMC register.

The time master transmits the reference message. If the reference message is disturbed by an error, it is retransmitted immediately. In a retransmission, the transmitted Master\_Ref\_Mark is updated. The reference message is sent periodically, but it is allowed to stop the periodic transmission (Next\_is\_Gap bit). It can initiate event-synchronized transmission at the start of the next basic cycle by the current time master or by one of the other potential time masters.

The node transmitting the reference message is the current time master. The time master is allowed to transmit other messages. If the current time master fails, its function is replicated by the potential time master with the highest priority. Nodes that are neither time master nor potential time master are time-receiving nodes.

#### 17.5.1.1 Level 1

Level 1 operation is configured via CANFDx\_CHy\_TTOCF.OM = 01 and CANFDx\_CHy\_TTOCF.GEN. External clock synchronization is not available in Level 1.

The information related to the reference message is stored in the first data byte as shown in [Table 17-8](#). Cycle\_Count is optional.

Table 17-8. First Byte of Level 1 Reference Message

| Bits       | 7           | 6        | 5                 | 4 | 3 | 2 | 1 | 0 |
|------------|-------------|----------|-------------------|---|---|---|---|---|
| First Byte | Next_is_Gap | Reserved | Cycle_Count [5:0] |   |   |   |   |   |

#### 17.5.1.2 Level 2

Level 2 operation is configured via CANFDx\_CHy\_TTOCF.OM = 10 and CANFDx\_CHy\_TTOCF.GEN.

The information related to the reference message is stored in the first four data bytes as shown in [Table 17-9](#). Cycle\_Count and the lower four bits of NTU\_Res are optional. The M\_TTCAN does not evaluate NTU\_Res[3:0] from received reference messages, it always transmits these bits as zero.

Table 17-9. First Four Bytes of Level 2 Reference Message

| Bits        | 7                     | 6        | 5                | 4            | 3 | 2 | 1        | 0 |
|-------------|-----------------------|----------|------------------|--------------|---|---|----------|---|
| First Byte  | Next_is_Gap           | Reserved | Cycle_Count[5:0] |              |   |   |          |   |
| Second Byte | NTU_Res[6:4]          |          |                  | NTU_Res[3:0] |   |   | Disc_Bit |   |
| Third Byte  | Master_Ref_Mark[7:0]  |          |                  |              |   |   |          |   |
| Fourth Byte | Master_Ref_Mark[15:8] |          |                  |              |   |   |          |   |

### 17.5.1.3 Level 0

Level 0 operation is configured via CANFDx\_CHy\_TTOCF.OM = 11. External event-synchronized time-triggered operation is not available in Level 0.

The information related to the reference message is stored in the first four data bytes as shown in Table 17-10. In Level 0, Next\_is\_Gap is always zero. Cycle\_Count and the lower four bits of NTU\_Res are optional. The M\_TTCAN does not evaluate NTU\_Res[3:0] from received reference messages; it always transmits these bits as zero.

Table 17-10. First four Bytes of Level 0 Reference Message

| Bits        | 7                     | 6        | 5                | 4            | 3 | 2 | 1        | 0 |
|-------------|-----------------------|----------|------------------|--------------|---|---|----------|---|
| First Byte  | Next_is_Gap           | Reserved | Cycle_Count[5:0] |              |   |   |          |   |
| Second Byte | NTU_Res[6:4]          |          |                  | NTU_Res[3:0] |   |   | Disc_Bit |   |
| Third Byte  | Master_Ref_Mark[7:0]  |          |                  |              |   |   |          |   |
| Fourth Byte | Master_Ref_Mark[15:8] |          |                  |              |   |   |          |   |

## 17.5.2 TTCAN Configuration

### 17.5.2.1 TTCAN Timing

The Network Time Unit (NTU) is the unit in which all times are measured. The NTU is a constant of the whole network and is defined by the network system designer. In TTCAN Level 1 the NTU is the nominal CAN bit time. In TTCAN Level 0 and Level 2 the NTU is a fraction of the physical second.

The NTU is the time base for the local time. The integer part of the local time (16-bit value) is incremented once for each NTU. Cycle time and global time are both derived from local time. The fractional part (3-bit value) of local time, cycle time, and global time is not readable.

In TTCAN Level 0 and Level 2, the length of the NTU is defined by the Time Unit Ratio (TUR). The TUR is a non-integer number given by the formula  $TUR = \text{CANFDx\_CHy\_TURNA.NAV} / \text{CANFDx\_CHy\_TURCF.DC}$ . The length of the NTU is given by the formula  $NTU = \text{CAN Clock Period} \times TUR$ .

The TUR Numerator Configuration NC is an 18-bit number, CANFDx\_CHy\_TURCF.NCL[15:0] can be programmed in the range 0x0000-0xFFFF. CANFDx\_CHy\_TURCF.NCH[17:16] is hard-wired to 0b01. When the number 0xn timer is written to CANFDx\_CHy\_TURCF.NCL[15:0], CANFDx\_CHy\_TURNA.NAV starts with the value  $0x10000 + 0x0n$ . The TUR Denominator Configuration CANFDx\_CHy\_TURCF.DC is a 14-bit number. CANFDx\_CHy\_TURCF.DC may be programmed in the range 0x0001 - 0x3FFF; 0x0000 is an illegal value.

In Level 1, NC must be  $\geq 4 \times \text{CANFDx\_CHy\_TURCF.DC}$ . In Level 0 and Level 2 NC must be  $\geq 8 \times \text{CANFDx\_CHy\_TURCF.DC}$  to allow the 3-bit resolution for the internal fractional part of the NTU.

A hardware reset presets CANFDx\_CHy\_TURCF.DC to 0x1000 and CANFDx\_CHy\_TURCF.NCL to 0x10000, resulting in an NTU consisting of 16 CAN clock periods. Local time and application watchdog are not started before either the CANFDx\_CHy\_CCCR.INIT is reset or CANFDx\_CHy\_TURCF.ELT is set. CANFDx\_CHy\_TURCF.ELT may not be set before the NTU is configured. Setting CANFDx\_CHy\_TURCF.ELT to '1' also locks the write access to register CANFDx\_CHy\_TURCF.

At startup CANFDx\_CHy\_TURNA.NAV is updated from NC (= CANFDx\_CHy\_TURCF.NCL + 0x10000) when CANFDx\_CHy\_TURCF.ELT is set. In TTCAN Level 1 there is no drift compensation. CANFDx\_CHy\_TURNA.NAV does not change during operation, it always equals NC.

In TTCAN Level 0 and Level 2, there are two possibilities for CANFDx\_CHy\_TURNA.NAV to change. When operating as time slave or backup time master, and when CANFDx\_CHy\_TTOCF.ECC is set, CANFDx\_CHy\_TURNA.NAV is updated automatically to the value calculated from the monitored global time speed, as long as the M\_TTCAN is in synchronization states In\_Schedule or In\_Gap. When it loses synchronization, it returns to NC. When operating as the actual time master, and when CANFDx\_CHy\_TTOCF.EECS is set, the host may update CANFDx\_CHy\_TURCF.NCL. When the host sets CANFDx\_CHy\_TTOCN.ECS, CANFDx\_CHy\_TURNA.NAV will be updated from the new value of NC at the next reference message. The status flag CANFDx\_CHy\_TTOST.WECS is set when CANFDx\_CHy\_TTOCN.ECS is set and is cleared when CANFDx\_CHy\_TURNA.NAV is updated. CANFDx\_CHy\_TURCF.NCL is write-locked while CANFDx\_CHy\_TTOST.WECS is set.

In TTCAN Level 0 and Level 2, the clock calibration process adapts CANFDx\_CHy\_TURNA.NAV in the range of the synchronization deviation limit (SDL) of  $NC \pm 2$  (CANFDx\_CHy\_TTOCF.LDSDL + 5). CANFDx\_CHy\_TURCF.NCL should be programmed to the largest applicable numerical value to achieve the best accuracy in the calculation of CANFDx\_CHy\_TURNA.NAV.

The synchronization deviation (SD) is the difference between NC and CANFDx\_CHy\_TURNA.NAV ( $SD = |NC - CANFDx\_CHy\_TURNA.NAV|$ ). It is limited by the SDL, which is configured by its dual logarithm CANFDx\_CHy\_TTOCF.LDSDL ( $SDL = 2$  (CANFDx\_CHy\_TTOCF.LDSDL + 5)) and should not exceed the clock tolerance given by the CAN bit timing configuration. SD is calculated at each new basic cycle. When the calculated CANFDx\_CHy\_TURNA.NAV deviates by more than SDL from NC, or if the Disc\_Bit in the reference message is set, the drift compensation is suspended, CANFDx\_CHy\_TTIR.GTE is set, and CANFDx\_CHy\_TTOSC.QCS is reset; if Disc\_Bit = '1', CANFDx\_CHy\_TTIR.GTD is set.

TUR configuration examples are shown in [Table 17-11](#).

Table 17-11. TUR Configuration Examples

| TUR                 | 8       | 10      | 24      | 50       | 510     | 125000  | 32.5    | 100/12  | 529/17  |
|---------------------|---------|---------|---------|----------|---------|---------|---------|---------|---------|
| NC                  | 0x1FFF8 | 0x1FFFE | 0x1FFF8 | 0x1FFFEA | 0x1FFFE | 0x1FFE0 | 0x1FFE0 | 0x19000 | 0x10880 |
| CANFDx_CHy_TURCF.DC | 0x3FFF  | 0x3333  | 0x1555  | 0x0A3D   | 0x0101  | 0x0001  | 0x0FC0  | 0x3000  | 0x0880  |

CANFDx\_CHy\_TTOCN.ECS schedules NC for activation by the next reference message. CANFDx\_CHy\_TTOCN.SGT schedules CANFDx\_CHy\_TTGTP.TP for activation by the next reference message. Setting of CANFDx\_CHy\_TTOCN.ECS and CANFDx\_CHy\_TTOCN.SGT requires CANFDx\_CHy\_TTOCF.EECS to be set (external clock synchronization enabled) while the M\_TTCAN is actual time master.

The M\_TTCAN module provides an application watchdog to verify the function of the application program. The host has to serve this watchdog regularly; otherwise, all CAN bus activity is stopped. The Application Watchdog Limit CANFDx\_CHy\_TTOCF.AWL specifies the number of NTUs the watchdog has to be served. The maximum number of NTUs is 256. The Application Watchdog is served by reading register CANFDx\_CHy\_TTOST. CANFDx\_CHy\_TTOST.AWE indicates whether the watchdog is served in time. In case the application failed to serve the application watchdog, interrupt flag CANFDx\_CHy\_TTIR.AW is set. For software development, the application watchdog may be disabled by programming CANFDx\_CHy\_TTOCF.AWL to 0x00 (see [17.3.1.10 Application Watchdog](#)).

### 17.5.2.2 Message Scheduling

CANFDx\_CHy\_TTOCF.TM controls whether the M\_TTCAN operates as a potential time master or as a time slave. If it is a potential time master, the three LSBs of the reference message identifier CANFDx\_CHy\_TTRMC.RID define the master priority, 0 being the highest and 7 the lowest priority. Two nodes in the network may not use the same master priority. CANFDx\_CHy\_TTRMC.RID is used for recognition of reference messages. CANFDx\_CHy\_TTRMC.RMPS is not relevant for time slaves.

The Initial Reference Trigger Offset CANFDx\_CHy\_TTOCF.IRTO is a 7-bit-value that defines (in NTUs) how long a backup time master waits before it starts the transmission of a reference message, when a reference message is expected but the bus remains idle. The recommended value for CANFDx\_CHy\_TTOCF.IRTO is the master priority multiplied with a factor depending on the expected clock drift between the potential time masters in the network. The sequential order of the backup time masters, when one of them starts the reference message if the current time master fails, should correspond to their master priority, even with maximum clock drift.

CANFDx\_CHy\_TTOCF.OM decides whether the node operates in TTCAN Level 0, Level 1, or Level 2. In one network, all potential time masters should operate on the same level. Time slaves may operate on Level 1 in a Level 2 network, but not vice versa. The configuration of the TTCAN operation mode via CANFDx\_CHy\_TTOCF.OM is the last step in the setup. When CANFDx\_CHy\_TTOCF.OM = 00 (event-driven CAN communication), the M\_TTCAN operates according to ISO 11898-1:2015, without time triggers. When CANFDx\_CHy\_TTOCF.OM = 01 (Level 1), the M\_TTCAN operates according to ISO 11898-4, but without the possibility to synchronize the basic cycles to external events, the Next\_is\_Gap bit in the reference message is ignored. When CANFDx\_CHy\_TTOCF.OM = 10 (Level 2), the M\_TTCAN operates according to ISO 11898-4, including the event-synchronized start of a basic cycle. When CANFDx\_CHy\_TTOCF.OM = 11 (Level 0), the M\_TTCAN operates as event-driven CAN but maintains a calibrated global time base similar to Level 2.

CANFDx\_CHy\_TTOCF.EECS enables the external clock synchronization, allowing the application program of the current time master to update the TUR configuration during time-triggered operation, to adapt the clock speed and (in Level 0,2 only) the global clock phase to an external reference.

CANFDx\_CHy\_TTMLM.ENTT in the TT Matrix Limits register specifies the number of expected Tx Triggers in the system matrix. This is the sum of Tx Triggers for exclusive single arbitrating and merged arbitrating windows, excluding the Tx\_Ref\_Triggers. Note that this is usually not the number of Tx\_Trigger memory elements; the number of basic cycles in the system matrix and the trigger's repeat factors must be taken into account. An inaccurate configuration of CANFDx\_CHy\_TTMLM.ENTT will result in either a TX Count Underflow (CANFDx\_CHy\_TTIR.TXU = 1 and CANFDx\_CHy\_TTOST.EL = 01, severity 1) or in a TX Count Overflow (CANFDx\_CHy\_TTIR.TXO = 1 and CANFDx\_CHy\_TTOST.EL = 10, severity 2).

**Note:** In case the first reference message seen by a node does not have Cycle\_Count zero, this node may finish its first matrix cycle with its TX count resulting in a TX Count Underflow condition. As long as a node is in state, synchronizing its Tx Triggers will not lead to transmissions.

CANFDx\_CHy\_TTMLM.CCM specifies the number of the last basic cycle in the system matrix. The counting of basic cycles starts at 0. In a system matrix consisting of eight basic cycles CANFDx\_CHy\_TTMLM.CCM would be 7. CANFDx\_CHy\_TTMLM.CCM is ignored by time slaves, a receiver of a reference message considers the received cycle count as the valid cycle count for the actual basic cycle.

CANFDx\_CHy\_TTMLM.TXEW specifies the length of the TX enable window in NTUs. The TX enable window is the period at the beginning of a time window where a transmission may be started. If the sample point of the first bit of a transmit message is not inside the TX enable window, the transmission cannot be started in that time window at all. An example is because of an overlap from the previous time window's message. CANFDx\_CHy\_TTMLM.TXEW should be chosen based on the network's synchronization quality and the relation between the length of the time windows and the length of messages.

### 17.5.2.3 Trigger Memory

The trigger memory is part of the message RAM. It stores up to 64 trigger elements. A trigger memory element consists of Time Mark TM, Cycle Code CC, Trigger Type TYPE, Filter Type FTYPE, Message Number MNR, Message Status Count MSC, Time Mark Event Internal TMIN, Time Mark Event External TMEX, and Asynchronous Serial Communication ASC (see [17.4.7 Trigger Memory Element](#)).

The time mark defines at which cycle time a trigger becomes active. The trigger elements in the trigger memory must be sorted by their time marks. The trigger element with the lowest time mark is written to the first trigger memory word. Message number and cycle code are ignored for triggers of type Tx\_Ref\_Trigger, Tx\_Ref\_Trigger\_Gap, Watch\_Trigger, Watch\_Trigger\_Gap, and End\_of\_List.

When the cycle time reaches the time mark of the actual trigger, the FSE switches to the next trigger and starts to read it from the trigger memory. For a transmit trigger, the TX handler starts to read the message from the message RAM as soon as the FSE switches to its trigger. The RAM access speed defines the minimum time step between a transmit trigger and its preceding trigger, the TX handler should be able to prepare the transmission before the transmit trigger's time mark is reached. The RAM access speed also limits the number of non-matching (with regard to their cycle code) triggers between two matching triggers, the next matching trigger must be read before its time mark is reached. If the reference message is n NTU long, a trigger with a time mark less than n will never become active and will be treated as a configuration error.

The starting point of cycle time is the sample point of the reference message's start-of-frame bit. The next reference message is requested when cycle time reaches the Tx\_Ref\_Trigger's time mark. The M\_TTCAN reacts to the transmission request at the next sample point. A new Sync\_Mark is captured at the start-of-frame bit, but the cycle time is incremented until the reference message is successfully transmitted (or received) and the Sync\_Mark is taken as the new Ref\_Mark. At that point, cycle time is restarted. As a consequence, cycle time can never (with the exception of initialization) be seen at a value less than n, with n being the length of the reference message measured in NTU.

Length of a basic cycle: Tx\_Ref\_Trigger time mark + 1 NTU + 1 CAN bit time

The trigger list will be different for all nodes in the TTCAN network. Each node knows only the Tx Triggers for its own transmit messages, the Rx Triggers for the receive messages that are processed by this node, and the triggers concerning the reference messages.

## Trigger Types

Tx\_Ref\_Trigger (TYPE = 0000) and Tx\_Ref\_Trigger\_Gap (TYPE = 0001) cause the transmission of a reference message by a time master. A configuration error (CANFDx\_CHy\_TTOST.EL = 11, severity 3) is detected when a time slave encounters a Tx\_Ref\_Trigger(\_Gap) in its trigger memory. Tx\_Ref\_Trigger\_Gap is only used in external event-synchronized time-triggered operation mode. In that mode, Tx\_Ref\_Trigger is ignored when the M\_TTCAN synchronization state is In\_Gap (CANFDx\_CHy\_TTOST.SYS = 10).

Tx\_Trigger\_Single (TYPE = 0010), Tx\_Continuous (TYPE = 0011), Tx\_Trigger\_Arbitration (TYPE = 0100), and Tx\_Trigger\_Merged (TYPE = 0101) cause the start of a transmission. They define the start of a time window.

Tx\_Trigger\_Single starts a single transmission in an exclusive time window when the message buffer's Transmission Request Pending bit is set. After successful transmission, the Transmission Request Pending bit is reset.

Tx\_Trigger\_Continuous starts a transmission in an exclusive time window when the message buffer's transmission Request Pending bit is set. After successful transmission, the Transmission Request Pending bit remains set, and the message buffer is transmitted again in the next matching time window.

Tx\_Trigger\_Arbitration starts an arbitrating time window, Tx\_Trigger\_Merged a merged arbitrating time window. The last Tx\_Trigger of a merged arbitrating time window must be of type Tx\_Trigger\_Arbitration. A Configuration Error (CANFDx\_CHy\_TTOST.EL = 11, severity 3) is detected when a trigger of type Tx\_Trigger\_Merged is followed by any other Tx\_Trigger than one of type Tx\_Trigger\_Merged or Tx\_Trigger\_Arbitration. Several Tx\_Triggers may be defined for the same TX message buffer. Depending on their cycle code, they may be ignored in some basic cycles. The cycle code should be considered when the expected number of Tx\_Triggers (CANFDx\_CHy\_TTMLM.ENTT) is calculated.

Watch\_Trigger (TYPE = 0110) and Watch\_Trigger\_Gap (TYPE = 0111) check for missing reference messages. They are used by both time masters and time slaves. Watch\_Trigger\_Gap is only used in external event-synchronized time-triggered operation mode. In that mode, a Watch\_Trigger is ignored when the M\_TTCAN synchronization state is In\_Gap (CANFDx\_CHy\_TTOST.SYS = 10).

Rx\_Trigger (TYPE = 1000) is used to check for the reception of periodic messages in exclusive time windows. Rx\_Triggers are not active until state In\_Schedule or In\_Gap is reached. The time mark of an Rx\_Trigger should be placed after the end of that message transmission, independent of time window boundaries. Depending on their cycle code, Rx\_Triggers may be ignored in some basic cycles. At the Rx\_Trigger time mark, it is checked whether the last received message before this time mark and after start of cycle or previous Rx\_Trigger matches the acceptance filter element referenced by MNR. Accepted messages are stored in one of two receive FIFOs, according to the acceptance filtering, independent of the Rx\_Trigger. Acceptance filter elements that are referenced by Rx\_Triggers should be placed at the beginning of the filter list to ensure that the filtering is finished before the Rx\_Trigger time mark is reached.

Time\_Base\_Trigger (TYPE = 1001) is used to generate internal/external events depending on the configuration of ASC, TMIN, and TMEX.

End\_of\_List (TYPE = 1010...1111) is an illegal trigger type, a configuration error (CANFDx\_CHy\_TTOST.EL = 11, severity 3) is detected when an End\_of\_List trigger is encountered in the trigger memory before the Watch\_Trigger or Watch\_Trigger\_Gap.

## Restrictions for the Node's Trigger List

Two triggers may not be active at the same cycle time and cycle count, but triggers that are active in different basic cycles (different cycle code) may share the same time mark.

Rx\_Triggers and Time\_Base\_Triggers may not be placed inside the TX enable windows of Tx\_Trigger\_Single/Continuous/Arbitration, but they may be placed after Tx\_Trigger\_Merged.

Triggers that are placed after the Watch\_Trigger (or the Watch\_Trigger\_Gap when CANFDx\_CHy\_TTOST.SYS = 10) will never become active. The watch triggers themselves will not become active when the reference messages are transmitted on time.

All unused trigger memory words (after the Watch\_Trigger or after the Watch\_Trigger\_Gap when CANFDx\_CHy\_TTOST.SYS = 10) must be set to trigger type End\_of\_List.

A typical trigger list for a potential time master will begin with a number of Tx\_Triggers and Rx\_Triggers followed by the Tx\_Ref\_Trigger and Watch\_Trigger. For networks with external event-synchronized time-triggered communication, this is followed by the Tx\_Ref\_Trigger\_Gap and the Watch\_Trigger\_Gap. The trigger list for a time slave will be the same but without the Tx\_Ref\_Trigger and the Tx\_Ref\_Trigger\_Gap.

At the beginning of each basic cycle, that is at each reception or transmission of a reference message, the trigger list is processed starting with the first trigger memory element. The FSE looks for the first trigger with a cycle code that matches the current cycle count. The FSE waits until cycle time reaches the trigger's time mark and activates the trigger. Later, the FSE looks for the next trigger in the list with a cycle code that matches the current cycle count.

Special consideration is needed for the time around Tx\_Ref\_Trigger and Tx\_Ref\_Trigger\_Gap. In a time master competing for master ship, the effective time mark of a Tx\_Ref\_Trigger may be decremented to be the first node to start a reference message. In backup time masters the effective time mark of a Tx\_Ref\_Trigger or Tx\_Ref\_Trigger\_Gap is the sum of its configured time mark and the Reference Trigger Offset CANFDx\_CHy\_TTOCF.IRTO. If error level 2 is reached (CANFDx\_CHy\_TTOST.EL = 10), the effective time mark is the sum of its time mark and 0x127. No other trigger elements should be placed in this range; otherwise, the time marks may appear out of order and are flagged as a configuration error. Trigger elements that are coming after Tx\_Ref\_Trigger may never become active as long as the reference messages come in time.

There are interdependencies between the following parameters:

- Host clock frequency
- Speed and waiting time for Trigger RAM accesses
- Length of the acceptance filter list
- Number of trigger elements
- Complexity of cycle code filtering in the trigger elements
- Offset between time marks of the trigger elements

**Examples of Trigger Handling**

The following example shows how the trigger list is derived from a node's system matrix. Assume that node A is a first time master; a section of the system matrix shown in [Table 17-12](#).

Table 17-12. System Matrix Node A

| Cycle Count | Time Mark1 | Time Mark2 | Time Mark3 | Time Mark4 | Time Mark5 | Time Mark6 | Time Mark7 |
|-------------|------------|------------|------------|------------|------------|------------|------------|
| 0           | Tx7        |            |            |            |            | TxRef      | Error      |
| 1           | Rx3        |            | Tx2, Tx4   |            |            | TxRef      | Error      |
| 2           |            |            |            |            |            | TxRef      | Error      |
| 3           | Tx7        |            | Rx5        |            |            | TxRef      | Error      |
| 4           | Tx7        |            |            | Rx6        |            | TxRef      | Error      |

The cycle count starts with 0 – 0, 1, 3, 7, 15, 31, 63 (the number of basic cycles in the system matrix is 1, 2, 4, 8, 16, 32, 64). The maximum cycle count is configured by CANFDx\_CHy\_TTMLM.CCM. The Cycle Code (CC) is composed of repeat factor (value of most significant '1') and the number of the first basic cycle in the system matrix (bit field after most significant '1').

Example: When CC is 0b0010011 (repeat factor: 16, first basic cycle: 3) and maximum cycle count of CANFDx\_CHy\_TTMLM.CCM = 0x3F, matches occur at cycle counts 3, 19, 35, 51.

A trigger element consists of Time Mark (TM), Cycle Code (CC), Trigger Type (TYPE), and Message Number (MNR). For transmission, MNR references the TX buffer number (0..31). For reception, MNR references the number of the filter element (0..127) that matched during acceptance filtering. Depending on the configuration of the Filter Type FTYPE, the 11-bit or 29-bit message ID filter list is referenced.

In addition, a trigger element can be configured for Asynchronous Serial Communication (ASC), generation of Time Mark Event Internal (TMIN), and Time Mark Event External (TMEX). The Message Status Count (MSC) holds the counter value (0..7) for scheduling errors for periodic messages in exclusive time windows when the time mark of the trigger element becomes active.

Table 17-13. Trigger List Node A

| Trigger | Time Mark TM[15:0] | Cycle Code CC[6:0] | Trigger Type TYPE[3:0] | Mess. No. MNR[6:0] |
|---------|--------------------|--------------------|------------------------|--------------------|
| 0       | Mark1              | 0b0000100          | Tx_Trigger_Single      | 7                  |
| 1       | Mark1              | 0b1000000          | Rx_Trigger             | 3                  |
| 2       | Mark1              | 0b1000011          | Tx_Trigger_Single      | 7                  |
| 3       | Mark3              | 0b1000001          | Tx_Trigger_Merged      | 2                  |
| 4       | Mark3              | 0b1000011          | Rx_Trigger             | 5                  |
| 5       | Mark4              | 0b1000001          | Tx_Trigger_Arbitration | 4                  |
| 6       | Mark4              | 0b1000100          | Rx_Trigger             | 6                  |
| 7       | Mark6              | n.a.               | Tx_Ref_Trigger         | 0 (Ref)            |
| 8       | Mark7              | n.a.               | Watch_Trigger          | n.a.               |
| 9       | n.a.               | n.a.               | End_of_List            | n.a.               |

Tx\_Trigger\_Single, Tx\_Trigger\_Continuous, Tx\_Trigger\_Merged, Tx\_Trigger\_Arbitration, Rx\_Trigger, and Time\_Base\_Trigger are only valid for the specified cycle code. For all other trigger types the cycle code is ignored.

The FSE starts the basic cycle by scanning the trigger list starting from zero until a trigger with time mark that is greater than the cycle time is reached, CC matches the actual cycle count, or a trigger of type Tx\_Ref\_Trigger, Tx\_Ref\_Trigger\_Gap, Watch\_Trigger, or Watch\_Trigger\_Gap is encountered.

When the cycle time reaches TM, the action defined by TYPE and MNR is started. There is an error in the configuration when it reaches End\_of\_List.

At Mark6, the reference message (always TxRef) is transmitted. After transmission, the FSE returns to the beginning of the trigger list. When it reaches Watch Trigger at Mark7, the node is unable to transmit the reference message; error treatment is then started.

### Detection of Configuration Errors

A configuration error is signaled via CANFDx\_CHy\_TTOST.EL = 11 (severity 3) when:

- The FSE comes to a trigger in the list with a cycle code that matches the current cycle count but with a time mark that is less than the cycle time.
- The previous active trigger was a Tx\_Trigger\_Merged and the FSE comes to a trigger in the list with a cycle code that matches the current cycle count but that is neither a Tx\_Trigger\_Merged nor a Tx\_Trigger\_Arbitration nor a Time\_Base\_Trigger nor an Rx\_Trigger.
- The FSE of a node with CANFDx\_CHy\_TTOCF.TM = 0 (time slave) encounters a Tx\_Ref\_Trigger or a Tx\_Ref\_Trigger\_Gap.
- Any time mark placed inside the TX enable window (defined by CANFDx\_CHy\_TTMLM.TXEW) of a Tx\_Trigger with a matching cycle code.
- A time mark is placed near the time mark of a Tx\_Ref\_Trigger and the Reference Trigger Offset CANFDx\_CHy\_TTOST.RTO causes a reversal of their sequential order measured in cycle time.

### 17.5.2.4 TTCAN Schedule Initialization

The synchronization to the M\_TTCAN message schedule starts when CANFDx\_CHy\_CCCR.INIT is reset. The M\_TTCAN can operate time-triggered (CANFDx\_CHy\_TTOCF.GEN = 0) or external event-synchronized time-triggered (CANFDx\_CHy\_TTOCF.GEN = 1). All nodes start with cycle time zero at the beginning of their trigger list with CANFDx\_CHy\_TTOST.SYS = 00 (out of synchronization); no transmission is enabled with the exception of the reference message. Nodes in external event-synchronized time-triggered operation mode will ignore Tx\_Ref\_Trigger and Watch\_Trigger and use Tx\_Ref\_Trigger\_Gap and Watch\_Trigger\_Gap instead until the first reference message decides whether a gap is active.

#### Time Slaves

After configuration, a time slave will ignore its Watch\_Trigger and Watch\_Trigger\_Gap when it does not receive any message before reaching the Watch\_Triggers. When it reaches Init\_Watch\_Trigger, interrupt flag CANFDx\_CHy\_TTIR.IWT is set, the FSE is frozen, and the cycle time will become invalid. However, the node will still be able to take part in CAN bus communication (to give acknowledge or to send error flags). The first received reference message will restart the FSE and the cycle time.

**Note:** Init\_Watch\_Trigger is not part of the trigger list. It is implemented as an internal counter that counts up to 0xFFFF = maximum cycle time.

When a time slave receives any message but the reference message before reaching the Watch\_Triggers, it will assume a fatal error (CANFDx\_CHy\_TTOST.EL = 11, severity 3), set interrupt flag CANFDx\_CHy\_TTIR.WT, switch off its CAN bus output, and enter the bus monitoring mode (CANFDx\_CHy\_CCCR.MON set to '1'). In the bus monitoring mode, it is still able to receive messages, but cannot send any dominant bits and therefore, cannot acknowledge.

**Note:** To leave the fatal error state, the host must set CANFDx\_CHy\_CCCR.INIT = '1'. After reset of CANFDx\_CHy\_CCCR.INIT, the node restarts TTCAN communication.

When no error is encountered during synchronization, the first reference message sets CANFDx\_CHy\_TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next\_is\_Gap bit) to CANFDx\_CHy\_TTOST.SYS = 11 (In\_Schedule) or CANFDx\_CHy\_TTOST.SYS = 10 (In\_Gap), enabling all Tx\_Triggers and Rx\_Triggers.

#### Potential Time Master

After configuration, a potential time master will start the transmission of a reference message when it reaches its Tx\_Ref\_Trigger (or its Tx\_Ref\_Trigger\_Gap when in external event-synchronized time-triggered operation). It will ignore its Watch\_Trigger and Watch\_Trigger\_Gap when it does not receive any message or transmit the reference message successfully before reaching the Watch\_Triggers (the reason assumed is that all other nodes still in reset or configuration and does not acknowledge). When it reaches Init\_Watch\_Trigger, the attempted transmission is aborted, interrupt flag CANFDx\_CHy\_TTIR.IWT is set, the FSE is frozen, and the cycle time will become invalid, but the node will still be able to take part in CAN bus communication (to acknowledge or send error flags). Resetting CANFDx\_CHy\_TTIR.IWT will re-enable the transmission of reference messages until the next time Init\_Watch\_Trigger condition is met, or another CAN message is received. The FSE will be restarted by the reception of a reference message.

When a potential time master reaches the Watch\_Triggers after it has received any message but the reference message, it will assume a fatal error (CANFDx\_CHy\_TTOST.EL = 11, severity 3), set interrupt flag CANFDx\_CHy\_TTIR.WT, switch off its CAN bus output, and enter the bus monitoring mode (CANFDx\_CHy\_CCCR.MON set to '1'). In bus monitoring mode, it is still able to receive messages, but it cannot send any dominant bits and therefore, cannot acknowledge.

When no error is detected during initialization, the first reference message sets CANFDx\_CHy\_TTOST.SYS = 01 (synchronizing), the second sets the TTCAN synchronization state (depending on its Next\_is\_Gap bit) to CANFDx\_CHy\_TTOST.SYS = 11 (In\_Schedule) or CANFDx\_CHy\_TTOST.SYS = 10 (In\_Gap), enabling all Tx\_Triggers and Rx\_Triggers.

A potential time master is current time master (CANFDx\_CHy\_TTOST.MS = 11) when it is the transmitter of the last reference message; otherwise, it is the backup time master (CANFDx\_CHy\_TTOST.MS = 10).

When all potential time masters have finished configuration, the node with the highest time master priority in the network will become the current time master.

### 17.5.3 TTCAN Gap Control

All functions related to gap control apply only when the M\_TTCAN is operated in external event-synchronized time-triggered mode (CANFDx\_CHy\_TTOCF.GEN = 1). In this operation mode the TTCAN message schedule may be interrupted by inserting gaps between the basic cycles of the system matrix. All nodes connected to the CAN network should be configured for external event-synchronized time-triggered operation.

During a gap, all transmissions are stopped and the CAN bus remains idle. A gap is finished when the next reference message starts a new basic cycle. The gap starts at the end of a basic cycle that was started by a reference message with bit Next\_is\_Gap = '1'; for example, gaps are initiated by the current time master.

The current time master has two options to initiate a gap. A gap can be initiated under software control when the application program writes CANFDx\_CHy\_TTOCN.NIG = 1. The Next\_is\_Gap bit will be transmitted as '1' with the next reference message. A gap can also be initiated under hardware control when the application program writes CANFDx\_CHy\_TTOCN.GCS = 1. When a reference message is started and CANFDx\_CHy\_TTOCN.GCS is set, Next\_is\_Gap = '1' will be set.

As soon as that reference message is completed, the CANFDx\_CHy\_TTOST.WFE bit will announce the gap to the time master and slaves. The current basic cycle will continue until its last time window. The time after the last time window is the gap time.

For the actual time master and the potential time masters, CANFDx\_CHy\_TTOST.GSI will be set when the last basic cycle has finished and the gap time starts. In nodes that are time slaves, the CANFDx\_CHy\_TTOST.GSI bit will remain at '0'.

When a potential time master is in synchronization state In\_Gap (CANFDx\_CHy\_TTOST.SYS = 10), it has four options to intentionally finish a gap:

- Under software control by writing CANFDx\_CHy\_TTOCN.FGP = 1.
- Under hardware control (CANFDx\_CHy\_TTOCN.GCS = 1), CANFDx\_CHy\_TTOCN.FGP will automatically be set when an edge from HIGH to LOW at the internal event trigger input pin is detected and restarts the schedule.
- The third option is a time-triggered restart. When CANFDx\_CHy\_TTOCN.TMG = 1, the next register time mark interrupt (CANFDx\_CHy\_TTIR.RTMI = 1) will set CANFDx\_CHy\_TTOCN.FGP and start the reference message.
- Any potential time master will finish a gap when it reaches its Tx\_Ref\_Trigger\_Gap, assuming that the event to synchronize to did not occur on time.

None of these options can cause a basic cycle to be interrupted with a reference message.

Setting CANFDx\_CHy\_TTOCN.FGP after the gap time has started will start the transmission of a reference message immediately and will thereby synchronize the message schedule. When CANFDx\_CHy\_TTOCN.FGP is set before the gap time has started (while the basic cycle is still in progress), the next reference message is started at the end of the basic cycle, at the Tx\_Ref\_Trigger – there will be no gap time in the message schedule.

In time-triggered operation, bit Next\_is\_Gap = '1' in the reference message will be ignored, and the bits CANFDx\_CHy\_TTOCN.NIG, CANFDx\_CHy\_TTOCN.FGP, and CANFDx\_CHy\_TTOCN.TMG will be considered.

### 17.5.4 Stop Watch

The stop watch function enables capturing of M\_TTCAN internal time values (local time, cycle time, or global time) triggered by an external event.

To enable the stop watch function, the application program must first define local time, cycle time, or global time as stop watch source via CANFDx\_CHy\_TTOCN.SWS. When CANFDx\_CHy\_TTOCN.SWS is not equal to '00' and TT Interrupt Register flag CANFDx\_CHy\_TTIR.SWE is '0', the actual value of the time selected by CANFDx\_CHy\_TTOCN.SWS will be copied into CANFDx\_CHy\_TTCPT.SWV on the next rising/falling edge (as configured via CANFDx\_CHy\_TTOCN.SWP) on stop watch trigger. This will set interrupt flag CANFDx\_CHy\_TTIR.SWE. After the application program has read CANFDx\_CHy\_TTCPT.SWV, it may enable the next stop watch event by resetting CANFDx\_CHy\_TTIR.SWE to '0'.

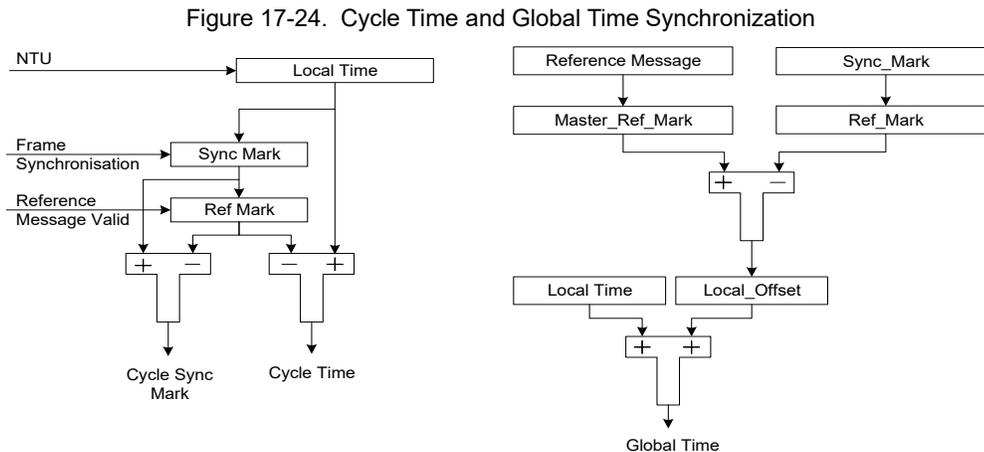
### 17.5.5 Local Time, Cycle Time, Global Time, and External Clock Synchronization

There are two possible levels in time-triggered CAN: Level 1 and Level 2. Level 1 provides only time-triggered operation using cycle time. Level 2 additionally provides increased synchronization quality, global time, and external clock synchronization. In both levels, all timing features are based on a local time base – the local time.

The local time is a 16-bit cyclic counter, it is incremented once each NTU. Internally the NTU is represented by a 3-bit counter, which can be regarded as a fractional part (three binary digits) of the local time. Generally, the 3-bit NTU counter is incremented eight times each NTU. If the length of the NTU is shorter than eight CAN clock periods (as may be configured in Level 1, or as a result of clock calibration in Level 2), the length of the NTU fraction is adapted, and the NTU counter is incremented only four times each NTU.

Figure 17-24 describes the synchronization of the cycle time and global time, performed in the same manner by all TTCAN nodes, including the time master. Any message received or transmitted invokes a capture of the local time taken at the message's frame synchronization event. This frame synchronization event occurs at the sample point of each Start-of-Frame (SoF) bit and causes the local time to be stored as Sync\_Mark. Sync\_Marks and Ref\_Marks are captured including the 3-bit fractional part.

Whenever a valid reference message is transmitted or received, the internal Ref\_Mark is updated from the Sync\_Mark. The difference between Ref\_Mark and Sync\_Mark is the Cycle Sync Mark (Cycle Sync Mark = Sync\_Mark – Ref\_Mark) stored in register CANFDx\_CHy\_TTCSM. The most significant 16 bits of the difference between Ref\_Mark and the actual value of the local time is the cycle time (Cycle Time = Local Time – Ref\_Mark).



The cycle time that can be read from CANFDx\_CHy\_TTCTC.CT is the difference of the node's local time and Ref\_Mark, both synchronized into the host clock domain and truncated to 16 bits.

The global time exists for TTCAN Level 0 and Level 2 only, in Level 1 it is invalid. The node's view of the global time is the local image of the global time in (local) NTUs. After configuration, a potential time master will use its own local time as global time. This is done by transmitting its own Ref\_Marks as Master\_Ref\_Marks in the reference message (bytes 3 and 4). The global time that can be read from CANFDx\_CHy\_TTLGT.GT is the sum of the node's local time and its local offset, both synchronized into the host clock domain and truncated to 16 bit. The fractional part is used for clock synchronization only.

A node that receives a reference message calculates its local offset to the global time by comparing its local Ref\_Mark with the received Master\_Ref\_Mark (see Figure 17-24). The node's view of the global time is local time + local offset. In a potential time master that has never received another time master's reference message, Local\_Offset will be zero. When a node becomes the current time master after having received other reference messages first, Local\_Offset will be frozen at its last value. In the time receiving nodes, Local\_Offset may be subject to small adjustments, due to clock drift, when another node becomes time master, or when there is a global time discontinuity, signaled by Disc\_Bit in the reference message. With the exception of global time discontinuity, the global time provided to the application program by register CANFDx\_CHy\_TTLGT is smoothed by a low-pass filtering to have a continuous monotonic value.

Figure 17-25. TTCAN Level 0 and Level 2 Drift Compensation

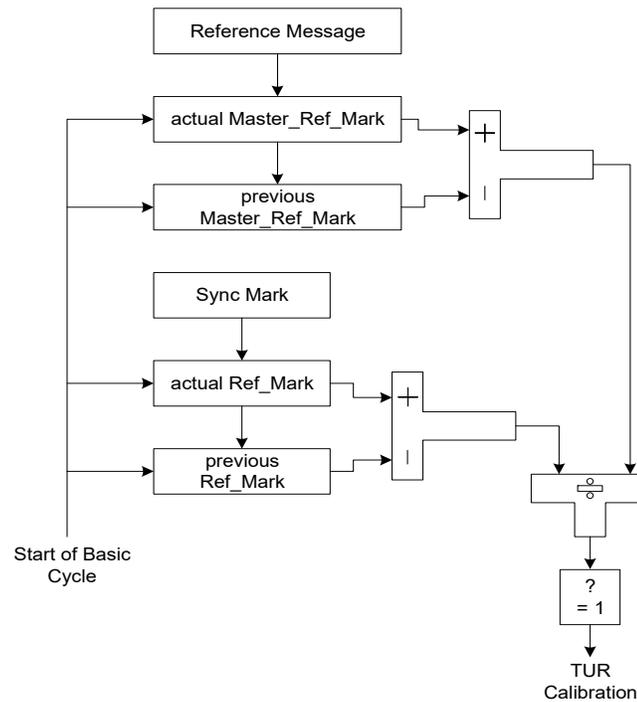


Figure 17-25 illustrates how in TTCAN Level 0 and Level 2 the receiving node compensates the drift between its own local clock and the time master's clock by comparing the length of a basic cycle in local time and in global time. If there is a difference between the two values, and the Disc\_Bit in the reference message is not set, a new value for CANFDx\_CHy\_TURNA.NAV is calculated. If the synchronization deviation (SD) =  $|NC - \text{CANFDx\_CHy\_TURNA.NAV}| \leq \text{SDL}$ , the new value for CANFDx\_CHy\_TURNA.NAV takes effect. Otherwise, the automatic drift compensation is suspended.

In TTCAN Level 0 and Level 2, CANFDx\_CHy\_TTOST.QCS indicates whether the automatic drift compensation is active or suspended. In TTCAN Level 1, CANFDx\_CHy\_TTOST.QCS is always '1'.

The current time master may synchronize its local clock speed and the global time phase to an external clock source. This is enabled by bit CANFDx\_CHy\_TTOCF.EECS.

The stop watch function (see [Stop Watch on page 226](#)) may be used to measure the difference in clock speed between the local clock and the external clock. The local clock speed is adjusted by first writing the newly calculated Numerator Configuration Low to CANFDx\_CHy\_TURCF.NCL (CANFDx\_CHy\_TURCF.DC cannot be updated during operation). The new value takes effect by writing CANFDx\_CHy\_TTOCN.ECS to '1'.

The global time phase is adjusted by first writing the phase offset into the TT Global Time Preset register (CANFDx\_CHy\_TTGTP). The new value takes effect by writing CANFDx\_CHy\_TTOCN.SGT to '1'. The first reference message transmitted after the global time phase adjustment will have the Disc\_Bit set to '1'.

CANFDx\_CHy\_TTOST.QGTP shows whether the node's global time is in phase with the time master's global time. CANFDx\_CHy\_TTOST.QGTP is permanently '0' in TTCAN Level 1 and when the SDL is exceeded in TTCAN Level 0,2 (CANFDx\_CHy\_TTOST.QCS = 0). It is temporarily '0' while the global time is low-pass filtered to supply the application with a continuous monotonic value. There is no low-pass filtering when the last reference message contains a Disc\_Bit = '1' or when CANFDx\_CHy\_TTOST.QCS = 0.

## 17.5.6 Synchronization Triggers

One of the benefits of TTCAN is that it can make communication latency deterministic. To maintain this property across multiple CAN networks (or a Flexray network) these networks must be synchronized. M\_TTCAN includes several trigger inputs and outputs to enable this synchronization.

Each M\_TTCAN channel has trigger input and trigger output connected to trigger multiplexer. Using trigger functionality each channel has the possibility to not just synchronize with any other M\_TTCAN channel, but also to other working network (such as the Flexray network). For more information refer to the [Trigger Multiplexer Block chapter on page 113](#).

Stop watch and Event trigger inputs for the M\_TTCAN channel are connected through the CAN\_TT\_TR\_IN signal coming from the trigger multiplexer. Output trigger from the channel such as Time Mark Trigger and Register Time Mark triggers are connected through CAN\_TT\_TR\_OUT to the trigger multiplexer.

Using this infrastructure, synchronously running networks are achievable.

## 17.5.7 TTCAN Error Level

The ISO 11898-4 specifies four levels of error severity:

- S0 - No Error
- S1 - Warning  
Only notification of application, reaction application-specific.
- S2 Error  
Notification of application. All transmissions in exclusive or arbitrating time windows are disabled (that is, no data or remote frames may be started). Potential time masters still transmit reference messages with the Reference Trigger Offset CANFDx\_CHy\_TTOST.RTO set to the maximum value of 127.
- S3 - Severe Error  
Notification of application. All CAN bus operations are stopped; that is, transmission of dominant bits is not allowed and CANFDx\_CHy\_CCCR.MON is set. The S3 error condition remains active until the application updates the configuration (sets CANFDx\_CHy\_CCCR.CCE).

If several errors are detected at the same time, the highest severity prevails. When an error is detected, the application is notified by CANFDx\_CHy\_TTIR.ELC. The error level is monitored by CANFDx\_CHy\_TTOST.EL.

The M\_TTCAN signals the following error conditions as required by ISO 11898-4:

### **Config\_Error (S3)**

Sets error level CANFDx\_CHy\_TTOST.EL to '11' when a merged arbitrating time window is not properly closed or when there is a Tx\_Trigger with a time mark beyond the Tx\_Ref\_Trigger.

### **Watch\_Trigger\_Reached (S3)**

Sets error level CANFDx\_CHy\_TTOST.EL to '11' when a watch trigger is reached because the reference message is missing.

### **Application\_Watchdog (S3)**

Sets error level CANFDx\_CHy\_TTOST.EL to '11' when the application fails to serve the application watchdog. The application watchdog is configured via CANFDx\_CHy\_TTOST.AWL. It is served by reading the CANFDx\_CHy\_TTOST register. When the watchdog is not served in time, bit CANFDx\_CHy\_TTOST.AWE and interrupt flag CANFDx\_CHy\_TTIR.AW are set, all TTCAN communication is stopped, and the M\_TTCAN is set into bus monitoring mode (CANFDx\_CHy\_CCCR.MON set to '1').

### **CAN\_Bus\_Off (S3)**

Entering CAN\_Bus\_Off state sets error level CANFDx\_CHy\_TTOST.EL to '11'. CAN\_Bus\_Off state is signaled by CANFDx\_CHy\_PSR.BO = 1 and CANFDx\_CHy\_CCCR.INIT = 1.

### **Scheduling\_Error\_2 (S2)**

Sets error level CANFDx\_CHy\_TTOST.EL to '10' if the MSC of one Tx\_Trigger has reached 7. In addition, interrupt flag CANFDx\_CHy\_TTIR.SE2 is set. CANFDx\_CHy\_TTOST.EL is reset to 00 at the beginning of a matrix cycle when no Tx\_Trigger has an MSC of 7 in the preceding matrix cycle.

**Tx\_Overflow (S2)**

Sets error level CANFDx\_CHy\_TTOST.EL to '10' when the TX count is equal or higher than the expected number of Tx\_Triggers CANFDx\_CHy\_TTMLM.ENTT and a Tx\_Trigger event occurs. In addition, interrupt flag CANFDx\_CHy\_TTIR.TXO is set. CANFDx\_CHy\_TTOST.EL is reset to 00 when the TX count is no more than CANFDx\_CHy\_TTMLM.ENTT at the start of a new matrix cycle.

**Scheduling\_Error\_1 (S1)**

Sets error level CANFDx\_CHy\_TTOST.EL to '01' if within one matrix cycle the difference between the maximum MSC and the minimum MSC for all trigger memory elements (of exclusive time windows) is larger than 2, or if one of the MSCs of an exclusive Rx\_Trigger has reached 7. In addition, interrupt flag CANFDx\_CHy\_TTIR.SE1 is set. If within one matrix cycle none of these conditions is valid, CANFDx\_CHy\_TTOST.EL is reset to 00.

**Tx\_Underflow (S1)**

Sets error level CANFDx\_CHy\_TTOST.EL to '01' when the TX count is less than the expected number of Tx\_Triggers CANFDx\_CHy\_TTMLM.ENTT at the start of a new matrix cycle. In addition, interrupt flag CANFDx\_CHy\_TTIR.TXU is set. CANFDx\_CHy\_TTOST.EL is reset to 00 when the TX count is at least CANFDx\_CHy\_TTMLM.ENTT at the start of a new matrix cycle.

**17.5.8 TTCAN Message Handling**

*17.5.8.1 Reference Message*

For potential time masters, the identifier of the reference message is configured via CANFDx\_CHy\_TTRMC.RID. No dedicated TX buffer is required for transmission of the reference message. When a reference message is transmitted, the first data byte (TTCAN Level 1) and the first four data bytes (TTCAN Level 0 and Level 2) will be provided by the FSE.

If the Payload Select reference message CANFDx\_CHy\_TTRMC.RMPS is set, the rest of the reference message's payload (Level 1: bytes 2-8, Level 0 and Level 2: bytes 5-6) is taken from TX Buffer 0. In this case, the data length DLC code from message buffer 0 is used.

Table 17-14. Number of Data Bytes Transmitted with a Reference Messages

| CANFDx_CHy_TTRMC.RMPS | CANFDx_CHy_TXBRP.TRP0 | Level 0 | Level 1 | Level 2 |
|-----------------------|-----------------------|---------|---------|---------|
| 0                     | 0                     | 4       | 1       | 4       |
| 0                     | 1                     | 4       | 1       | 4       |
| 1                     | 0                     | 4       | 1       | 4       |
| 1                     | 1                     | 4 + MB0 | 1 + MB0 | 4 + MB0 |

To send additional payload with the reference message in Level 1, a DLC > 1 should be configured. For Level 0 and Level 2 a DLC > 4 is required. In addition, the transmission request pending bit CANFDx\_CHy\_TXBRP.TRP0 of message buffer 0 must be set (see Table 17-14). If CANFDx\_CHy\_TXBRP.TRP0 is not set when a reference message is started, the reference message is transmitted with the data bytes supplied by the FSE only.

For acceptance filtering of reference messages the Reference Identifier CANFDx\_CHy\_TTRMC.RID is used.

### 17.5.8.2 Message Reception

Message reception is done via the two RX FIFOs in the same way as for event-driven CAN communication.

The MSC is part of the corresponding trigger memory element and must be initialized to zero during configuration. It is updated while the M\_TTCAN is in synchronization states In\_Gap or In\_Schedule. The update happens at the message's Rx\_Trigger. At this point, it is checked at which acceptance filter element the latest message received in this basic cycle is matched. The matching filter number is stored as the acceptance filter result. If this is the same as the filter number defined in this trigger memory element, the MSC is decremented by one. If the acceptance filter result is not the same filter number as defined for this filter element, or if the acceptance filter result is cleared, the MSC is incremented by one. At each Rx\_Trigger and at each start of cycle, the last acceptance filter result is cleared.

The time mark of an Rx\_Trigger should be set to a value that ensures reception and acceptance filtering for the targeted message is completed. This should consider the RAM access time and the order of the filter list. It is recommended, that filters used for Rx\_Triggers are placed at the beginning of the filter list. It is not recommended to use an Rx\_Trigger for the reference message.

### 17.5.8.3 Message Transmission

For time-triggered message transmission, the M\_TTCAN supplies 32 dedicated TX buffers (see [TTCAN Configuration on page 219](#)). A TX FIFO or TX queue is not available when the M\_TTCAN is configured for time-triggered operation (CANFDx\_CHy\_TTOCF.OM = 01 or 10).

Each Tx\_Trigger in the trigger memory points to a particular TX buffer containing a specific message. There may be more than one Tx\_Trigger for a given TX buffer if that TX buffer contains a message that is to be transmitted more than once in a basic cycle or matrix cycle.

The application program must update the data regularly and on time, synchronized to the cycle time. The host CPU should ensure that no partially updated messages are transmitted. To assure this the host should proceed in the following way:

Tx\_Trigger\_Single/Tx\_Trigger\_Merged/Tx\_Trigger\_Arbitration:

- Check whether the previous transmission has completed by reading TXBTO
- Update the TX buffer's configuration and/or payload
- Issue an Add Request to set the TX Buffer Request Pending bit

Tx\_Trigger\_Continuous:

- Issue a Cancellation Request to reset the TX Buffer Request Pending bit
- Check whether the cancellation has finished by reading CANFDx\_CHy\_TXBCF
- Update TX buffer configuration and/or payload
- Issue an Add Request to set the TX Buffer Request Pending bit

The message MSC stored with the corresponding Tx\_Trigger provides information on the success of the transmission.

The MSC is incremented by one when the transmission cannot be started because the CAN bus was not idle within the corresponding transmit enable window or when the message was started but could not be completed successfully. The MSC is decremented by one when the message is transmitted successfully or when the message could have been started within its transmit enable window but was not started because transmission was disabled (M\_TTCAN in Error Level S2 or host has disabled this particular message).

The TX buffers may be managed dynamically – several messages with different identifiers may share the same TX buffer element. In this case the host must ensure that no transmission request is pending for the TX buffer element to be reconfigured by checking CANFDx\_CHy\_TXBRP.

If a TX buffer with pending transmission request should be updated, the host must first issue a cancellation request and check whether the cancellation has completed by reading CANFDx\_CHy\_TXBCF before it starts updating.

The TX handler will transfer a message from the message RAM to its intermediate output buffer at the trigger element, which becomes active immediately before the Tx\_Trigger element that defines the beginning of the transmit window. During and after transfer time, the transmit message may not be updated and its CANFDx\_CHy\_TXBRP bit may not be changed. To control this transfer time, an additional trigger element may be placed before the Tx\_Trigger. An example is a Time\_Base\_Trigger, which does not cause any other action. The difference in time marks between the Tx\_Trigger and the preceding trigger should be large enough to guarantee that the TX handler can read four words from the message RAM even at high RAM access load from other modules.

#### **Transmission in Exclusive Time Windows**

A transmission is started time-triggered when the cycle time reaches the time mark of a Tx\_Trigger\_Single or Tx\_Trigger\_Continuous. There is no arbitration on the bus with messages from other nodes. The MSC is updated according to the result of the transmission attempt. After successful transmission started by a Tx\_Trigger\_Single, the respective TX Buffer Request Pending bit is reset. After successful transmission started by a Tx\_Trigger\_Continuous the respective TX Buffer Request Pending bit remains set. When the transmission is not successful due to disturbances, it will be repeated the next time one of its Tx\_Triggers becomes active.

#### **Transmission in Arbitrating Time Windows**

A transmission is started time-triggered when the cycle time reaches the time mark of a Tx\_Trigger\_Arbitration. Several nodes may start to transmit at the same time. In this case the message has to arbitrate with the messages from other nodes. The MSC is not updated. When the transmission is not successful (lost arbitration or disturbance), it will be repeated the next time one of its Tx\_Triggers becomes active.

#### **Transmission in Merged Arbitrating Time Windows**

The purpose of a merged arbitrating time window is to enable multiple nodes to send a limited number of frames, which are transmitted in immediate sequence, the order given by CAN arbitration. It is not intended for burst transmission by a single node. Because the node does not have exclusive access within this time window, all requested transmissions may not be successful.

Messages that have lost arbitration or were disturbed by an error, may be retransmitted inside the same merged arbitrating time window. The retransmission will not be started if the corresponding Transmission Request Pending flag was reset by a successful TX cancellation.

In single transmit windows, the TX handler transmits the message indicated by the message number of the trigger element. In merged arbitrating time windows, it can handle up to three message numbers from the trigger list. Their transmissions will be attempted in the sequence defined by the trigger list. If the time mark of a fourth message is read before the first is transmitted (or canceled by the host), the fourth request will be ignored.

The transmission inside a merged arbitrating time window is not time-triggered. The transmission of a message may start before its time mark, or after the time mark if the bus was not idle.

The messages transmitted by a specific node inside a merged arbitrating time window will be started in the order of their Tx\_Triggers. Therefore, a message with low CAN priority may prevent the successful transmission of a following message with higher priority, if there is competing bus traffic. This should be considered for the configuration of the trigger list. Time\_Base\_Triggers may be placed between consecutive Tx\_Triggers to define the time until the data of the corresponding TX buffer needs to be updated.

## 17.5.9 TTCAN Interrupt and Error Handling

The TT Interrupt Register CANFDx\_CHy\_TTIR consists of four segments. Each interrupt can be enabled separately by the corresponding bit in the TT Interrupt Enable register CANFDx\_CHy\_TTIE. The flags remain set until the host clears them. A flag is cleared by writing a '1' to the corresponding bit position.

The first segment consists of flags CER, AW, WT, and IWT. Each flag indicates a fatal error condition where the CAN communication is stopped. With the exception of IWT, these error conditions require a reconfiguration of the M\_TTCAN module before the communication can be restarted.

The second segment consists of flags ELC, SE1, SE2, TXO, TXU, and GTE. Each flag indicates an error condition where the CAN communication is disturbed. If they are caused by a transient failure, such as by disturbances on the CAN bus, they will be handled by the TTCAN protocol's failure handling and do not require intervention by the application program.

The third segment consists of flags GTD, GTW, SWE, TTMI, and RTMI. The first two flags are controlled by global time events (Level 0 and Level 2 only) that require a reaction by the application program. With a Stop Watch Event, internal time values are captured. The Trigger Time Mark Interrupt notifies the application that a specific Time\_Base\_Trigger is reached. The Register Time Mark Interrupt signals that the time referenced by CANFDx\_CHy\_TTOCN.TMC (cycle, local, or global) equals time mark CANFDx\_CHy\_TTTMK.TM. It can also be used to finish a gap.

The fourth segment consists of flags SOG, CSM, SMC, and SBC. These flags provide a means to synchronize the application program to the communication schedule.

## 17.5.10 Level 0

TTCAN Level 0 is not part of ISO11898-4. This operation mode makes the hardware, that in TTCAN Level 2 maintains the calibrated global time base, also available for event-driven CAN according to ISO 11898-1:2015.

Level 0 operation is configured via CANFDx\_CHy\_TTOCF.OM = 11. In this mode, M\_TTCAN operates in event-driven CAN communication; there is no fixed schedule, the configuration of CANFDx\_CHy\_TTOCF.GEN is ignored. External event-synchronized operation is not available in Level 0. A synchronized time base is maintained by transmission of reference messages.

In Level 0 the trigger memory is not active and need not be configured. The time mark interrupt flag (CANFDx\_CHy\_TTIR.TTMI) is set when the cycle time has reached CANFDx\_CHy\_TTOCF.IRTO × 0x200. It reminds the host to set a transmission request for message buffer 0. The Watch\_Trigger interrupt flag (CANFDx\_CHy\_TTIR.WT) is set when the cycle time has reached 0xFF00. These values were chosen to have enough margin for a stable clock calibration. There are no further TT-error-checks.

Register time mark interrupts (CANFDx\_CHy\_TTIR.RTMI) are also possible.

The reference message is configured as for Level 2 operation. Received reference messages are recognized by the identifier configured in register CANFDx\_CHy\_TTRMC. For the transmission of reference messages only message buffer 0 may be used. The node transmits reference messages any time the host sets a transmission request for message buffer 0; there is no reference trigger offset.

Level 0 operation is configured via:

- CANFDx\_CHy\_TTRMC
- CANFDx\_CHy\_TTOCF except EVTP, AWL, GEN
- CANFDx\_CHy\_TTMLM except ENTT, TXEW
- CANFDx\_CHy\_TURCF

Level 0 operation is controlled via:

- CANFDx\_CHy\_TTOCN except NIG, TMG, FGP, GCS, TTMIE
- CANFDx\_CHy\_TTGTP
- CANFDx\_CHy\_TTTMK
- CANFDx\_CHy\_TTIR excluding bits CER, AW, IWT SE2, SE1, TXO, TXU, SOG (no function)

- CANFDx\_CHy\_TTIR – the following bits have changed function:
  - TTMI not defined by trigger memory - activated at cycle time CANFDx\_CHy\_TTOCF.IRTO × 0x200
  - WT not defined by trigger memory - activated at cycle time 0xFF00

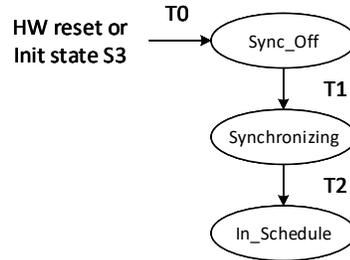
Level 0 operation is signaled via:

- CANFDx\_CHy\_TTOST excluding bits AWE, WFE, GSI, GFI, RTO (no function)

### 17.5.10.1 Synchronizing

Figure 17-26 describes the states and state transitions in TTCAN Level 0 operation. Level 0 has no In\_Gap state.

Figure 17-26. Level 0 Schedule Synchronization State Machine



T0: transition condition always taking prevalence

T1: Init state left, cycle time is zero

T2: at least two successive reference messages observed  
(last reference message did not contain a set Disc\_Bit bit)

### 17.5.10.2 Handling Error Levels

During Level 0 operation only the following error conditions may occur:

- Watch\_Trigger\_Reached (S3), reached cycle time 0xFF00
- CAN\_Bus\_Off (S3)

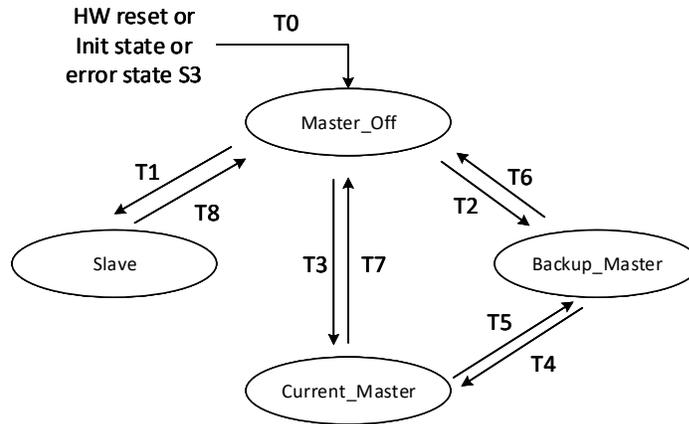
Because S1 and S2 errors are not possible, the error level can only switch between S0 (No Error) and S3 (Severe Error). In TTCAN Level 0 an S3 error is handled differently. When S3 error is reached, both CANFDx\_CHy\_TTOST.SYS and CANFDx\_CHy\_TTOST.MS are reset, and interrupt flags CANFDx\_CHy\_TTIR.GTE and CANFDx\_CHy\_TTIR.GTD are set.

When S3 (CANFDx\_CHy\_TTOST.EL = 11) is entered, bus monitoring mode is, contrary to TTCAN Level 1 and Level 2, not entered. S3 error level is left automatically after transmission (time master) or reception (time slave) of the next reference message.

### 17.5.10.3 Master Slave Relation

Figure 17-27 describes the master slave relation in TTCAN Level 0. In case of an S3 error, the M\_TTCAN returns to state Master\_Off.

Figure 17-27. Level 0 Master to Slave Relation



- T0: transition condition always taking prevalence
- T1: reference message observed when not potential master
- T2: reference message observed with master priority != own master priority, error state != S3
- T3: reference message observed with master priority = own master priority, error state != S3
- T4: reference message observed with own master priority
- T5: reference message observed with master priority higher than own master priority
- T6: error state S3
- T7: error state S3
- T8: error state S3

### 17.5.11 Synchronization to External Time Schedule

This feature can be used to synchronize the phase of the M\_TTCAN's schedule to an external schedule (for example, that of a second TTCAN network). It is applicable only when the M\_TTCAN is current time master (CANFDx\_CHy\_TTOST.MS = 11).

External synchronization is controlled by the CANFDx\_CHy\_TTOCN.ESCN bit. If CANFDx\_CHy\_TTOCN.ESCN is set, at rising edge of the internal event trigger pin, the M\_TTCAN compares its actual cycle time with the target phase value configured by CANFDx\_CHy\_TTGTP.CTP.

Before setting CANFDx\_CHy\_TTOCN.ESCN, the host should adapt the phases of the two time schedules, for example, by using the TTCAN gap control (see 17.5.3 TTCAN Gap Control). When the host sets CANFDx\_CHy\_TTOCN.ESCN, CANFDx\_CHy\_TTOST.SPL is set.

If the difference between the cycle time and target phase value CANFDx\_CHy\_TTGTP.CTP at the trigger is greater than 9 NTU, the phase lock bit CANFDx\_CHy\_TTOST.SPL is reset, and interrupt flag CANFDx\_CHy\_TTIR.CSM is set. CANFDx\_CHy\_TTOST.SPL is also reset (and CANFDx\_CHy\_TTIR.CSM is set), when another node becomes time master.

If both CANFDx\_CHy\_TTOST.SPL and CANFDx\_CHy\_TTOCN.ESCN are set, and if the difference between the cycle time and the target phase value CANFDx\_CHy\_TTGTP.CTP is less or equal 9 NTU, the phase lock bit CANFDx\_CHy\_TTOST.SPL remains set, and the measured difference is used as reference trigger offset value to adjust the phase at the next transmitted reference message.

**Note:** The rising edge detection at the internal pin is enabled at the start of each basic cycle. The first rising edge triggers the compare of the actual cycle time with CANFDx\_CHy\_TTGTP.CTP. All further edges until the beginning of the next basic cycle are ignored.

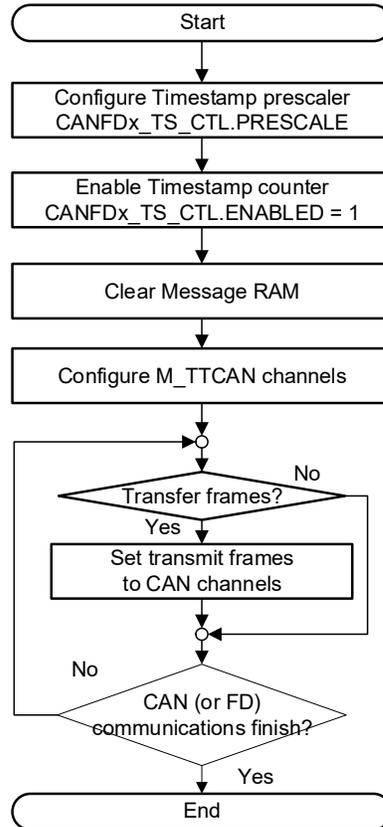
## 17.6 Setup Procedures

This section provides example procedures for configurations of M\_TTCAN group and flow for respective M\_TTCAN channels.

### 17.6.1 General Program Flow

This is a general flow to configure the M\_TTCAN module.

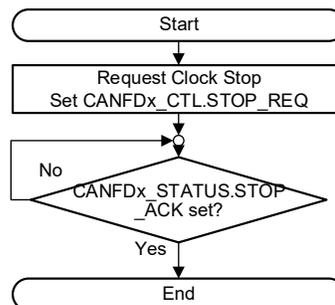
Figure 17-28. General Program Flow



### 17.6.2 Clock Stop Request

To save power, the application can stop providing clock to unused M\_TTCAN channels by following these steps.

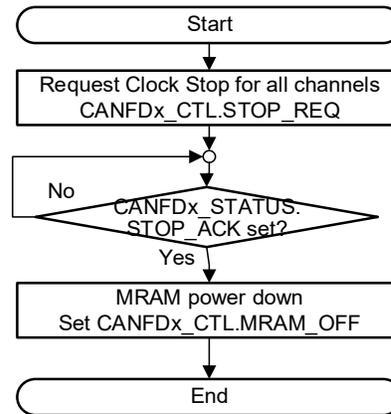
Figure 17-29. Clock Stop Request Procedure



To resume providing clock, the CANFDx\_CTL.STOP\_REQ bit should be reset.

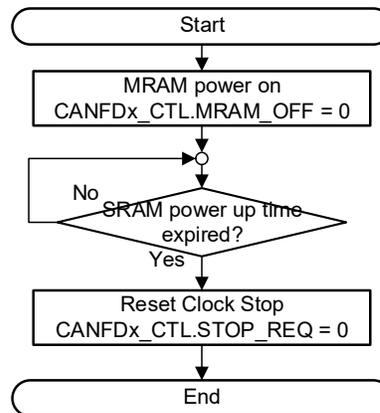
### 17.6.3 Message RAM OFF Operation

Figure 17-30. Message RAM OFF Operation



### 17.6.4 Message RAM ON Operation

Figure 17-31. Message RAM On Procedure

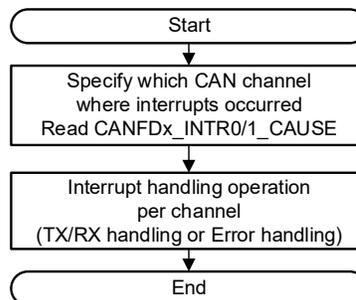


After switching message RAM ON again, software needs to allow a certain power-up time before message RAM can be used; that is, before STOP\_REQ can be deasserted. The power-up time is equivalent to the system SRAM power-up time specified in the CPUSS.RAM\_PWR\_DELAY\_CTL register.

### 17.6.5 Consolidated Interrupt Handling

When using consolidated interrupt for the M\_TTCAN group, follow the procedure given in [Figure 17-32](#).

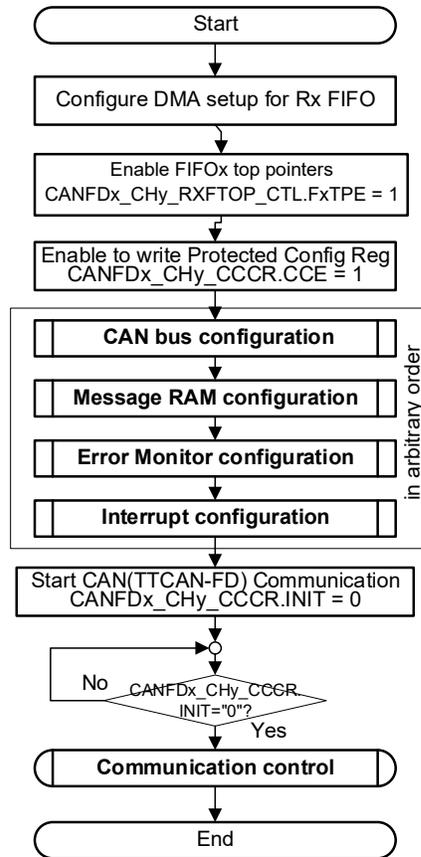
Figure 17-32. Consolidated Interrupt Processing



### 17.6.6 Procedures Specific to M\_TTCAN Channel

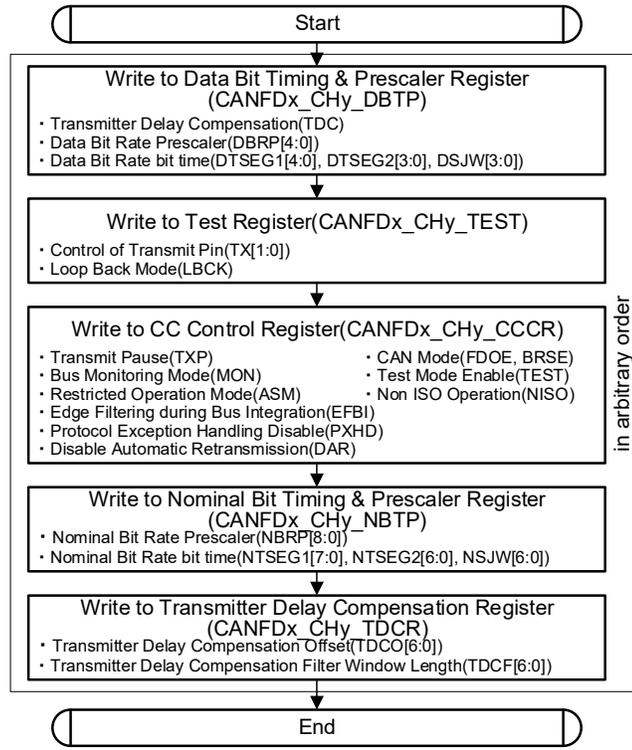
This section describes sample procedures per channel. If several M\_TTCAN channels are used, the application should configure each channel as shown in Figure 17-33. The figure shows the general program flow (per channel).

Figure 17-33. Configuration Sequence Specific to Channel



### 17.6.6.1 CAN Bus Configuration

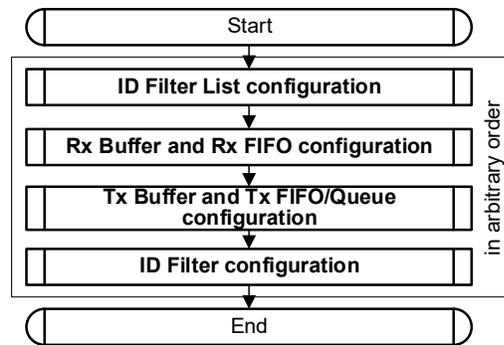
Figure 17-34. Configuration Required for CAN Bus



### 17.6.6.2 Message RAM Configuration

Figure 17-35 shows an overview of the message RAM configuration.

Figure 17-35. Message RAM Configuration Overview



Each configuration mentioned in the overview is detailed in Figure 17-36 through Figure 17-39.

Figure 17-36. ID Filter List Configuration

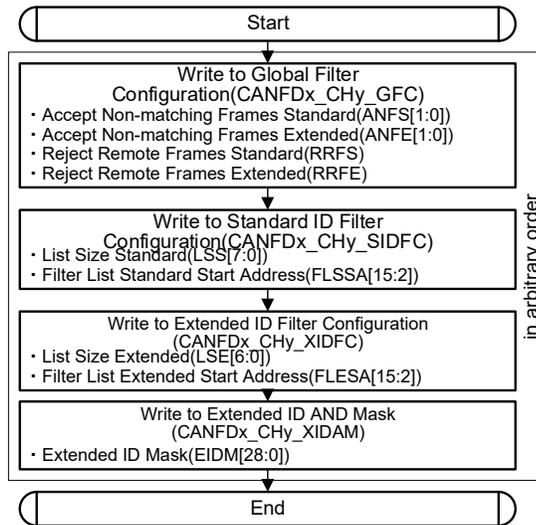


Figure 17-37. RX FIFO and RX Buffer Configuration

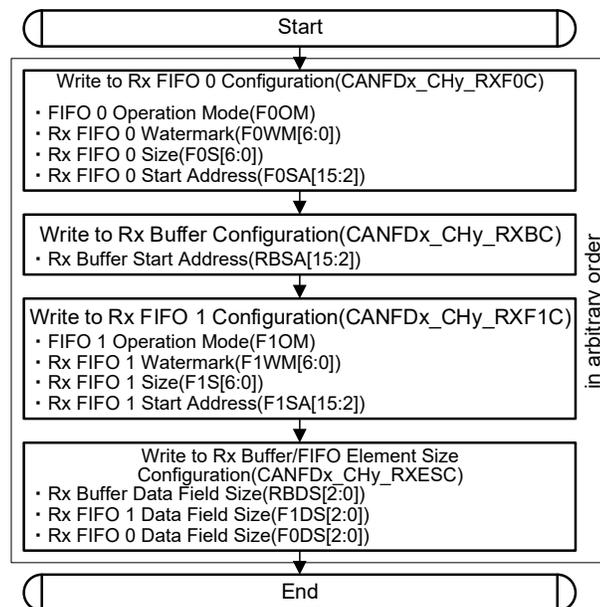


Figure 17-38. TX Buffer and TX FIFO/Queue Configuration

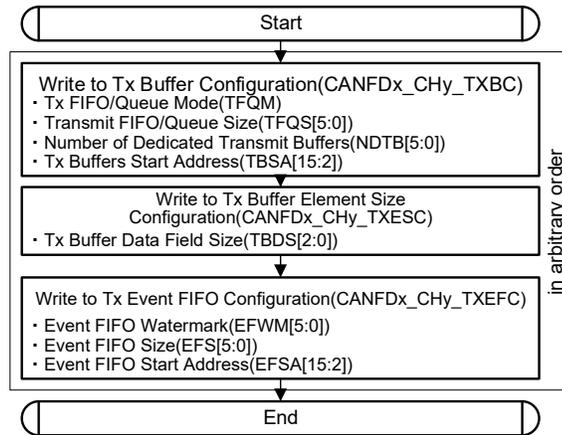
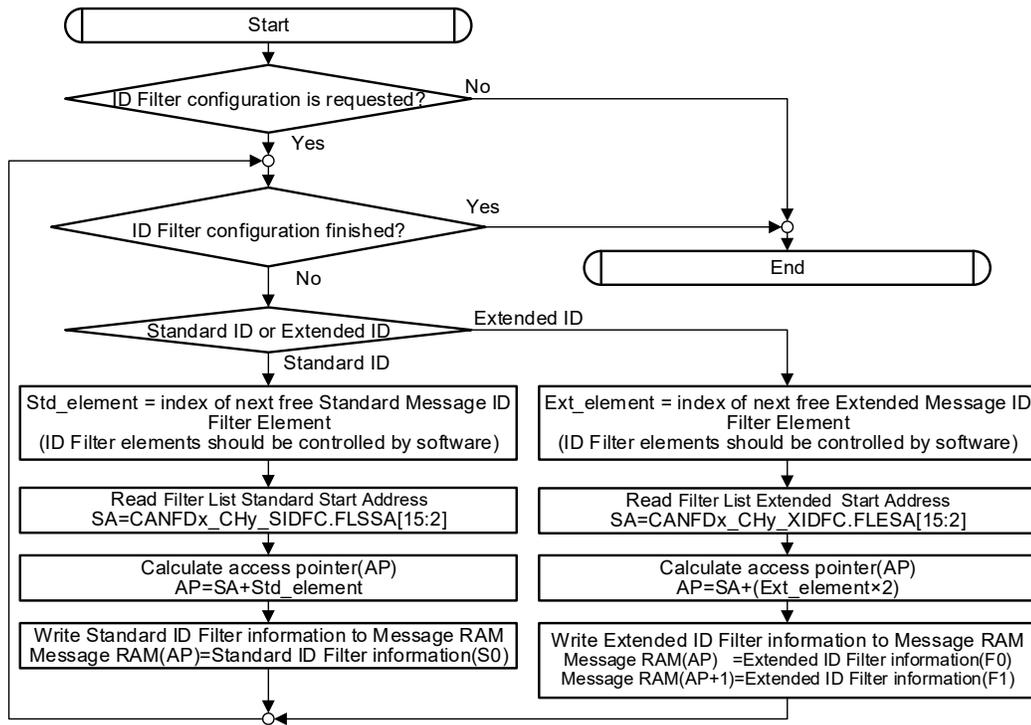
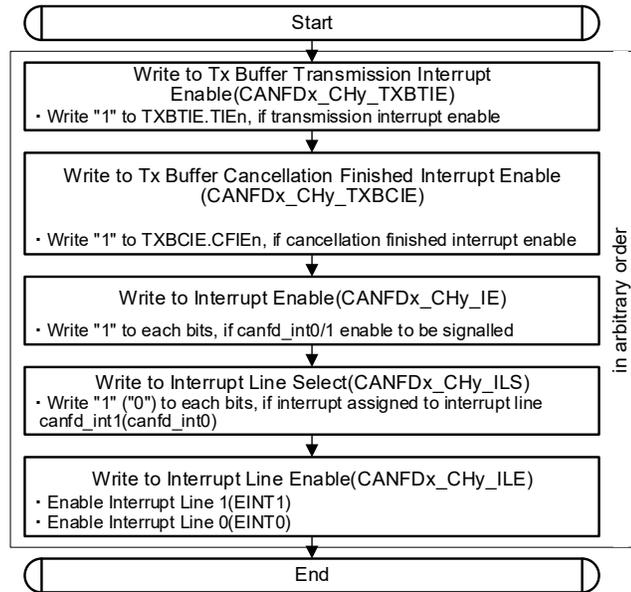


Figure 17-39. ID Filter Configuration



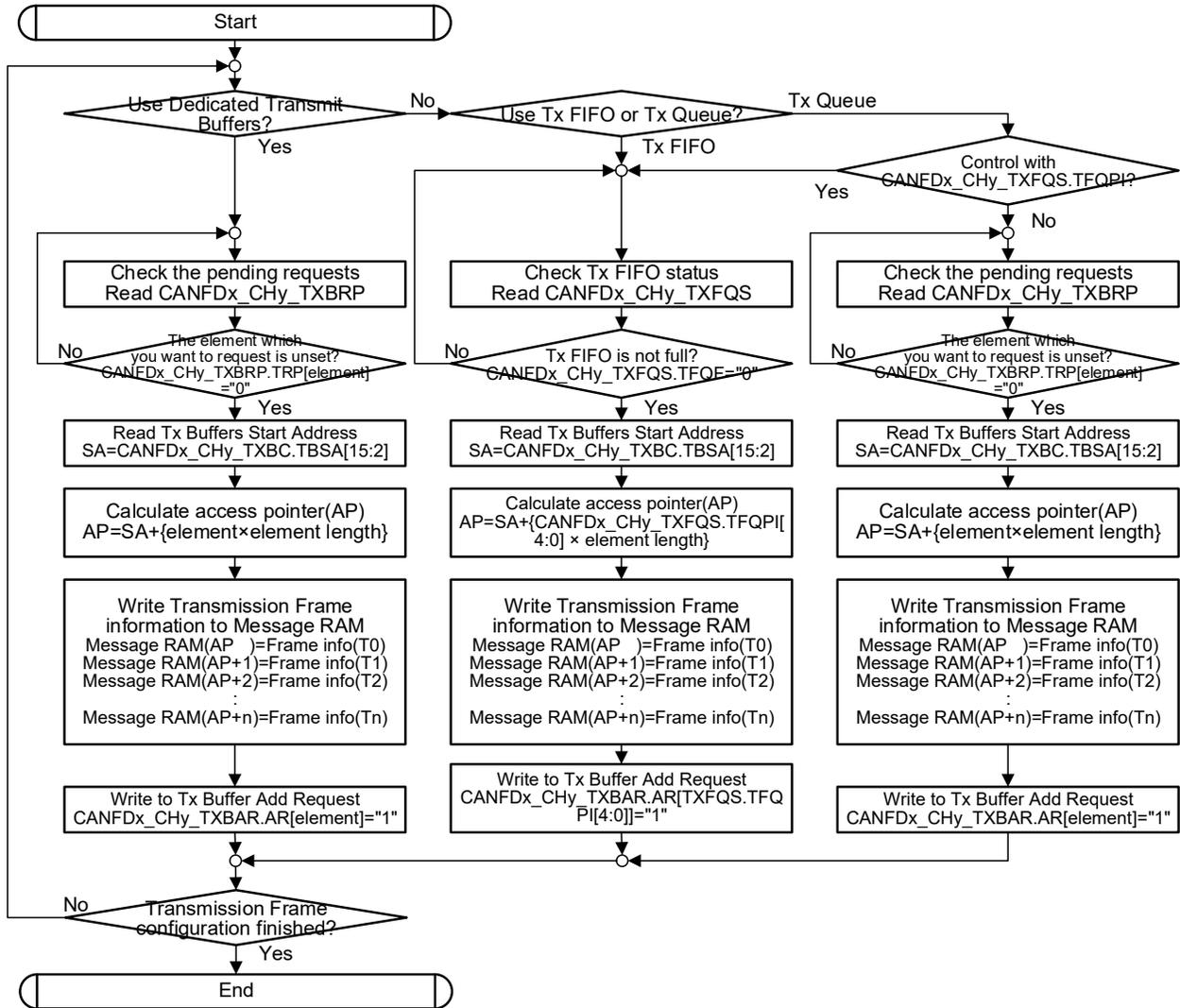
### 17.6.6.3 Interrupt Configuration

Figure 17-40. Interrupt Configuration



17.6.6.4 Transmit Frame Configuration

Figure 17-41. Transmit Frame Configuration



### 17.6.6.5 Interrupt Handling

When consolidated interrupts are configured, INTR0/1\_CAUSE register will be read to find out the source M\_TTCAN channel for the triggered interrupt. Figure 17-42 shows a general interrupt handling flow chart.

Figure 17-42. Interrupt Handling

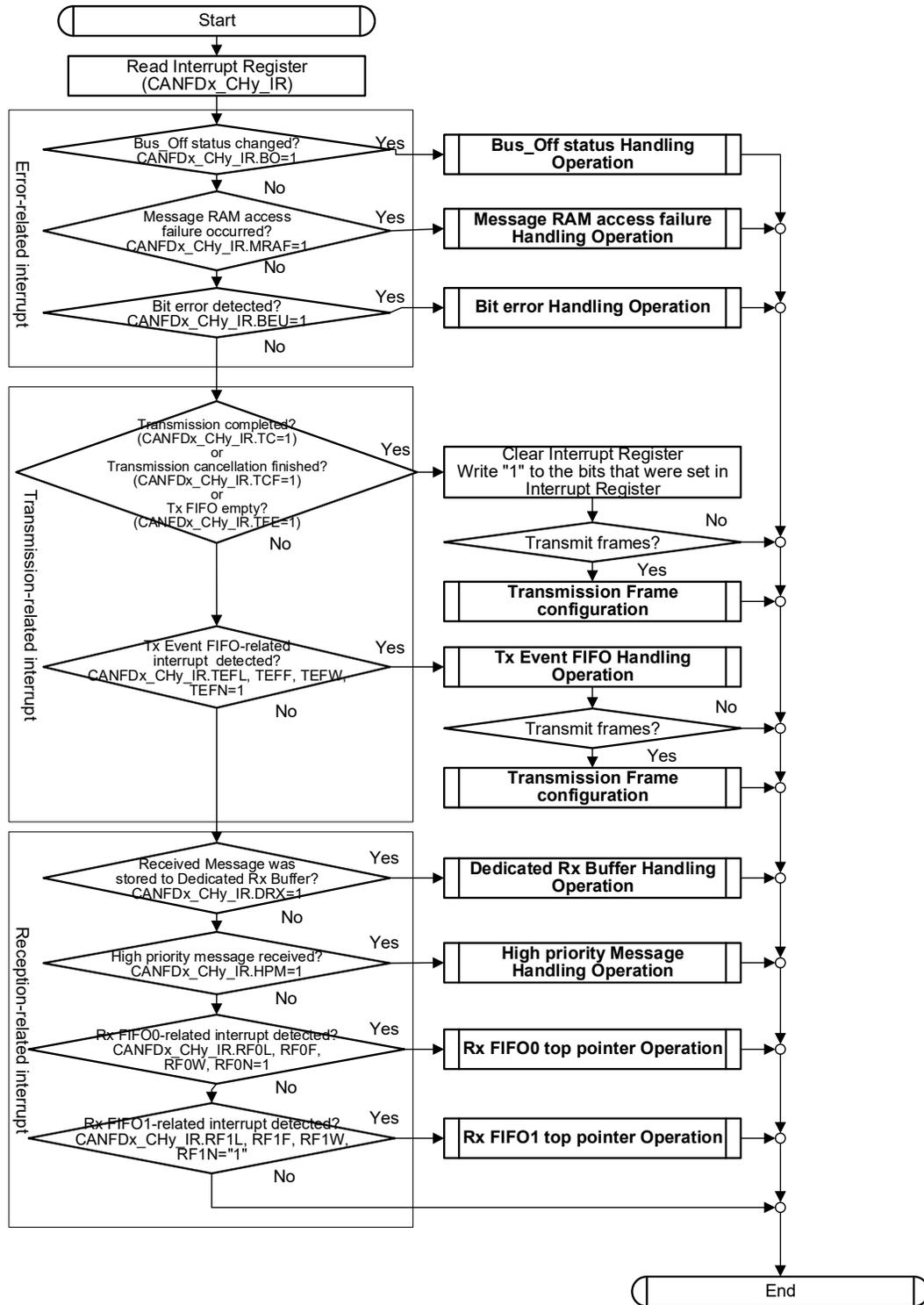


Figure 17-43. Bus OFF Error Handling

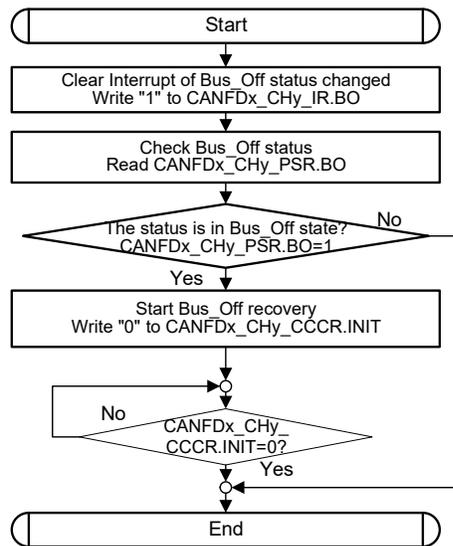


Figure 17-44. Bit Error Handling

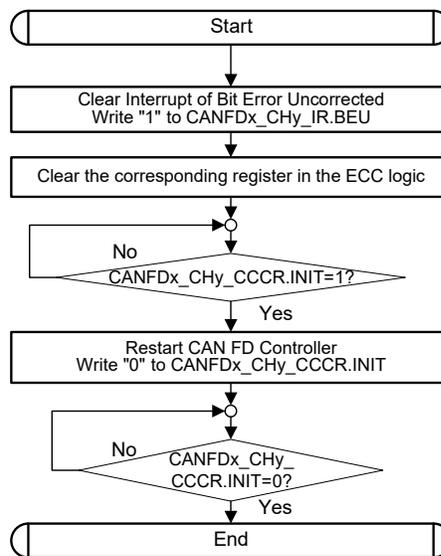


Figure 17-45. Message RAM Access Failure Handling

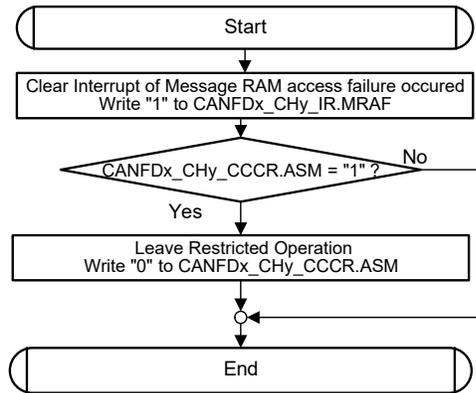


Figure 17-46. TX Event FIFO Handling

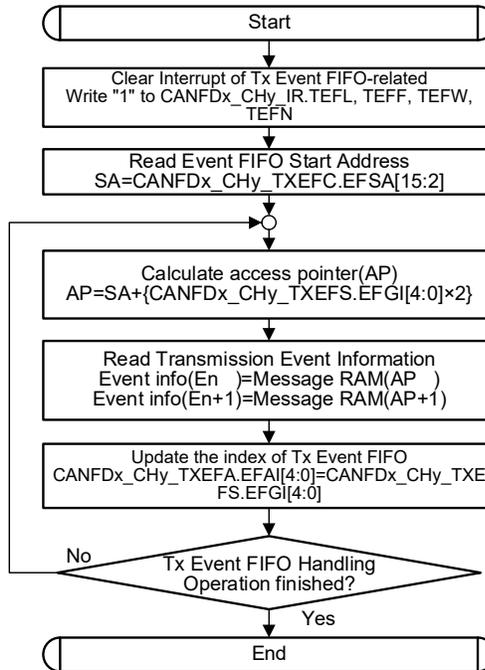


Figure 17-47. Dedicated RX Buffer Handling

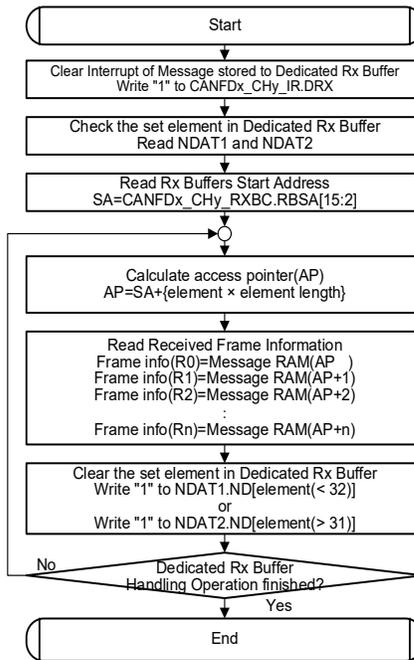


Figure 17-48. High Priority Message Handling

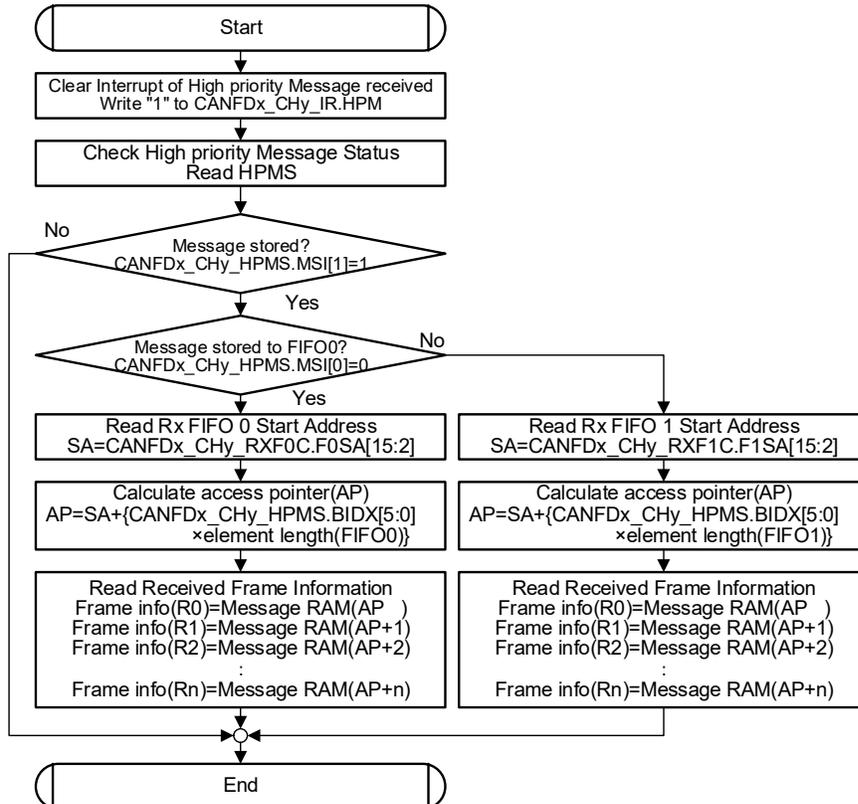
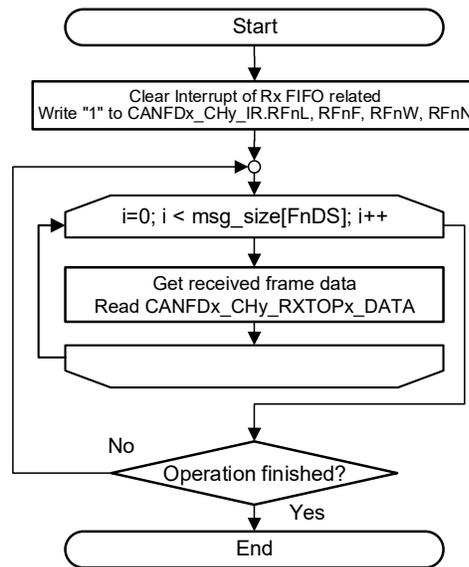


Figure 17-49. RX FIFO Top Pointer Handling



**Note:** 'x' in CANFDx signifies the CAN macro instance and 'y' in CANFDx\_CHy signifies the channel under the CAN instance.

# 18. Timer, Counter, and PWM (TCPWM)



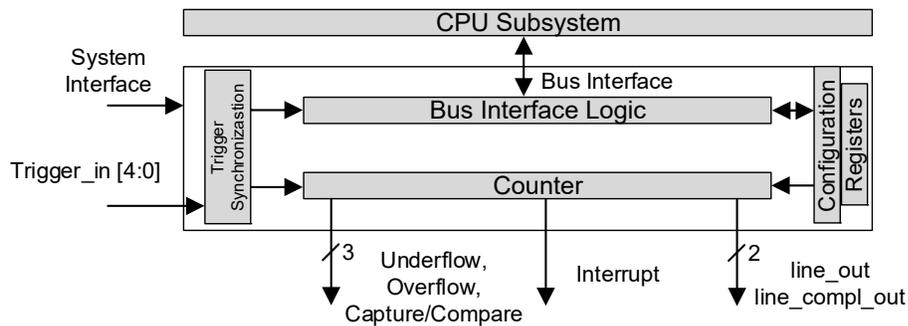
The Timer, Counter, Pulse Width Modulator (TCPWM) block in the PSoC 4 MCU uses a 16-bit counter, which can be configured as a timer, counter, pulse width modulator (PWM), or quadrature decoder. The block can be used to measure the period and pulse width of an input signal (timer), find the number of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

## 18.1 Features

- The TCPWM block supports the following operational modes:
  - Timer-counter with compare
  - Timer-counter with capture
  - Quadrature decoding
  - Pulse width modulation
  - Pseudo-random PWM
  - PWM with dead time
- Up, Down, and Up/Down counting modes.
- Clock prescaling (division by 1, 2, 4, ... 64, 128)
- Double buffering of compare/capture and period values
- Underflow, overflow, and capture/compare output signals
- Supports interrupt on:
  - Terminal count – Depends on the mode; typically occurs on overflow or underflow
  - Capture/compare – The count is captured to the capture register or the counter value equals the value in the compare register
- Complementary output for PWMs
- Selectable start, reload, stop, count, and capture event signals (events refer to peripheral generated signals that trigger specific functions in each counter in the TCPWM block) for each TCPWM – with rising edge, falling edge, both edges, and level trigger options

## 18.2 Architecture

Figure 18-1. TCPWM Block Diagram



The TCPWM block can contain up to eight counters. Each counter can be 16-bit wide. The three main registers that control the counters are:

- TCPWM\_CNT\_CC is used to capture the counter value in CAPTURE mode. In all other modes this value is compared to the counter value.
- TCPWM\_CNT\_COUNTER holds the current counter value.
- TCPWM\_CNT\_PERIOD holds the upper value of the counter. When the counter counts for  $n$  cycles, this field should be set to  $n-1$ .

In this chapter, a TCPWM refers to the entire block and all the counters inside. A counter refers to the individual counter inside the TCPWM. Within a TCPWM block the width of each counter is the same.

TCPWM has these interfaces:

- I/O signal interface: Consists of input triggers (such as reload, start, stop, count, and capture) and output signals (such as line\_out, line\_compl\_out, overflow (OV), underflow (UN), and capture/compare (CC)). All of these input signals are used to trigger an event within the counter, such as a reload trigger generating a reload event. The output signals are generated by internal events (underflow, overflow, and capture/compare) and can be connected to other peripherals to trigger events.
- Interrupts: Provides interrupt request signals from each counter, based on TC or CC conditions.

The TCPWM block can be configured by writing to the TCPWM registers. See [“TCPWM Registers” on page 284](#) for more information on all registers required for this block.

### 18.2.1 Enabling and Disabling Counters in a TCPWM Block

A counter can be enabled by setting the corresponding bit to 1 in the COUNTER\_ENABLED field of the control register TCPWM\_CTRL. It can be disabled by setting the same bit back to 0.

**Note:** The counter must be configured before enabling it. Disabling the counter retains the values in the configuration registers.

### 18.2.2 Clocking

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

#### 18.2.2.1 Clock Prescaling

clk\_counter can be further divided inside each counter, with values of 1, 2, 4, 8...64, 128. This division is called prescaling. The prescaling is set in the GENERIC field of the TCPWM\_CNT\_CTLR register.

**Note:** Clock prescaling is not available in quadrature mode and pulse width modulation mode with dead time.

### 18.2.2.2 Count Input

The counter increments or decrements on a prescaled clock in which the count input is active – “active count”.

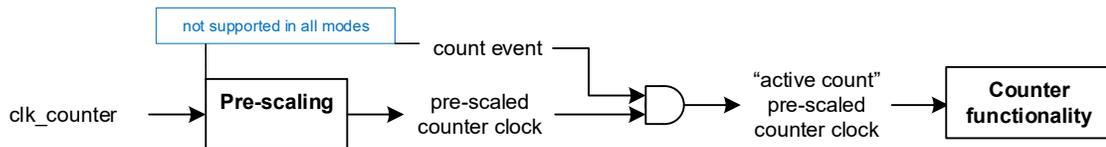
When the count input is configured as level, the count value is changed on each prescaled `clk_counter` edge in which the count input is high.

When the count input is configured as rising/falling the count value is changed on each prescaled `clk_counter` edge in which an edge is detected on the count input.

The next section contains additional details on edge detection configuration.

**Note:** Count events are not supported in quadrature and pulse-width modulation pseudo-random modes; the `clk_counter` is used in these cases instead of the active count prescaled clock.

Figure 18-2. Counter Clock Generation



**Note:** The count event and pre-scaled counter clock are AND together, which means that a count event must occur to generate an active count pre-scaled counter clock.

### 18.2.3 Trigger Inputs

Each TCPWM block has 14 `Trigger_In` signals, which come from other on-chip resources and low power comparators. The `Trigger_In` signals are shared with all counters inside of one TCPWM block. Use the Trigger Mux registers to configure which signals get routed to the `Trigger_In` for each TCPWM block. See the [Trigger Multiplexer Block chapter on page 113](#) for more details. Two constant trigger inputs ‘0’ and ‘1’ are available in addition to the 14 `Trigger_In`. For each counter, the trigger input source is selected using the `TCPWM_CNT_TR_CTRL0` register.

Each counter can select any of the 16 trigger signals to be the source for any of the following events:

- Reload
- Start
- Stop/Kill
- Count
- Capture/swap

**Note:** In the `TCPWM_CMD` register, the `COUNTER_START`, `COUNTER_STOP`, `COUNTER_RELOAD`, and `COUNTER_CAPTURE` sections are used to trigger start, stop/kill, reload, and capture from software.

The sections describing each TCPWM mode will describe the function of each input in detail.

Typical operation uses the reload input to initialize and start the counter and the stop input to stop the counter. When the counter is stopped, the start input can be used to start the counter with its counter value unmodified from when it was stopped.

If stop, reload, and start coincide, the following precedence relationship holds:

- A stop has higher priority than a reload.
- A reload has higher priority than a start.

As a result, when a reload or start coincides with a stop, the reload or start has no effect.

Before going to the counter each `Trigger_IN` can pass through a positive edge detector, negative edge detector, both edge detector, or pass straight through to the counter. This is controlled using `TCPWM_CNT_TR_CTRL1`. In the quadrature mode, edge detection is done using `clk_counter`. For all other modes, edge detection is done using the `clk_hf_counter` which is the gated version of the `clk_l_hf` and has the same frequency as `clk_hf`.

Multiple detected events are treated as follows:

- In the rising edge and falling edge modes, multiple events are effectively reduced to a single event. As a result, events may be lost (see Figure 18-3).
- In the rising/falling edge mode, an even number of events are not detected and an odd number of events are reduced to a single event. This is because the rising/falling edge mode is typically used for capture events to determine the width of a pulse. The current functionality will ensure that the alternating pattern of rising and falling is maintained (see Figure 18-4).

Figure 18-3. Multiple Rising Edge Capture

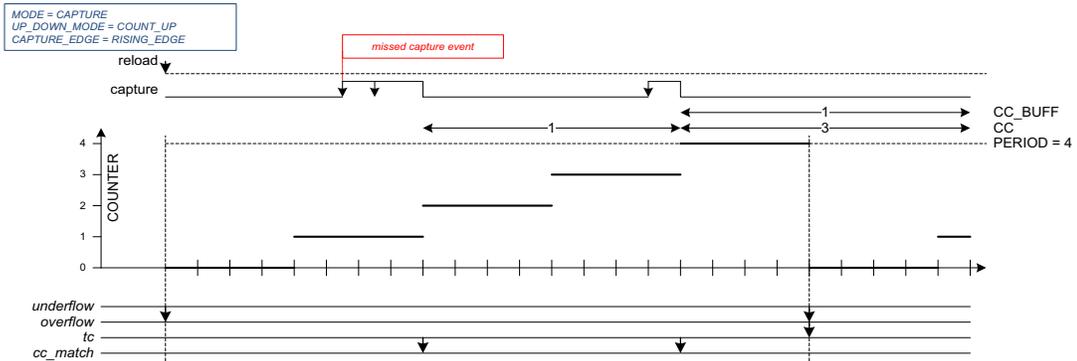


Figure 18-4. Multiple Both Edge Capture

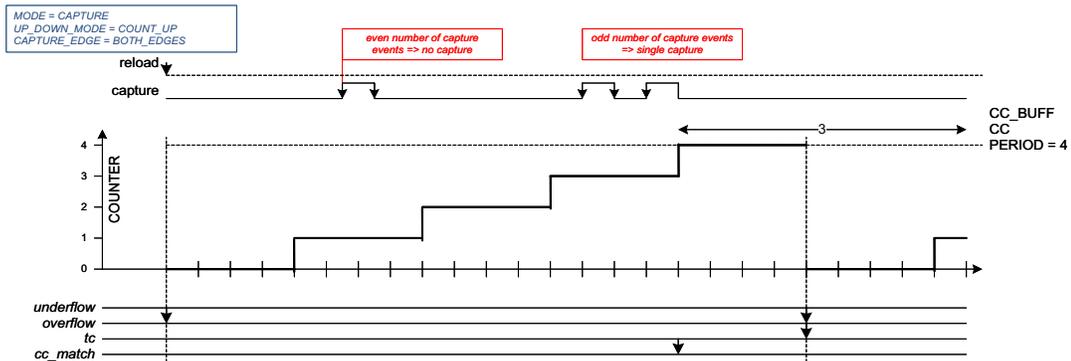


Figure 18-5. TCPWM Input Events

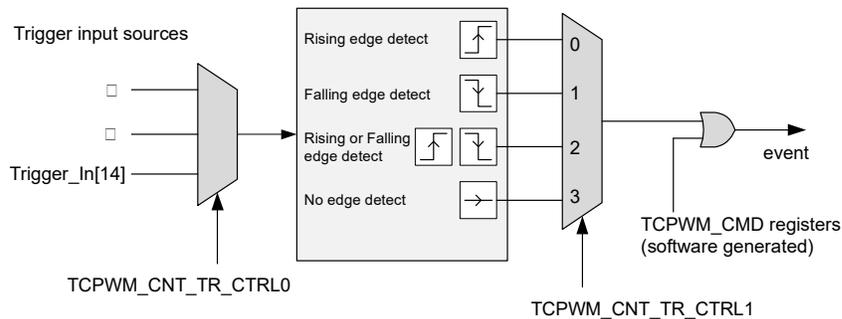
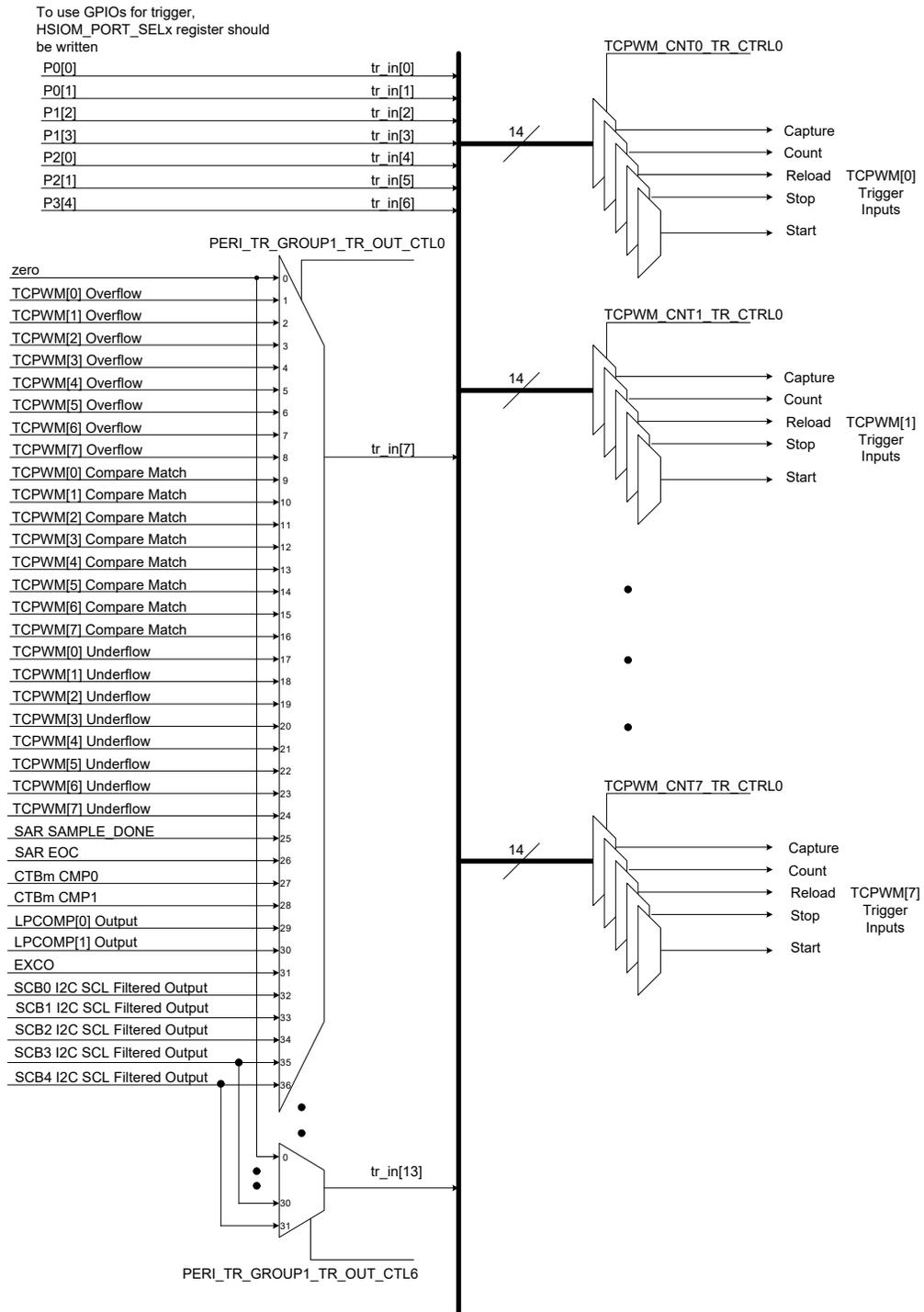


Figure 18-6. TCPWM Trigger Sources for PSoC 4100S Max



**Notes:**

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

## 18.2.4 Trigger Outputs

Each counter can generate three trigger output events. These trigger output events can be routed through the trigger mux to other peripherals on the device. The three trigger outputs are:

- **Overflow (OV):** An overflow event indicates that in up-counting mode, COUNTER equals the PERIOD register, and is changed to a different value.
- **Underflow (UN):** An underflow event indicates that in down-counting mode, COUNTER equals 0, and is changed to a different value.
- **Compare/Capture (CC):** This event is generated when the counter is running and one of the following conditions occur:
  - Counter equals the compare value. This event is either generated when the match is about to occur (COUNTER does not equal the CC register and is changed to CC) or when the match is not about to occur (COUNTER equals CC and is changed to a different value).
  - A capture event has occurred and the CC/CC\_BUFF registers are updated.

**Note:** These signals remain high only for two cycles of clk\_sys.

## 18.2.5 Interrupts

The TCPWM block provides a dedicated interrupt output for each counter. This interrupt can be generated for a terminal count (TC) or CC event. A TC is the logical OR of the OV and UN events.

Four registers are used to handle interrupts in this block, as shown in [Table 18-1](#).

Table 18-1. Interrupt Register

| Interrupt Registers  | Bits | Name     | Description  |
|--|------|----------|--|
| TCPWM_CNT_INTR<br>(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<br>(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<br>(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<br>(Interrupt masked request register) | 0    | TC       | Logical AND of the corresponding TC request and mask bits.   |
|  | 1    | CC_MATCH | Logical AND of the corresponding CC_MATCH request and mask bits.   |

## 18.2.6 PWM Outputs

Each counter has two outputs, pwm (line\_out) and pwm\_n (line\_compl\_out) (complementary of pwm). Note that the OV, UN, and CC conditions are used to drive line\_out and line\_compl\_out, by configuring the TCPWM\_CNT\_TR\_CTRL2 register (see [Table 18-2](#)).

Table 18-2. Configuring Output for OV, UN, and CC Conditions

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

Table 18-2. Configuring Output for OV, UN, and CC Conditions (continued)

| Field                            | Bit | Value | Event           | Description                                      |
|----------------------------------|-----|-------|-----------------|--|
| UNDERFLOW_MODE Default Value = 3 | 5:4 | 0     | Set pwm to '1   | Configures output line on a underflow (UN) event |
|                                  |     | 1     | Clear pwm to '0 |  |
|                                  |     | 2     | Invert pwm      |  |
|                                  |     | 3     | No change       |  |

### 18.2.7 Power Modes

The TCPWM block works in Active and Sleep modes. The TCPWM block is powered from  $V_{CCD}$ . The configuration registers and other logic are powered in Deep Sleep mode to keep the states of configuration registers. See [Table 18-3](#) for details.

Table 18-3. Power Modes in TCPWM Block

| Power Mode       | Block Status  |
|------------------|---|
| CPU Active       | This block is fully operational in this mode with clock running and power switched on.                |
| CPU Sleep        | The CPU is in sleep but the block is still functional in this mode. All counter clocks are on.        |
| CPU Deep Sleep   | Both power and clocks to the block are turned off, but configuration registers retain their states.   |
| System Hibernate | In this mode, the power to this block is switched off. Configuration registers will lose their state. |

## 18.3 Operation Modes

The counter block can function in six operational modes, as shown in [Table 18-4](#). The MODE [26:24] field of the counter control register (TCPWM\_CNTx\_CTRL) configures the counter in the specific operational mode.

Table 18-4. Operational Mode Configuration

| Mode       | MODE Field [26:24] | Description  |
|------------|--------------------|--|
| Timer      | 000                | The counter increments or decrements by '1' at every clk_counter cycle in which a count event is detected. The Compare/Capture register is used to compare the count.          |
| Capture    | 010                | The counter increments or decrements by '1' at every clk_counter cycle in which a count event is detected. A capture event copies the counter value into the capture register. |
| Quadrature | 011                | Quadrature decoding. The counter is decremented or incremented based on two phase inputs according to an X1, X2, or X4 decoding scheme.  |
| PWM        | 100                | Pulse width modulation.  |
| PWM_DT     | 101                | Pulse width modulation with dead time insertion.   |
| PWM_PR     | 110                | Pseudo-random PWM using a 16-bit linear feedback shift register (LFSR) to generate pseudo-random noise.  |

The counter can be configured to count up, down, and up/down by setting the UP\_DOWN\_MODE[17:16] field in the TCPWM\_CNT\_CTRL register, as shown in [Table 18-5](#).

Table 18-5. 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) and Overflow (OV) condition is generated when the counter changes from the period value.  |
| DOWN Counting Mode      | 01                  | Decrements the counter from the period value until 0 is reached. A TC and Underflow (UN) condition is generated when the counter changes from a value of '0'.   |
| UP/DOWN Counting Mode 1 | 10                  | Increments the counter until the period value is reached, and then decrements the counter until '0' is reached. TC and UN conditions are generated only when the counter changes from a value of '0'.   |
| UP/DOWN Counting Mode 2 | 11                  | Similar to up/down counting mode 1 but a TC condition is generated when the counter changes from '0' and when the counter value changes from the period value. OV and UN conditions are generated similar to how they are generated in UP and DOWN counting modes respectively. |

### 18.3.1 Timer Mode

The timer mode can be used to measure how long an event takes or the time difference between two events. The timer functionality increments/decrements a counter between 0 and the value stored in the PERIOD register. When the counter is running, the count value stored in the COUNTER register is compared with the compare/capture register (CC). When the counter changes from a state in which COUNTER equals CC, the cc\_match event is generated.

Timer functionality is typically used for one of the following:

- Timing a specific delay – the count event is a constant '1'.
- Counting the occurrence of a specific event – the event should be connected as an input trigger and selected for the count event.

Table 18-6. Timer Mode Trigger Input Description

| Trigger Inputs | Usage   |
|----------------|---|
| Reload         | Sets the counter value and starts the counter. Behavior is dependent on UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The counter is set to "0" and count direction is set to "up".</li> <li>■ COUNT_DOWN: The counter is set to PERIOD and count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The counter is set to "1" and count direction is set to "up".</li> </ul> Can be used when the counter is running or not running.   |
| Start          | Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE. When the counter is not running: <ul style="list-style-type: none"> <li>■ COUNT_UP: The count direction is set to "up".</li> <li>■ COUNT_DOWN: The count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The count direction is not modified.</li> </ul> Note that when the counter is running, the start event has no effect.<br>Can be used when the counter is running or not running. |
| Stop           | Stops the counter.  |
| Count          | Count event increments/decrements the counter.  |
| Capture        | Not used.   |

Incrementing and decrementing the counter is controlled by the count event and the counter clock clk\_counter. Typical operation will use a constant '1' count event and clk\_counter without pre-scaling. Advanced operations are also possible; for example, the counter event configuration can decide to count the rising edges of a synchronized input trigger.

Table 18-7. Timer Mode Supported Features

| Supported Features | Description  |
|--------------------|--|
| Clock pre-scaling  | Pre-scales the counter clock clk_counter.  |
| One-shot           | Counter is stopped by hardware, after a single period of the counter: <ul style="list-style-type: none"> <li>■ COUNT_UP: on an overflow event.</li> <li>■ COUNT_DOWN, COUNT_UPDN1/2: on an underflow event.</li> </ul>   |
| Auto reload CC     | CC and CC_BUFF are exchanged on a cc_match event (when specified by CTRL.AUTO_RELOAD_CC)   |
| Up/down modes      | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The counter counts from 0 to PERIOD.</li> <li>■ COUNT_DOWN: The counter counts from PERIOD to 0.</li> <li>■ COUNT_UPDN1/2: The counter counts from 1 to PERIOD and back to 0.</li> </ul> |

Table 18-8 lists the trigger outputs and the conditions when they are triggered.

Table 18-8. Timer Mode Trigger Outputs

| Trigger Outputs | Description  |
|-----------------|--|
| cc_match (CC)   | Counter changes from a state in which COUNTER equals CC.                         |
| Underflow (UN)  | Counter is decrementing and changes from a state in which COUNTER equals "0".    |
| Overflow (OV)   | Counter is incrementing and changes from a state in which COUNTER equals PERIOD. |

**Note:** Each output is only two clk\_sys wide and is represented by an arrow in the timing diagrams in this chapter, for example see Figure 18-8.

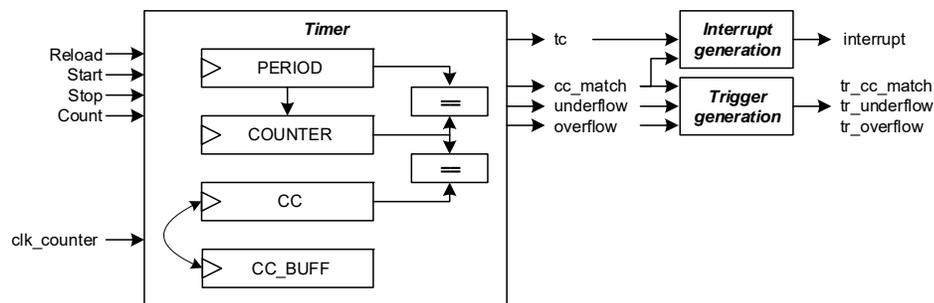
Table 18-9. Timer Mode Interrupt Outputs

| Interrupt Outputs | Description  |
|-------------------|--|
| tc                | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The tc event is the same as the overflow event.</li> <li>■ COUNT_DOWN: The tc event is the same as the underflow event.</li> <li>■ COUNT_UPDN1: The tc event is the same as the underflow event.</li> <li>■ COUNT_UPDN2: The tc event is the same as the logical OR of the overflow and underflow events.</li> </ul> |
| cc_match (CC)     | Counter changes from a state in which COUNTER equals CC.   |

Table 18-10. Timer Mode PWM Outputs

| PWM Outputs    | Description |
|----------------|-------------|
| line_out       | Not used.   |
| line_compl_out | Not used.   |

Figure 18-7. Timer Functionality



**Notes:**

- The timer functionality uses only PERIOD (and not PERIOD\_BUFF).
- Do not write to COUNTER when the counter is running.

Figure 18-8 illustrates a timer in up-counting mode. The counter is initialized (to 0) and started with a software-based reload event.

**Notes:**

- PERIOD is 4, resulting in an effective repeating counter pattern of  $4+1 = 5$  clk\_counter periods. The CC register is 2, and sets the condition for a cc\_match event.
- When the counter changes from a state in which COUNTER is 4, overflow and tc events are generated.
- When the counter changes from a state in which COUNTER is 2, a cc\_match event is generated.
- A constant count event of '1' and clk\_counter without prescaling is used in the following scenarios. If the count event is '0' and a reload event is triggered, the reload will be registered only on the first clock edge when the count event is '1'. This means that the first clock edge when the count event is '1' will not be used for counting. It will be used for reload.

Figure 18-8. Timer in Up-counting Mode

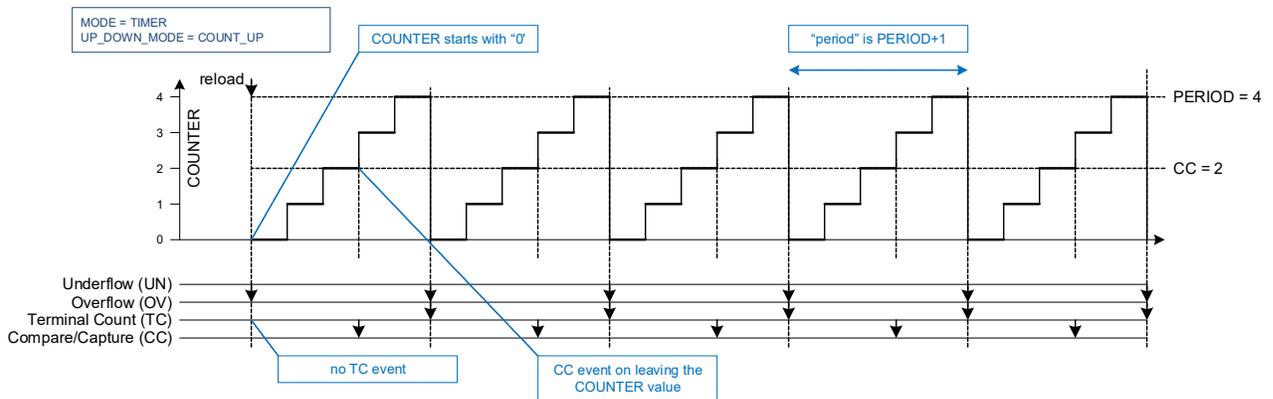


Figure 18-9 illustrates a timer in "one-shot" operation mode. Note that the counter is stopped on a tc event.

Figure 18-9. Timer in One-shot Mode

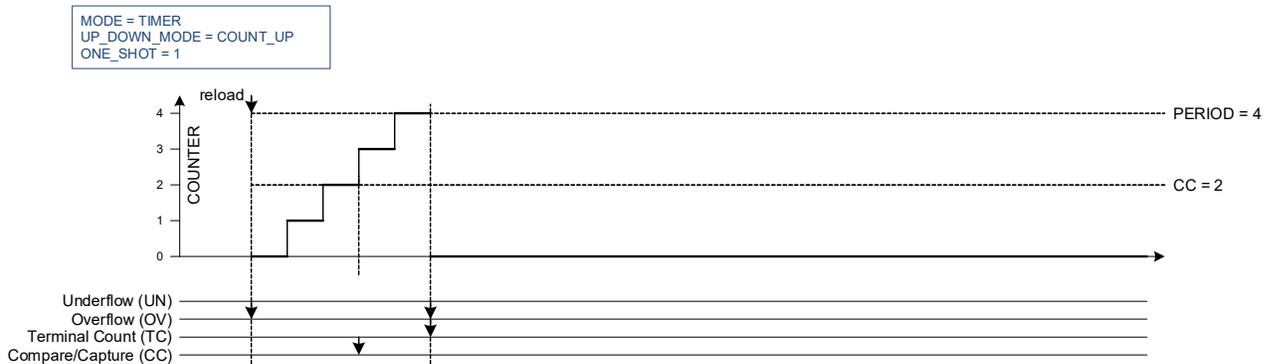


Figure 18-10 illustrates clock pre-scaling. Note that the counter is only incremented every other counter cycle.

Figure 18-10. Timer Clock Pre-scaling

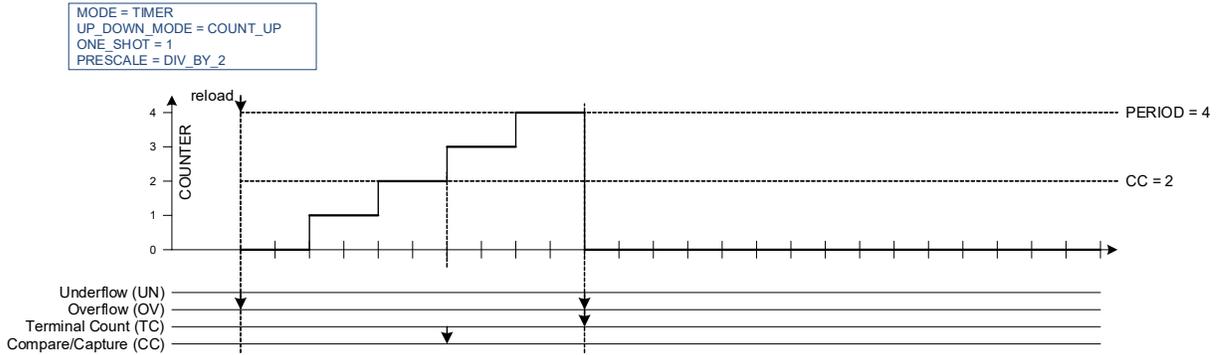


Figure 18-11 illustrates a counter that is initialized and started (reload event), stopped (stop event), and continued/started (start event). Note that the counter does not change value when it is not running (STATUS.RUNNING).

Figure 18-11. Counter Start/Stopped/Continued

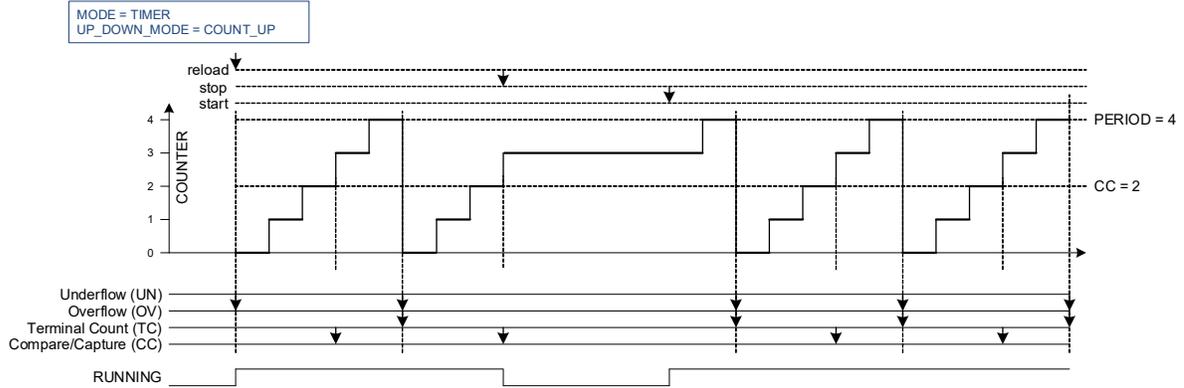


Figure 18-12 illustrates a timer that uses both CC and CC\_BUFF registers. Note that CC and CC\_BUFF are exchanged on a cc\_match event.

Figure 18-12. Use of CC and CC\_BUFF Register Bits

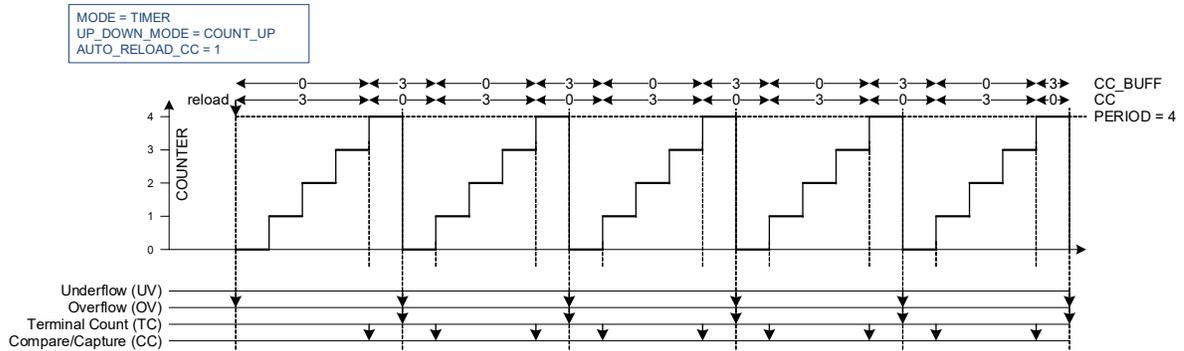


Figure 18-13 illustrates a timer in down counting mode. The counter is initialized (to PERIOD) and started with a software-based reload event.

**Notes:**

- When the counter changes from a state in which COUNTER is 0, a UN and TC events are generated.
- When the counter changes from a state in which COUNTER is 2, a cc\_match event is generated.
- PERIOD is 4, resulting in an effective repeating counter pattern of  $4+1 = 5$  counter clock periods.

Figure 18-13. Timer in Down-counting Mode

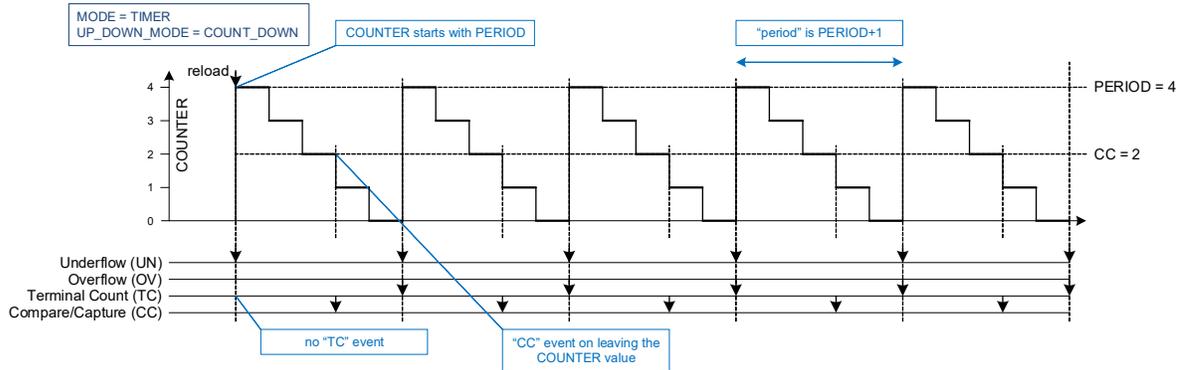


Figure 18-14 illustrates a timer in up/down counting mode 1. The counter is initialized (to 1) and started with a software-based reload event.

**Notes:**

- When the counter changes from a state in which COUNTER is 4, an overflow is generated.
- When the counter changes from a state in which COUNTER is 0, an underflow and tc event are generated.
- When the counter changes from a state in which COUNTER is 2, a cc\_match event is generated.
- PERIOD is 4, resulting in an effective repeating counter pattern of  $2*4 = 8$  counter clock periods.

Figure 18-14. Timer in Up/Down Counting Mode 1

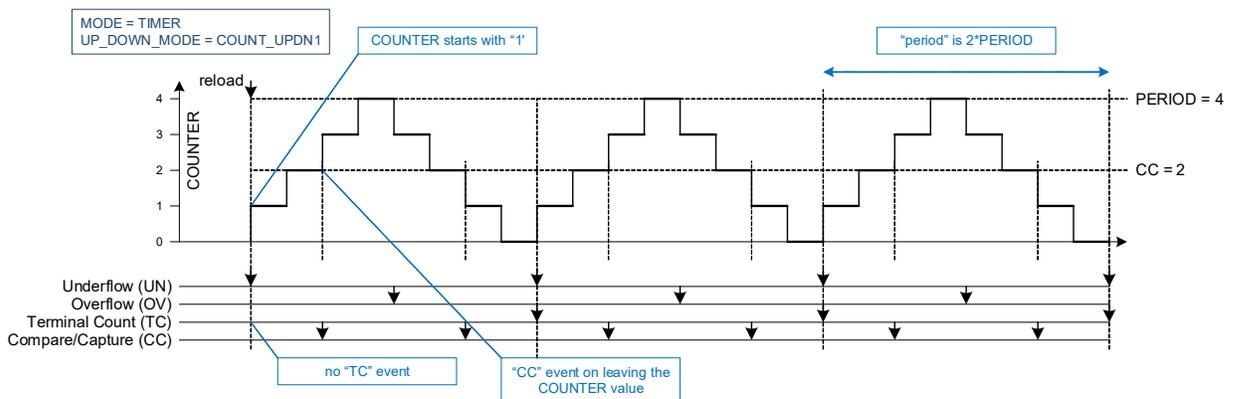


Figure 18-15 illustrates a timer in up/down counting mode 1, with different CC values.

**Notes:**

- When CC is 0, the cc\_match event is generated at the start of the period (when the counter changes from a state in which COUNTER is 0).
- When CC is PERIOD, the cc\_match event is generated at the middle of the period (when the counter changes from a state in which COUNTER is PERIOD).

Figure 18-15. Up/Down Counting Mode with Different CC Values

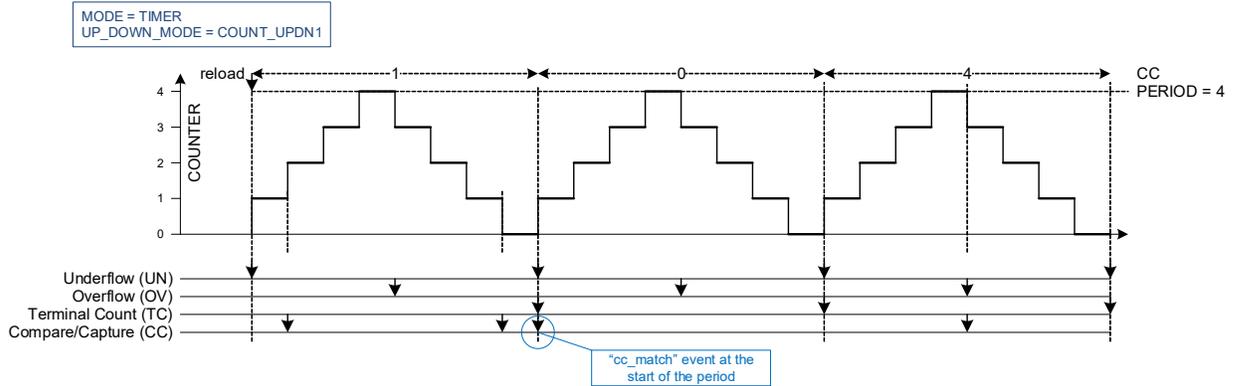
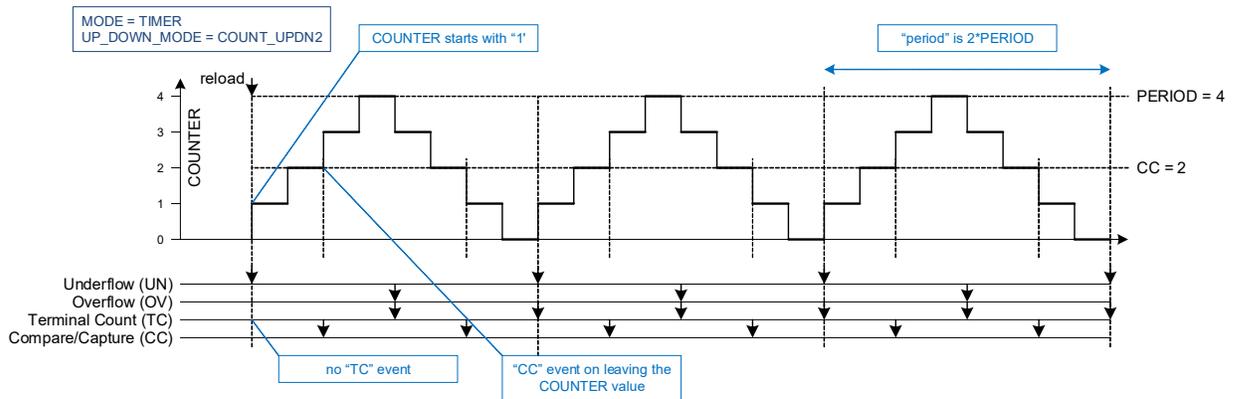


Figure 18-16 illustrates a timer in up/down counting mode 2. This mode is same as up/down counting mode 1, except for the TC event, which is generated when either underflow or overflow event occurs.

Figure 18-16. Up/Down Counting Mode 2



### 18.3.1.1 Configuring Counter for Timer Mode

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

1. Disable the counter by writing '0' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM\_CNT\_CTRL register.
3. Set the required 16-bit period in the TCPWM\_CNT\_PERIOD register.
4. Set the 16-bit compare value in the TCPWM\_CNT\_CC register and the buffer compare value in the TCPWM\_CNT\_C-C\_BUFF register.
5. Set AUTO\_RELOAD\_CC field of the TCPWM\_CNT\_CTRL register, if required to swap values at every CC condition.
6. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM\_CNT\_CTRL register.
7. Set the direction of counting by writing to the UP\_DOWN\_MODE[17:16] field of the TCPWM\_CNT\_CTRL register.
8. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE\_SHOT[18] field of the TCPWM\_CNT\_CTRL register.
9. Set the TCPWM\_CNT\_TR\_CTRL0 register to select the trigger that causes the event (reload, start, stop, capture, and count).
10. Set the TCPWM\_CNT\_TR\_CTRL1 register to select the edge of the trigger that causes the event (reload, start, stop, capture, and count).
11. If required, set the interrupt upon TC or CC condition.
12. Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A reload trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware reload signal is not enabled.

### 18.3.2 Capture Mode

The capture functionality increments/decrements a counter between 0 and PERIOD. When the capture event is activated the counter value COUNTER is copied to CC (and CC is copied to CC\_BUFF).

The capture functionality can be used to measure the width of a pulse (connected as one of the input triggers and used as capture event).

The capture event can be triggered through the capture trigger input or through a firmware write to command register (COUNTER\_CAPTURE[4:0] field of the TCPWM\_CMD register).

Table 18-11. Capture Mode Trigger Input Description

| Trigger Inputs | Usage   |
|----------------|---|
| reload         | Sets the counter value and starts the counter. Behavior is dependent on UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The counter is set to "0" and count direction is set to "up".</li> <li>■ COUNT_DOWN: The counter is set to PERIOD and count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The counter is set to "1" and count direction is set to "up".</li> </ul> Can be used only when the counter is not running. |
| start          | Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The count direction is set to "up".</li> <li>■ COUNT_DOWN: The count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The count direction is not modified.</li> </ul> Can be used only when the counter is not running.                             |
| stop           | Stops the counter.  |
| count          | Count event increments/decrements the counter.  |
| capture        | Copies the counter value to CC and copies CC to CC_BUFF.  |

Table 18-12. Capture Mode Supported Features

| Supported Features | Description  |
|--------------------|--|
| Clock pre-scaling  | Pre-scales the counter clock <code>clk_counter</code> .  |
| One-shot           | Counter is stopped by hardware, after a single period of the counter: <ul style="list-style-type: none"> <li>■ <code>COUNT_UP</code>: on an overflow event.</li> <li>■ <code>COUNT_DOWN</code>, <code>COUNT_UPDN1/2</code>: on an underflow event.</li> </ul>  |
| Up/down modes      | Specified by <code>UP_DOWN_MODE</code> : <ul style="list-style-type: none"> <li>■ <code>COUNT_UP</code>: The counter counts from 0 to <code>PERIOD</code>.</li> <li>■ <code>COUNT_DOWN</code>: The counter counts from <code>PERIOD</code> to 0.</li> <li>■ <code>COUNT_UPDN1/2</code>: The counter counts from 1 to <code>PERIOD</code> and back to 0.</li> </ul> |

Table 18-13. Capture Mode Trigger Output Description

| Trigger Outputs            | Description   |
|----------------------------|---|
| <code>cc_match (CC)</code> | <code>CC</code> is copied to <code>CC_BUFF</code> and counter value is copied to <code>CC</code> ( <code>cc_match</code> equals capture event). |
| Underflow (UN)             | Counter is decrementing and changes from a state in which <code>COUNTER</code> equals "0".  |
| Overflow (OV)              | Counter is incrementing and changes from a state in which <code>COUNTER</code> equals <code>PERIOD</code> .                                     |

Table 18-14. Capture Mode Interrupt Outputs

| Interrupt Outputs          | Description  |
|----------------------------|--|
| <code>tc</code>            | Specified by <code>UP_DOWN_MODE</code> : <ul style="list-style-type: none"> <li>■ <code>COUNT_UP</code>: <code>tc</code> event is the same as the overflow event.</li> <li>■ <code>COUNT_DOWN</code>: <code>tc</code> event is the same as the underflow event.</li> <li>■ <code>COUNT_UPDN1</code>: <code>tc</code> event is the same as the underflow event.</li> <li>■ <code>COUNT_UPDN2</code>: <code>tc</code> event is the same as the logical OR of the overflow and underflow events.</li> </ul> |
| <code>cc_match (CC)</code> | <code>CC</code> is copied to <code>CC_BUFF</code> and counter value is copied to <code>CC</code> ( <code>cc_match</code> equals capture event).  |

Table 18-15. Capture Mode PWM Outputs

| PWM Outputs                 | Description |
|-----------------------------|-------------|
| <code>line_out</code>       | Not used.   |
| <code>line_compl_out</code> | Not used.   |

Figure 18-17. Capture Functionality

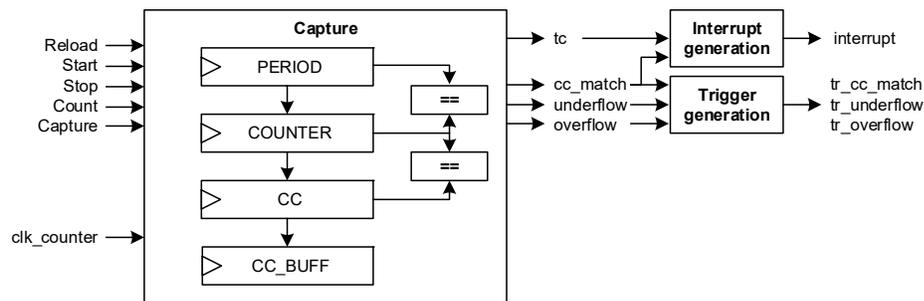
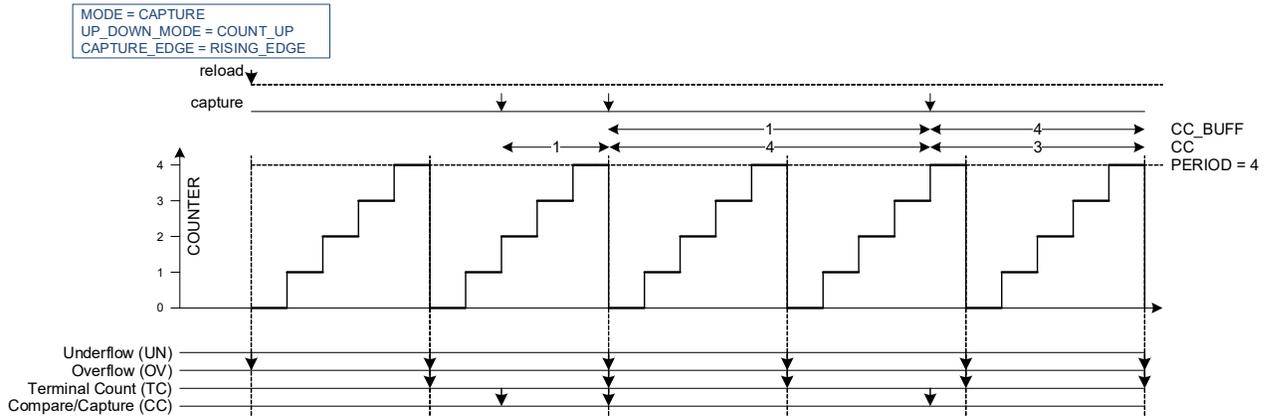


Figure 18-18 illustrates capture behavior in the up counting mode.

**Notes:**

- The capture event detection uses rising edge detection. As a result, the capture event is remembered until the next “active count” pre-scaled counter clock.
- When a capture event occurs, COUNTER is copied into CC. CC is copied to CC\_BUFF.
- A cc\_match event is generated when the counter value is captured.

Figure 18-18. Capture in Up Counting Mode

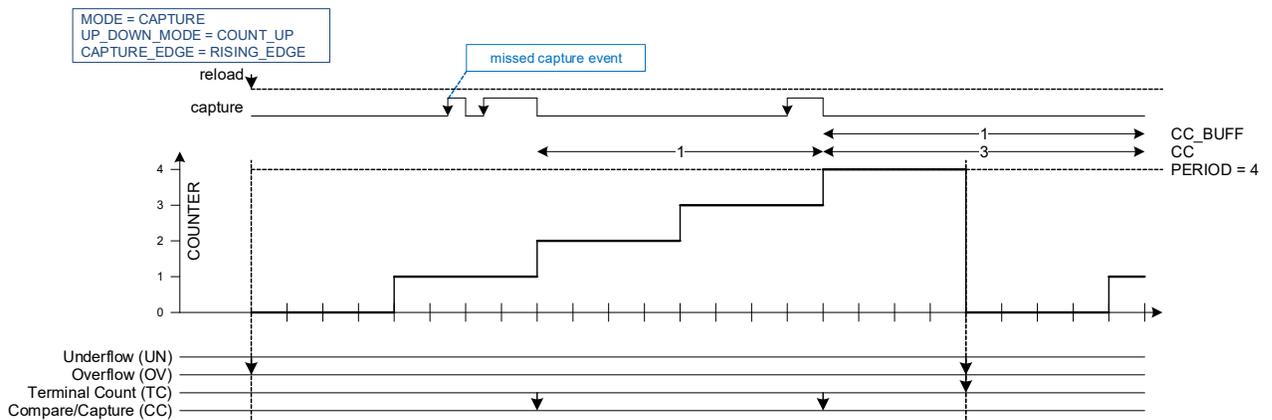


When multiple capture events are detected before the next “active count” pre-scaled counter clock, capture events are treated as follows:

- In the rising edge and falling edge modes, multiple events are effectively reduced to a single event.
- In the rising/falling edge mode, an even number of events is not detected and an odd number of events is reduced to a single event.

This behavior is illustrated by Figure 18-19, in which a pre-scaler by a factor of 4 is used.

Figure 18-19. Multiple Events Detected before Active-Count



### 18.3.2.1 Configuring Counter for Capture Mode

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

1. Disable the counter by writing '0' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM\_CNT\_CTRL register.
3. Set the required 16-bit period in the TCPWM\_CNT\_PERIOD register.
4. Set clock prescaling by writing to the GENERIC[15:8] field of the TCPWM\_CNT\_CTRL register.
5. Set the direction of counting by writing to the UP\_DOWN\_MODE[17:16] field of the TCPWM\_CNT\_CTRL register.
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.
10. Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A reload trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware reload signal is not enabled.

### 18.3.3 Quadrature Decoder Mode

Quadrature functionality increments and decrements a counter between 0 and 0xFFFF. Counter updates are under control of quadrature signal inputs: index, phiA, and phiB. The index input is used to indicate an absolute position. The phiA and phiB inputs are used to determine a change in position (the rate of change in position can be used to derive speed). The quadrature inputs are mapped onto triggers (as described in [Table 18-16](#)).

Table 18-16. Quadrature Mode Trigger Input Description

| Trigger Input | Usage  |
|---------------|--|
| reload/index  | This event acts as a quadrature index input. It initializes the counter to the counter midpoint 0x8000 and starts the quadrature functionality. Rising edge event detection or falling edge detection mode must be used. |
| start/phiB    | This event acts as a quadrature phiB input. Pass through (no edge detection) event detection mode must be used.  |
| stop          | Stops the quadrature functionality.  |
| count/phiA    | This event acts as a quadrature phiA input. Pass through (no edge detection) event detection mode must be used.  |
| capture       | Not used.  |

Table 18-17. Quadrature Mode Supported Features

| Supported Features  | Description   |
|---------------------|---|
| Quadrature encoding | Three encoding schemes for the phiA and phiB inputs are supported (as specified by CTRL.QUADRATURE_MODE):<br>X1 encoding.<br>X2 encoding.<br>X4 encoding. |

**Note:** Clock pre-scaling is not supported and the count event is used as a quadrature input phiA. As a result, the quadrature functionality operates on the counter clock (clk\_counter), rather than on an "active count" prescaled counter clock.

Table 18-18. Quadrature Mode Trigger Output Description

| Trigger Outputs | Description   |
|-----------------|---|
| cc_match (CC)   | Counter value COUNTER equals 0 or 0xFFFF or a reload/index event. |
| Underflow (UN)  | Not used.   |
| Overflow (OV)   | Not used.   |

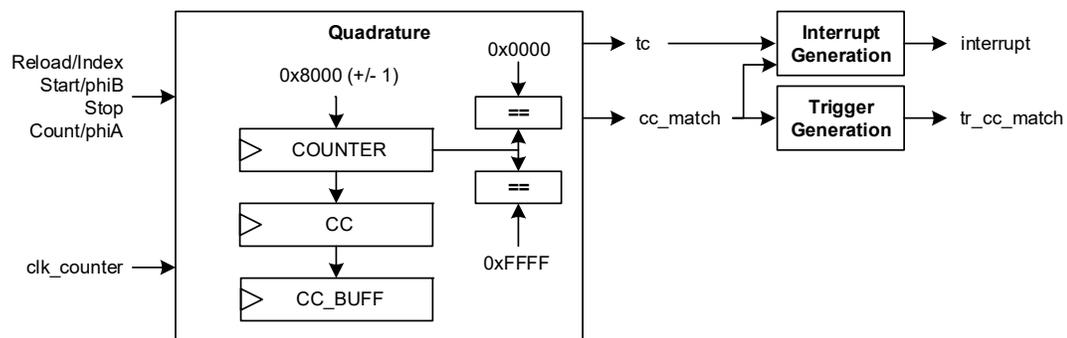
Table 18-19. Quadrature Mode Interrupt Outputs

| Interrupt Outputs | Description   |
|-------------------|---|
| cc_match (CC)     | Counter value COUNTER equals 0 or 0xFFFF or a reload/index event. |
| tc                | Reload/index event.   |

Table 18-20. Quadrature Mode PWM Outputs

| PWM Outputs    | Description |
|----------------|-------------|
| line_out       | Not used.   |
| line_compl_out | Not used.   |

Figure 18-20. Quadrature Functionality (16-bit Example)



Quadrature functionality is described as follows:

- A software-generated reload event starts quadrature operation. As a result, COUNTER is set to 0x8000 (16-bit), which is the counter midpoint (the COUNTER is set to 0x7FFF if the reload event coincides with a decrement event; the COUNTER is set to 0x8001 if the reload event coincides with an increment event). Note that a software-generated reload event is generated only once, when the counter is not running. All other reload/index events are hardware-generated reload events as a result of the quadrature index signal.
- During quadrature operation:
  - The counter value COUNTER is incremented or decremented based on the specified quadrature encoding scheme.
  - On a reload/index event, CC is copied to CC\_BUFF, COUNTER is copied to CC, and COUNTER is set to 0x8000. In addition, the tc and cc\_match events are generated.
  - When the counter value COUNTER is 0x0000, CC is copied to CC\_BUFF, COUNTER (0x0000) is copied to CC, and COUNTER is set to 0x8000. In addition, the cc\_match event is generated.
  - When the counter value COUNTER is 0xFFFF, CC is copied to CC\_BUFF, COUNTER (0xFFFF) is copied to CC, and COUNTER is set to 0x8000. In addition, the cc\_match event is generated.

**Note:** When the counter reaches 0x0000 or 0xFFFF, the counter is automatically set to 0x8000 without an increase or decrease event.

The software interrupt handler uses the tc and cc\_match interrupt cause fields to distinguish between a reload/index event and a situation in which a minimum/maximum counter value was reached (about to wrap around). The CC and CC\_BUFF registers are used to determine when the interrupt causing event occurred.

Note that a counter increment/decrement can coincide with a reload/index/tc event or with a situation cc\_match event. Under these circumstances, the counter value set to either 0x8000+1 (increment) or 0x8000-1 (decrement).

Counter increments (incr1 event) and decrements (decr1 event) are determined by the quadrature encoding scheme as illustrated by [Figure 18-21](#).

Figure 18-21. Quadrature Mode Waveforms

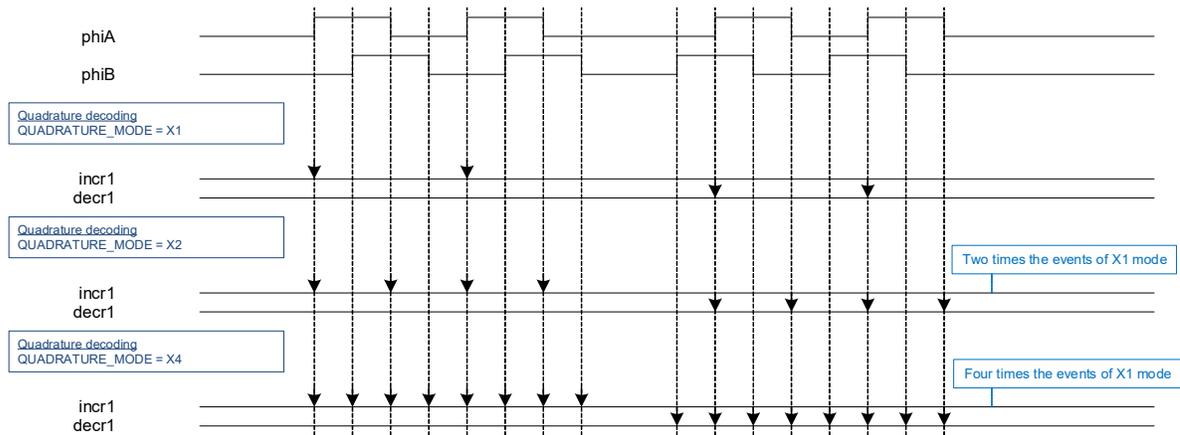


Figure 18-22 illustrates quadrature functionality as a function of the reload/index, incr1, and decr1 events. Note that the first reload/index event copies the counter value COUNTER to CC.

Figure 18-22. Quadrature Mode Reload/Index Timing

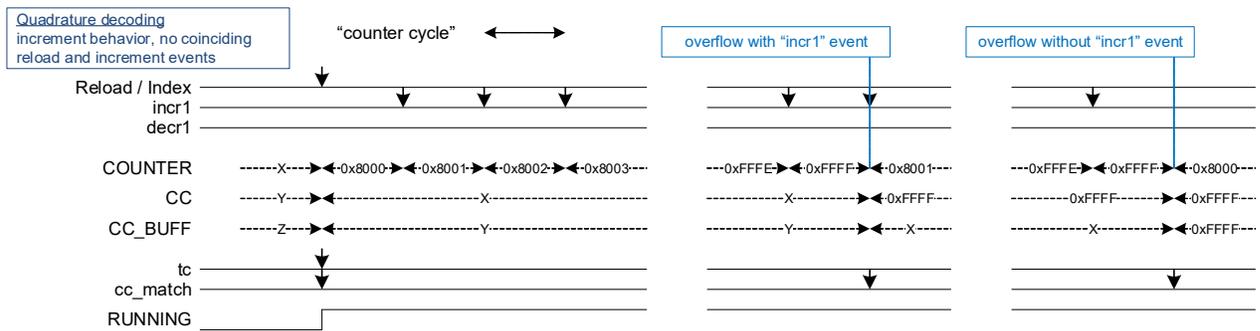
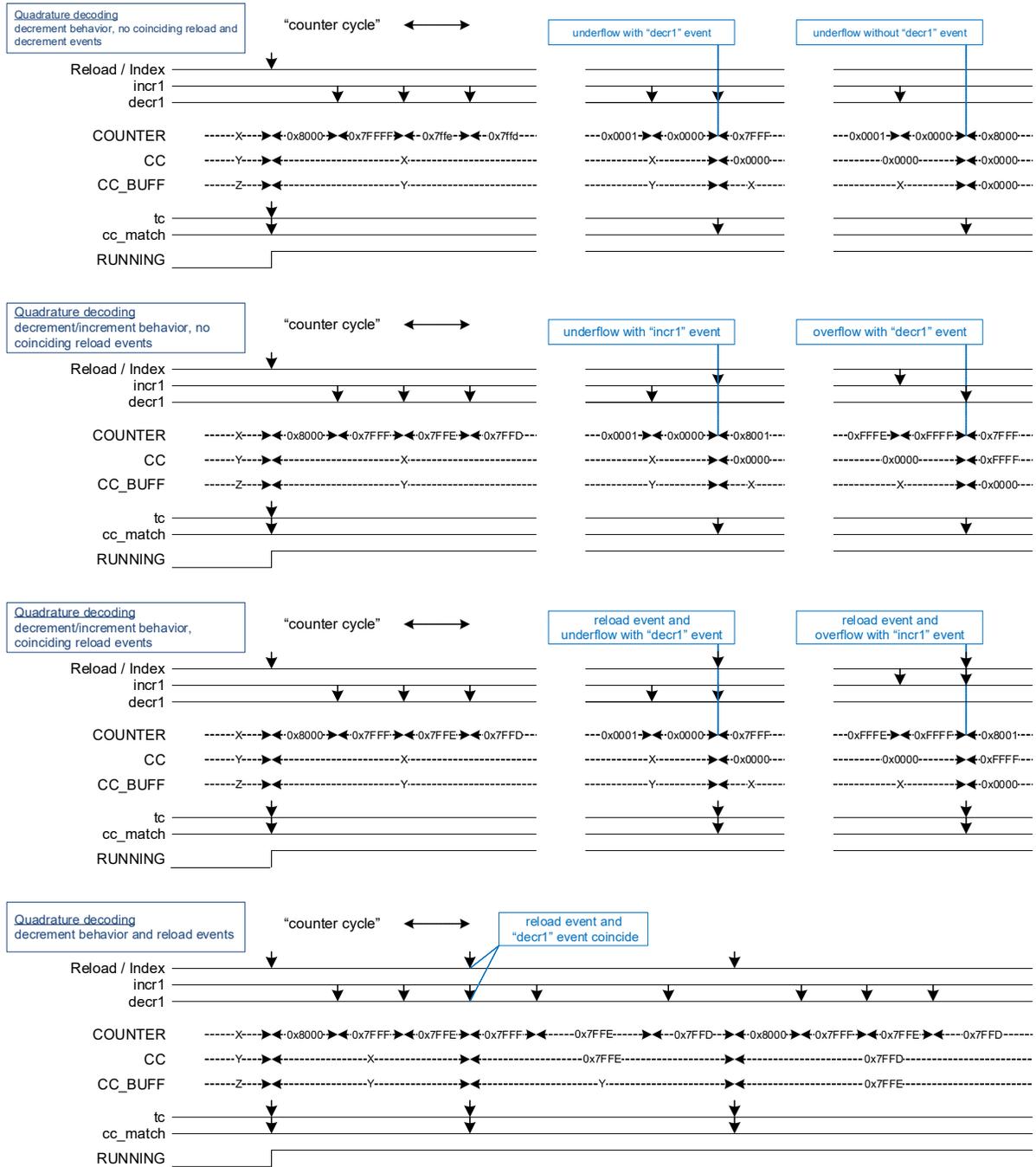


Figure 18-23 illustrate quadrature functionality for different event scenarios (including scenarios with coinciding events). In all scenarios, the first reload/index event is generated by software when the counter is not yet running.

Figure 18-23. Quadrature Mode Timing Cases



### 18.3.3.1 Configuring Counter for Quadrature Mode

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

1. Disable the counter by writing '0' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM\_CNT\_CTRL register.
3. Set the required encoding mode by writing to the QUADRATURE\_MODE[21:20] field of the TCPWM\_CNT\_CTRL register.
4. Set the TCPWM\_CNT\_TR\_CTRL0 register to select the trigger that causes the event (Index and Stop).
5. Set the TCPWM\_CNT\_TR\_CTRL1 register to select the edge that causes the event (Index and Stop).
6. If required, set the interrupt upon TC or CC condition.
7. Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A reload trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware reload signal is not enabled.

### 18.3.4 Pulse Width Modulation Mode

The PWM can output left, right, center, or asymmetrically-aligned PWM. The PWM signal is generated by incrementing or decrementing a counter between 0 and PERIOD, and comparing the counter value COUNTER with CC. When COUNTER equals CC, the cc\_match event is generated. The pulse-width modulated signal is then generated by using the cc\_match event along with overflow and underflow events. Two pulse-width modulated signals "line\_out" and "line\_compl\_out" are output from the PWM.

Table 18-21. PWM Mode Trigger Input Description

| Trigger Inputs | Usage   |
|----------------|---|
| reload         | Sets the counter value and starts the counter. Behavior is dependent on UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The counter is set to "0" and count direction is set to "up".</li> <li>■ COUNT_DOWN: The counter is set to PERIOD and count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The counter is set to "1" and count direction is set to "up".</li> </ul> Can be used only when the counter is not running.   |
| start          | Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The count direction is set to "up".</li> <li>■ COUNT_DOWN: The count direction is set to "down".</li> <li>■ COUNT_UPDN1/2: The count direction is set to "up".</li> </ul> Can be used only when the counter is not running.  |
| stop/kill      | Stops the counter or suppresses the PWM output, depending on PWM_STOP_ON_KILL and PWM_SYNC_KILL.  |
| count          | Count event increments/decrements the counter.  |
| capture/swap   | This event acts as a swap event. When this event is active, the CC/CC_BUFF and PERIOD/PERIOD_BUFF registers are exchanged on a tc event (when specified by CTRL.AUTO_RELOAD_CC and CTRL.AUTO_RELOAD_PERIOD). A swap event requires rising, falling, or rising/falling edge event detection mode. Pass-through mode is not supported, unless the selected event is a constant '0' or '1'. <p><b>Note:</b> When COUNT_UPDN2 mode exchanges PERIOD and PERIOD_BUFF at a TC event that coincides with an OV event, software should ensure that the PERIOD and PERIOD_BUFF values are the same.</p> When a swap event is detected and the counter is running, the event is kept pending until the next tc event. When a swap event is detected and the counter is not running, the event is cleared by hardware. |

Table 18-22. PWM Mode Supported Features

| Supported Features        | Description  |
|---------------------------|--|
| Clock pre-scaling         | Pre-scales the counter clock "clk_counter".  |
| One-shot                  | Counter is stopped by hardware, after a single period of the counter: <ul style="list-style-type: none"> <li>■ COUNT_UP: on an overflow event.</li> <li>■ COUNT_DOWN and COUNT_UPDN1/2: on an underflow event.</li> </ul>  |
| Compare Swap              | CC and CC_BUFF are exchanged on a swap event and tc event (when specified by CTRL.AUTO_RELOAD_CC).   |
| Period Swap               | PERIOD and PERIOD_BUFF are exchanged on a swap event and tc event (when specified by CTRL.AUTO_RELOAD_PERIOD). <p><b>Note:</b> When COUNT_UPDN2/Asymmetric mode exchanges PERIOD and PERIOD_BUFF at a tc event that coincides with an overflow event, software should ensure that the PERIOD and PERIOD_BUFF values are the same.</p>  |
| Alignment (Up/Down modes) | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: The counter counts from 0 to PERIOD. Generates a left-aligned PWM output.</li> <li>■ COUNT_DOWN: The counter counts from PERIOD to 0. Generates a right-aligned PWM output.</li> <li>■ COUNT_UPDN1/2: The counter counts from 1 to PERIOD and back to 0. Generates a center-aligned/asymmetric PWM output.</li> </ul>  |
| Kill modes                | Specified by PWM_STOP_ON_KILL and PWM_SYNC_KILL: <ul style="list-style-type: none"> <li>■ PWM_STOP_ON_KILL = '1' (PWM_SYNC_KILL = don't care): Stop on Kill mode. This mode stops the counter on a stop/kill event. Reload or start event is required to restart counting.</li> <li>■ PWM_STOP_ON_KILL = '0' and PWM_SYNC_KILL = '0': Asynchronous kill mode. This mode keeps the counter running, but suppresses the PWM output signals and continues to do so for the duration of the stop/kill event.</li> <li>■ PWM_STOP_ON_KILL = '0' and PWM_SYNC_KILL = '1': Synchronous kill mode. This mode keeps the counter running, but suppresses the PWM output signals and continues to do so until the next tc event without a stop/kill event.</li> </ul> |

Note that the PWM mode does not support dead time insertion. This functionality is supported by the separate PWM\_DT mode.

Table 18-23. PWM Mode Trigger Output Description

| Trigger Output | Description   |
|----------------|---|
| cc_match (CC)  | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP and COUNT_DOWN: The counter changes to a state in which COUNTER equals CC.</li> <li>■ COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC.</li> </ul> |
| Underflow (UN) | Counter is decrementing and changes from a state in which COUNTER equals "0".   |
| Overflow (OV)  | Counter is incrementing and changes from a state in which COUNTER equals PERIOD.  |

Table 18-24. PWM Mode Interrupt Output Description

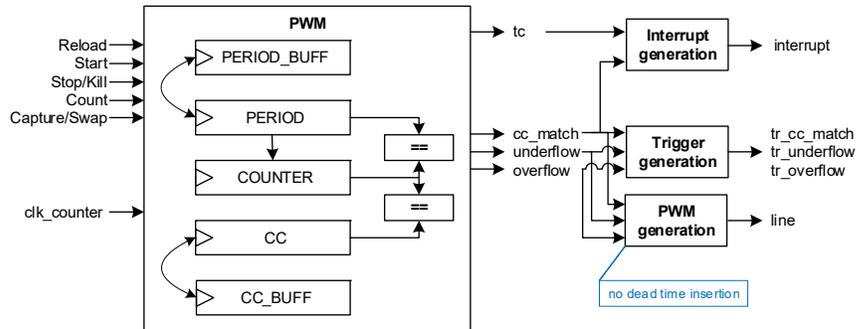
| Interrupt Outputs | Description  |
|-------------------|--|
| tc                | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP: tc event is the same as the overflow event.</li> <li>■ COUNT_DOWN: tc event is the same as the underflow event.</li> <li>■ COUNT_UPDN1: tc event is the same as the underflow event.</li> <li>■ COUNT_UPDN2: tc event is the same as the logical OR of the overflow and underflow events.</li> </ul> |
| cc_match (CC)     | Specified by UP_DOWN_MODE: <ul style="list-style-type: none"> <li>■ COUNT_UP and COUNT_DOWN: The counter changes to a state in which COUNTER equals CC.</li> <li>■ COUNT_UPDN1/2: counter changes from a state in which COUNTER equals CC.</li> </ul>  |

Table 18-25. PWM Mode PWM Outputs

| PWM Outputs    | Description               |
|----------------|---------------------------|
| line_out       | PWM output.               |
| line_compl_out | Complementary PWM output. |

Note that the cc\_match event generation in COUNT\_UP and COUNT\_DOWN modes are different from the generation in other functional modes or counting modes. This is to ensure that 0 percent and 100 percent duty cycles can be generated.

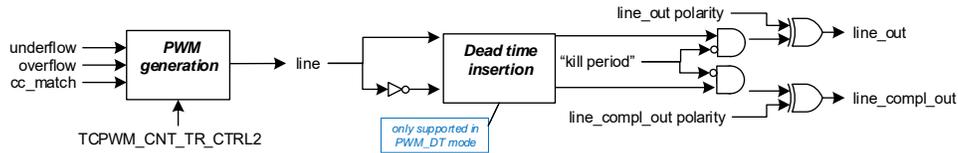
Figure 18-24. PWM Mode Functionality



The generation of PWM output signals is a multi-step process and is illustrated in Figure 18-25. The PWM output signals are generated by using the underflow, overflow, and cc\_match events. Each of these events can be individually set to INVERT, SET, or CLEAR line.

**Note:** An underflow and cc\_match or an overflow and cc\_match can occur at the same time. When this happens, underflow and overflow events take priority over cc\_match. For example, if overflow = SET and cc\_match = CLEAR then line will be SET to '1' first and then CLEARED to '0' immediately after. This can be seen in Figure 18-27.

Figure 18-25. PWM Output Generation



line\_out polarity and line\_compl\_out polarity as seen in Figure 18-25, allow the PWM outputs to be inverted. line\_out polarity is controlled through CTRL.QUADRATURE\_MODE[0] and line\_compl\_out polarity is controlled through CTRL.QUADRATURE\_MODE[1].

PWM behavior depends on the PERIOD and CC registers. The software can update the PERIOD\_BUFF and CC\_BUFF registers, without affecting the PWM behavior. This is the main rationale for double buffering these registers.

Figure 18-26 illustrates a PWM in up counting mode. The counter is initialized (to 0) and started with a software-based reload event.

**Notes:**

- When the counter changes from a state in which COUNTER is 4, an overflow and tc event are generated.
- When the counter changes to a state in which COUNTER is 2, a cc\_match event is generated.
- PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods.

Figure 18-26. PWM in Up Counting Mode

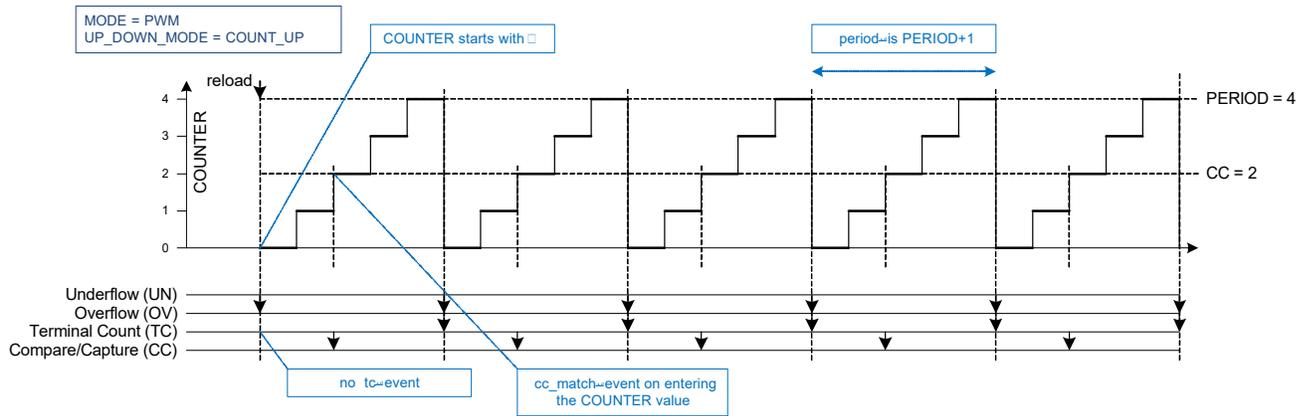


Figure 18-27 illustrates a PWM in up counting mode generating a left-aligned PWM. The figure also illustrates how a right-aligned PWM can be created using the PWM in up counting mode by inverting the OVERFLOW\_MODE and CC\_MATCH\_MODE and using a CC value that is complementary ( $PERIOD+1 - \text{pulse width}$ ) to the one used for left-aligned PWM. Note that CC is changed (to CC\_BUFF, which is not depicted) on a tc event. The duty cycle is controlled by setting the CC value.  $CC = \text{desired duty cycle} \times (PERIOD+1)$ .

Figure 18-27. PWM Left- and Right-Aligned Outputs

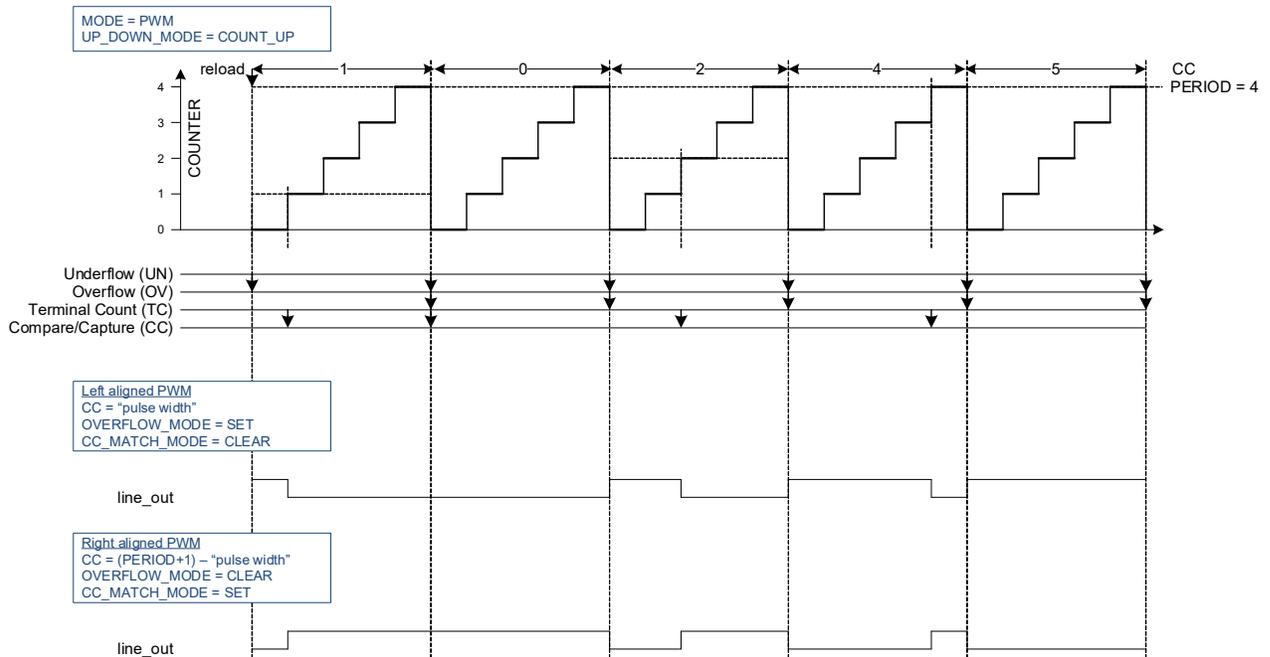


Figure 18-28 illustrates a PWM in down counting mode. The counter is initialized (to PERIOD) and started with a software-based reload event.

**Notes:**

- When the counter changes from a state in which COUNTER is 0, an underflow and tc event are generated.
- When the counter changes to a state in which COUNTER is 2, a cc\_match event is generated.
- PERIOD is 4, resulting in an effective repeating counter pattern of 4+1 = 5 counter clock periods.

Figure 18-28. PWM in Down Counting Mode

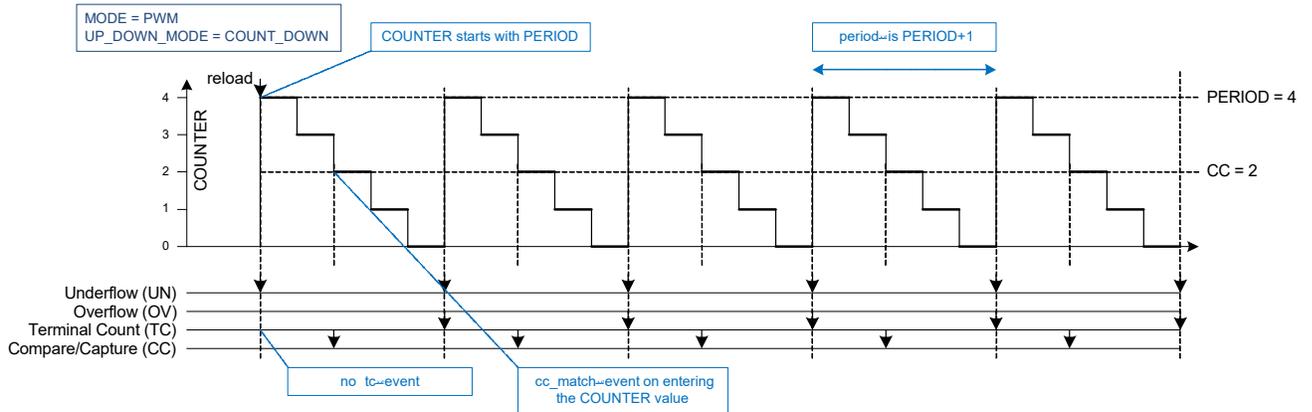


Figure 18-29 illustrates a PWM in down counting mode with different CC values. The figure also illustrates how a right-aligned PWM can be creating using the PWM in down counting mode. Note that the CC is changed (to CC\_BUFF, which is not depicted) on a tc event.

Figure 18-29. Right- and Left-Aligned Down Counting PWM

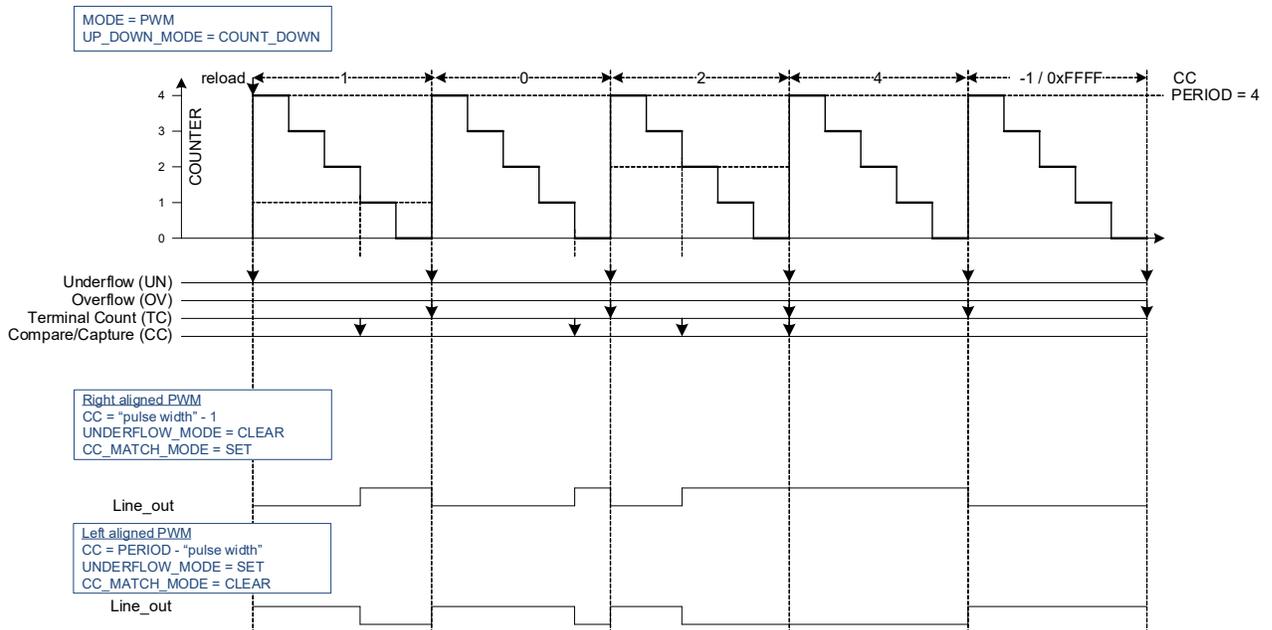


Figure 18-30 illustrates a PWM in up/down counting mode. The counter is initialized (to 1) and started with a software-based reload event.

**Notes:**

- When the counter changes from a state in which COUNTER is 4, an overflow is generated.
- When the counter changes from a state in which COUNTER is 0, an underflow and tc event are generated.
- When the counter changes from a state in which COUNTER is 2, a cc\_match event is generated. Note that the actual counter value COUNTER from before the reload event is NOT used, instead the counter value before the reload event is considered to be 0.
- PERIOD is 4, resulting in an effective repeating counter pattern of  $2 * 4 = 8$  counter clock periods.

Figure 18-30. Up/Down Counting PWM

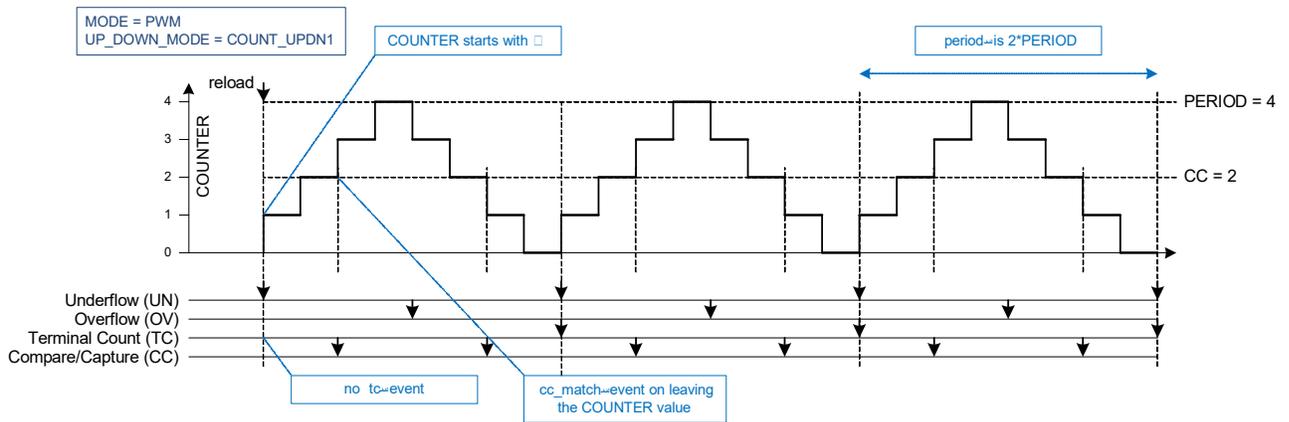
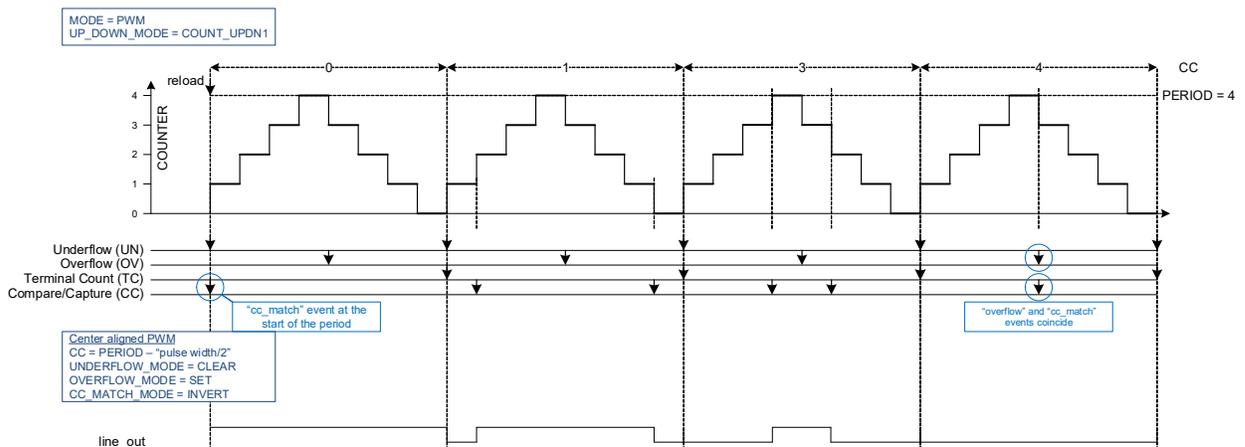


Figure 18-31 illustrates a PWM in up/down counting mode with different CC values. The figure also illustrates how a center-aligned PWM can be creating using the PWM in up/down counting mode.

**Note:**

- The actual counter value COUNTER from before the reload event is NOT used. Instead the counter value before the reload event is considered to be 0. As a result, when the first CC value at the reload event is 0, a cc\_match event is generated.
- CC is changed (to CC\_BUFF, which is not depicted) on a tc event.

Figure 18-31. Up/Down Counting Center-Aligned PWM



Different stop/kill modes exist. The mode is specified by PWM\_STOP\_ON\_KILL and PWM\_SYNC\_KILL.

The following three modes are supported:

- PWM\_STOP\_ON\_KILL is '1' (PWM\_SYNC\_KILL is don't care): Stop on Kill mode. This mode stops the counter on a stop/kill event. Reload or start event is required to restart the counter. Both software and external trigger input can be selected as stop kill. Edge detection mode is required.
- PWM\_STOP\_ON\_KILL is '0' and PWM\_SYNC\_KILL is '0': Asynchronous Kill mode. This mode keeps the counter running, but suppresses the PWM output signals synchronously on the next count clock ("active count" pre-scaled clk\_counter) and continues to do so for the duration of the stop/kill event. Only the external trigger input can be selected as asynchronous kill. Pass through detection mode is required.
- PWM\_STOP\_ON\_KILL is '0' and PWM\_SYNC\_KILL is '1': Synchronous Kill mode. This mode keeps the counter running, but suppresses the PWM output signals synchronously on the next count clock ("active count" pre-scaled clk\_counter) and continues to do so until the next tc event without a stop/kill event. Only the external trigger input can be selected as synchronous kill. Rising edge detection mode is required.

Figure 18-32, Figure 18-33, and Figure 18-34 illustrate the above three modes.

Figure 18-32. PWM Stop on Kill

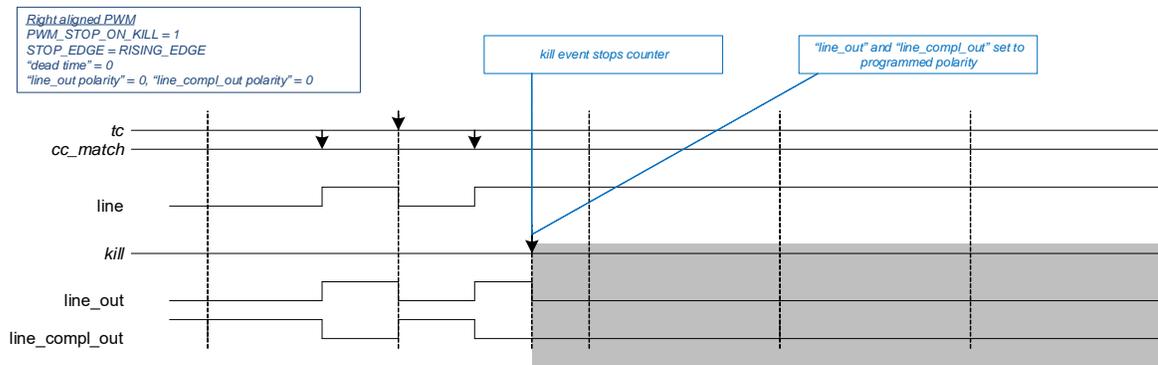


Figure 18-33. PWM Async Kill

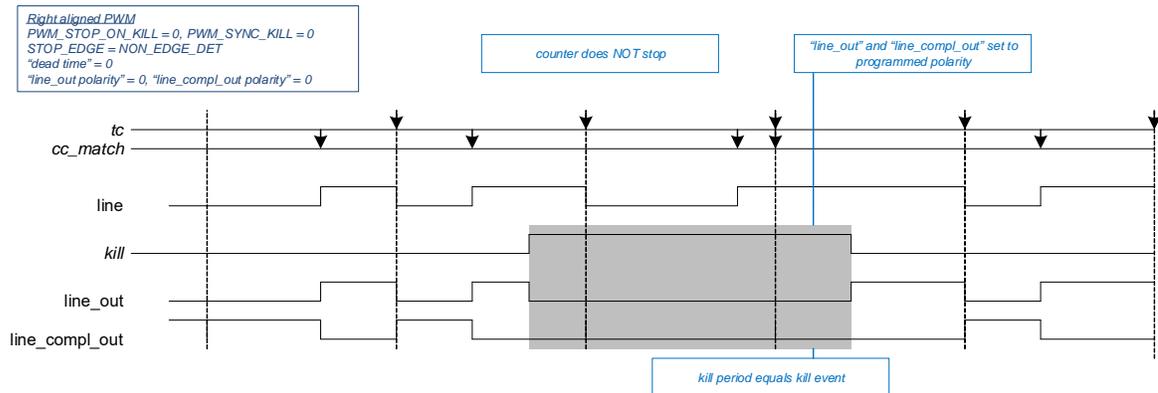


Figure 18-34. PWM Sync Kill

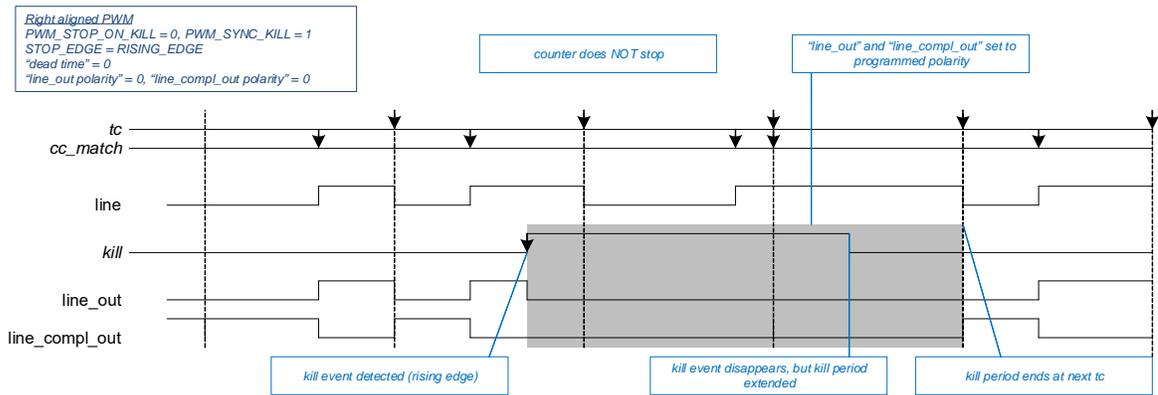
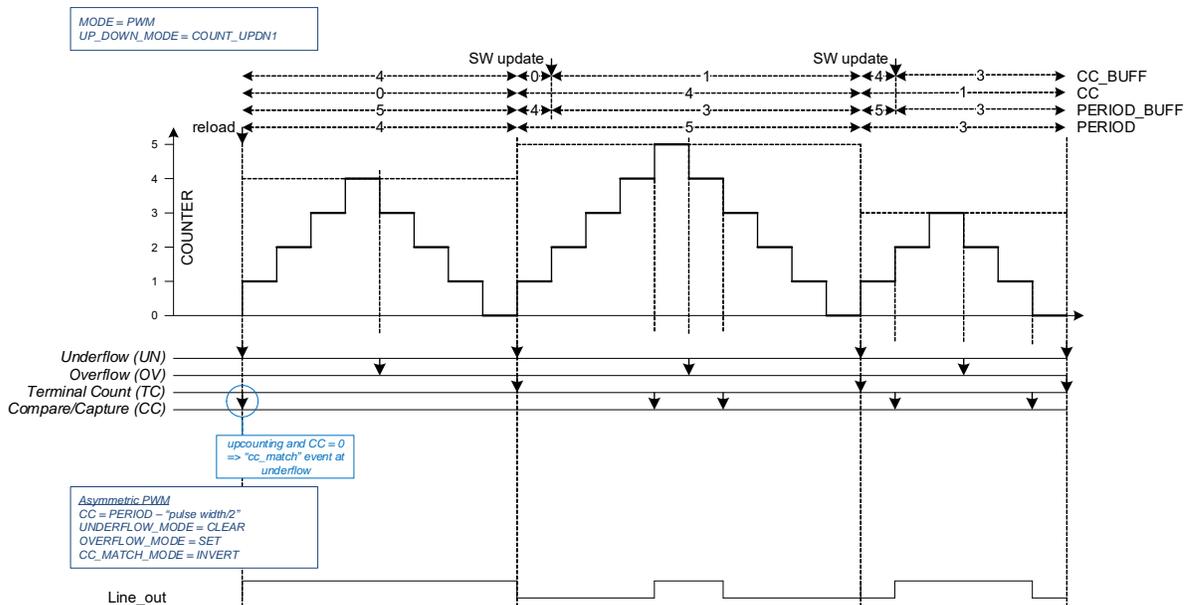


Figure 18-35 illustrates center-aligned PWM with PERIOD/PERIOD\_BUFF and CC/CC\_BUFF registers (up/down counting mode 1). At the TC condition, the PERIOD and CC registers are automatically exchanged with the PERIOD\_BUFF and CC\_BUFF registers. The swap event is generated by hardware trigger 1, which is a constant '1' and therefore always active at the TC condition. After the hardware exchange, the software handler on the tc interrupt updates PERIOD\_BUFF and CC\_BUFF.

Figure 18-35. PWM Mode CC Swap Event



The PERIOD swaps with PERIOD\_BUFF on a terminal count. The CC swaps with CC\_BUFF on a terminal count. Software can then update PERIOD\_BUFF and CC\_BUFF so that on the next terminal count PERIOD and CC will be updated with the values written into PERIOD\_BUFF and CC\_BUFF.

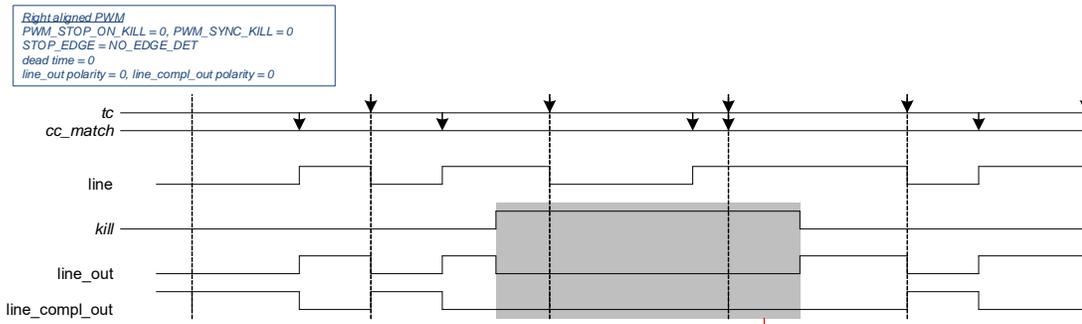
A potential problem arises when software updates are not completed before the next tc event with an active pending swap event. For example, if software updates PERIOD\_BUFF before the tc event and CC\_BUFF after the tc event, swapping does not reflect the CC\_BUFF register update. To prevent this from happening, the swap event should be generated by software through a register write after both the PERIOD\_BUFF and CC\_BUFF registers are updated. The swap event is kept pending by the hardware until the next tc event occurs.

The previous section addressed synchronized updates of the CC/CC\_BUFF and PERIOD/PERIOD\_BUFF registers of a single PWM using a software-generated swap event. During motor control, three PWMs work in unison and updates to all period and compare register pairs should be synchronized. All three PWMs have synchronized periods and as a result have synchronized tc events. The swap event for all three PWMs is generated by software through a single register write. The software should generate the swap events after the PERIOD\_BUFF and CC\_BUFF registers of all three PWMs are updated.

Figure 18-25 uses CTRL.QUADRATURE\_MODE[0] for “line\_out” polarity and CTRL.QUADRATURE\_MODE[1] for “line\_compl\_out” polarity. Figure 18-36 illustrates how the polarity settings control the PWM output signals “line\_out” and “line\_compl\_out”.

**Note:** When the counter is not running ((temporarily) stopped or killed), the PWM output signal values are determined by their respective polarity settings. When the counter is disabled the output values are low.

Figure 18-36. PWM Outputs When Killed



### 18.3.4.1 Asymmetric PWM

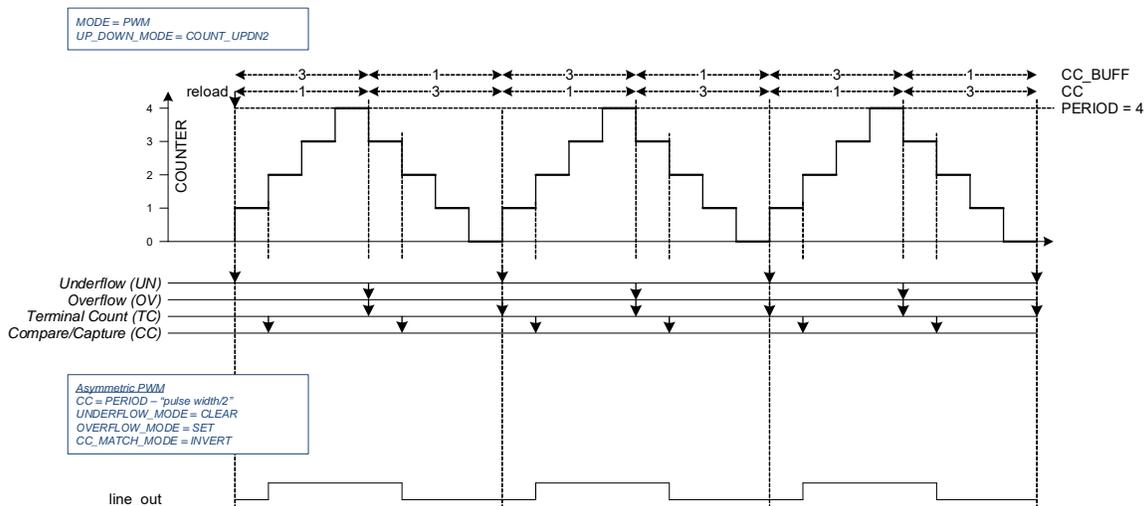
This PWM mode supports the generation of an asymmetric PWM. For an asymmetric PWM, the line pulse is not necessarily centered in the middle of the period. This functionality is realized by having a different CC value when counting up and when counting down. The CC and CC\_BUFF values are exchanged on an overflow event.

The COUNT\_UPDN2 mode should use the same period value when counting up and counting down. When PERIOD and PERIOD\_BUFF are swapped on a tc event (overflow or underflow event), care should be taken to ensure that:

- Within a PWM period (tc event coincides with an overflow event), the period values are the same (an overflow swap of PERIOD and PERIOD\_BUFF should not change the period value; that is, PERIOD\_BUFF should be PERIOD)
- Between PWM periods (tc event coincides with an underflow event), the period value can change (an underflow swap of PERIOD and PERIOD\_BUFF may change the period value; that is, PERIOD\_BUFF may be different from PERIOD).

Figure 18-37 illustrates how the COUNT\_UPDN2 mode is used to generate an asymmetric PWM.

Figure 18-37. Asymmetric PWM



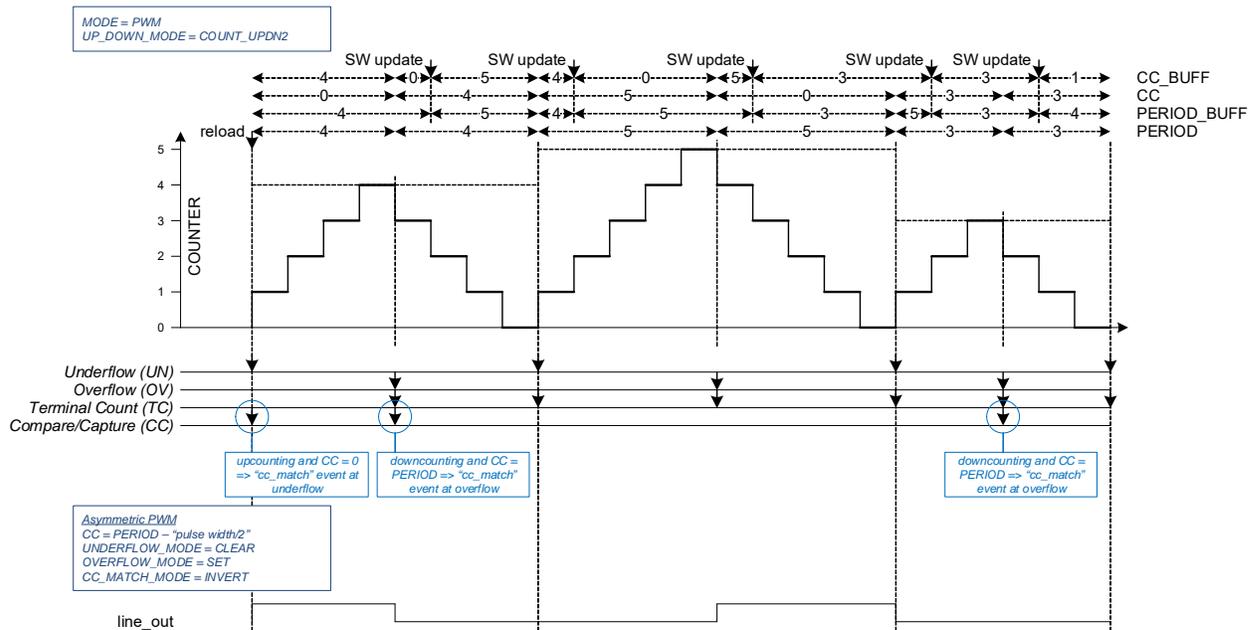
The previous waveform illustrated functionality when the CC values are neither “0” nor PERIOD. Corner case conditions in which the CC values equal “0” or PERIOD are illustrated as follows.

Figure 18-38 illustrates how the COUNT\_UPDN2 mode is used to generate an asymmetric PWM.

**Notes:**

- When up counting, when CC value at the underflow event is 0, a cc\_match event is generated.
- When down counting, when CC value at the overflow event is PERIOD, a cc\_match event is generated.
- A tc event is generated for both an underflow and overflow event. The tc event is used to exchange the CC and CC\_BUFF values.
- Software updates CC\_BUFF and PERIOD\_BUFF in an interrupt handler on the tc event (and overwrites the hardware updated values from the CC/CC\_BUFF and PERIOD/PERIOD\_BUFF exchanges).

Figure 18-38. Asymmetric PWM when Compare = 0 or Period



### 18.3.4.2 Configuring Counter for PWM Mode

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

1. Disable the counter by writing ‘0’ to the COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select PWM mode by writing ‘100b’ 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.
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 swap 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 swap 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.
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, swap, and count).
9. Set the TCPWM\_CNT\_TR\_CTRL1 register to select the edge that causes the event (reload, start, kill, swap, and count).
10. line\_out and line\_compl\_out 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.

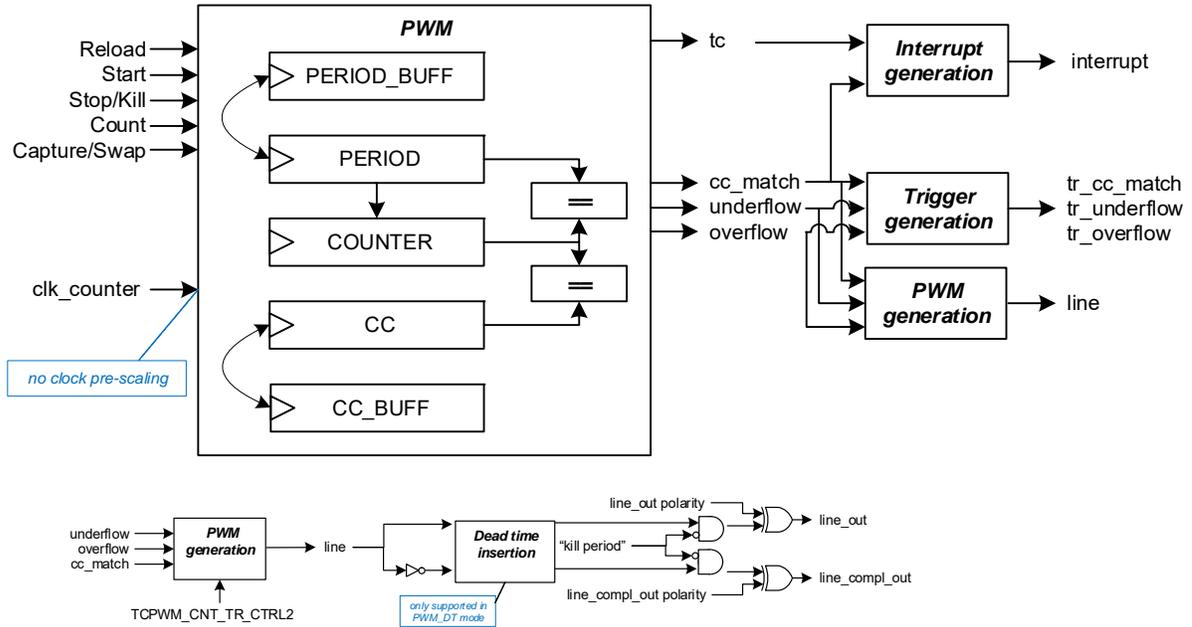
12. "Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A reload trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware reload signal is not enabled.

### 18.3.5 Pulse Width Modulation with Dead Time Mode

Dead time is used to delay the transitions of both "line" and "line\_n" signals. It separates the transition edges of these two signals by a specified time interval. A maximum dead time of 255 clocks can be generated using this feature. The PWM-DT functionality is the same as PWM functionality, except for the following differences:

- PWM\_DT supports dead time insertion; PWM does not support dead time insertion.
- PWM\_DT does not support clock pre-scaling; PWM supports clock pre-scaling.

Figure 18-39. PWM with Dead Time Functionality



Dead time insertion is a step that operates on a preliminary PWM output signal line, as illustrated in [Figure 18-39](#).

[Figure 18-40](#) illustrates dead time insertion for different dead times and different output signal polarity settings.

Figure 18-40. Dead-time Timing

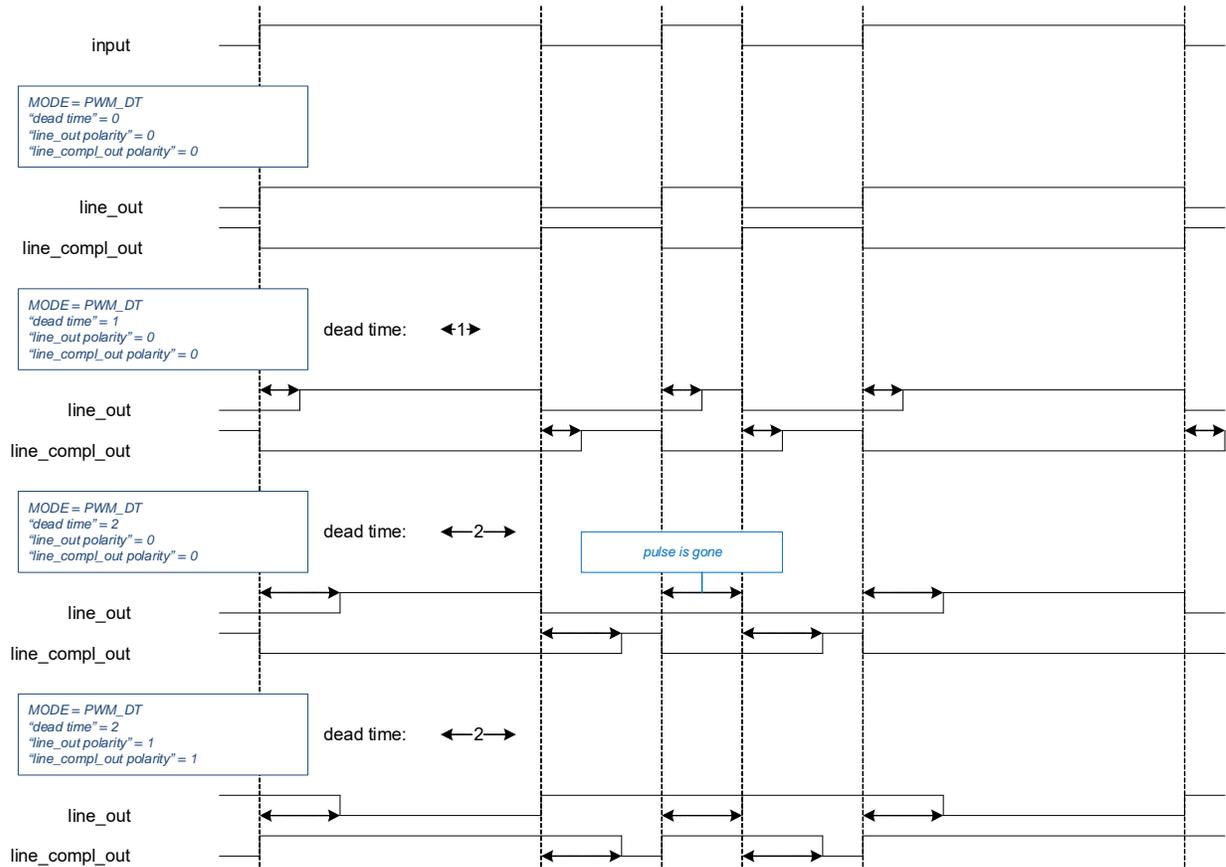
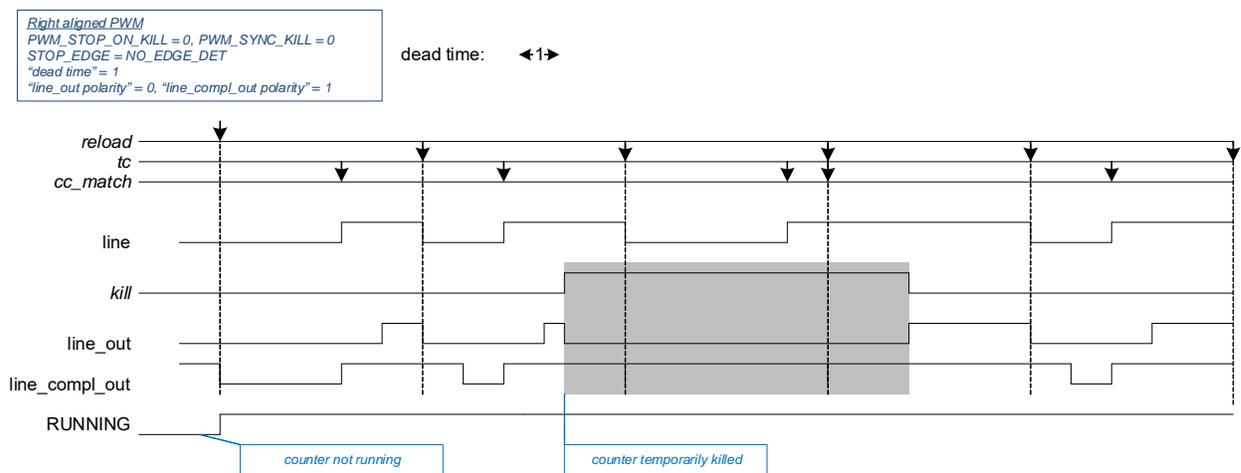


Figure 18-41 illustrates how the polarity settings and stop/kill functionality combined control the PWM output signals “line\_out” and “line\_compl\_out”.

Figure 18-41. Dead Time and Kill



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

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

1. Disable the counter by writing '0' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM\_CNT\_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM\_CNT\_CTRL register.
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 swap 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 swap 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.
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, swap, and count).
9. Set the TCPWM\_CNT\_TR\_CTRL1 register to select the edge that causes the event (reload, start, kill, swap, and count).
10. line\_out and line\_compl\_out 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.
12. Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A start trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware start signal is not enabled.

### 18.3.6 Pulse Width Modulation Pseudo-Random Mode (PWM\_PR)

The PWM\_PR functionality changes the counter value using the linear feedback shift register (LFSR). This results in a pseudo random number sequence. A signal similar to PWM signal is created by comparing the counter value COUNTER with CC. The generated signal has different frequency/noise characteristics than a regular PWM signal.

Table 18-26. PWM\_PR Mode Trigger Inputs

| Trigger Inputs | Usage  |
|----------------|--|
| reload         | Same behavior as start event.<br>Can be used only when the counter is not running.   |
| start          | Starts the counter. The counter is not initialized by hardware. The current counter value is used. Behavior is dependent on UP_DOWN_MODE.<br>Can be used only when the counter is not running.   |
| stop/kill      | Stops the counter. Different stop/kill modes exist.  |
| count          | Not used.  |
| capture        | This event acts as a swap event. When this event is active, the CC/CC_BUFF and PERIOD/PERIOD_BUFF registers are exchanged on a tc event (when specified by CTRL.AUTO_RELOAD_CC and CTRL.AUTO_RELOAD_PERIOD).<br>A swap event requires rising, falling, or rising/falling edge event detection mode. Pass-through mode is not supported, unless the selected event is a constant '0' or '1'.<br>When a swap event is detected and the counter is running, the event is kept pending until the next tc event.<br>When a swap event is detected and the counter is not running, the event is cleared by hardware. |

**Note:** Event detection is on the "clk\_counter" while in Quadrature mode and "clk\_hf\_counter" for all other modes.

Table 18-27. PWM\_PR Supported Features

| Supported Features | Description  |
|--------------------|--|
| Clock pre-scaling  | Pre-scales the counter clock, <code>clk_counter</code> .   |
| One-shot           | Counter is stopped by hardware, after a single period of the counter (counter value equals period value PERIOD). |
| Auto reload CC     | CC and CC_BUFF are exchanged on a swap event AND tc event (when specified by CTRL.AUTO_RELOAD_CC).               |
| Auto reload PERIOD | PERIOD and PERIOD_BUFF are exchanged on a swap event and tc event (when specified by CTRL.AUTO_RELOAD_PERIOD).   |
| Kill modes         | Specified by PWM_STOP_ON_KILL. See memory map for further details.   |

**Note:** The count event is not used. As a result, the PWM\_PR functionality operates on the pre-scaled counter clock (`clk_counter`), rather than on an “active count” pre-scaled counter clock.

Table 18-28. PWM\_PR Trigger Outputs

| Trigger Outputs            | Description  |
|----------------------------|--|
| <code>cc_match (CC)</code> | Counter changes from a state in which COUNTER equals CC. |
| Underflow (UN)             | Not used.  |
| Overflow (OV)              | Not used.  |

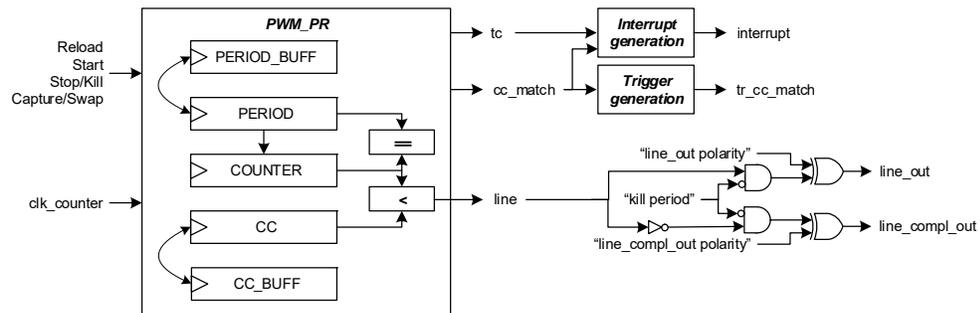
Table 18-29. PWM\_PR Interrupt Outputs

| Interrupt Outputs          | Description  |
|----------------------------|--|
| <code>cc_match (CC)</code> | Counter changes from a state in which COUNTER equals CC.     |
| <code>tc</code>            | Counter changes from a state in which COUNTER equals PERIOD. |

Table 18-30. PWM\_PR PWM Outputs

| PWM Outputs                 | Description               |
|-----------------------------|---------------------------|
| <code>line_out</code>       | PWM output.               |
| <code>line_compl_out</code> | Complementary PWM output. |

Figure 18-42. PWM\_PR Functionality



The PWM\_PR functionality is described as follows:

- The counter value COUNTER is initialized by software (to a value different from 0).
  - A reload or start event starts PWM\_PR operation.
  - During PWM\_PR operation:
    - The counter value COUNTER is changed based on the LFSR polynomial:  $x^{16} + x^{14} + x^{13} + x^{11} + 1$  ([en.wikipedia.org/wiki/Linear\\_feedback\\_shift\\_register](http://en.wikipedia.org/wiki/Linear_feedback_shift_register)).
- temp = (COUNTER >> (16-16)) ^ (COUNTER >> (16-14)) ^ (COUNTER >> (16-13)) ^ (COUNTER >> (16-11)) or  
temp = (COUNTER >> 0) ^ (COUNTER >> 2) ^ (COUNTER >> 3) ^ (COUNTER >> 5);  
(COUNTER = (temp << 15)) | (COUNTER >> 1)

This will result in a pseudo random number sequence for COUNTER. For example, when COUNTER is initialized to 0xACE1, the number sequence is: 0xACE1, 0x5670, 0xAB38, 0x559C, 0x2ACE, 0x1567, 0x8AB3... This sequence will repeat itself after  $2^{16} - 1$  or 65535 counter clock cycles.

- When the counter value COUNTER equals CC, a cc\_match event is generated.
- When the counter value COUNTER equals PERIOD, a tc event is generated.
- On a tc event, the CC/CC\_BUFF and PERIOD/PERIOD\_BUFF can be conditionally exchanged under control of the capture/swap event and the CTRL.AUTO\_RELOAD\_CC and CTRL.AUTO\_RELOAD\_PERIOD field (see PWM functionality).
- The output line reflects: COUNTER[14:0] < CC[15:0]. Note that only the lower 15 bits of COUNTER are used for comparison, while the COUNTER itself can run up to 16-bit values. As a result, for CC greater or equal to 0x8000, pwm\_dt\_input is always 1. The line polarity can be inverted (as specified by CTRL.QUADRATURE\_MODE[0]).

As mentioned, different stop/kill modes exist. The mode is specified by PWM\_STOP\_ON\_KILL (PWM\_SYNC\_KILL should be '0' – asynchronous kill mode). The memory map describes the modes and the desired settings for the stop/kill event. The following two modes are supported:

- PWM\_STOP\_ON\_KILL is '1'. This mode stops the counter on a stop/kill event.
- PWM\_STOP\_ON\_KILL is '0'. This mode keeps the counter running, but suppresses the PWM output signals immediately and continues to do so for the duration of the stop/kill event.

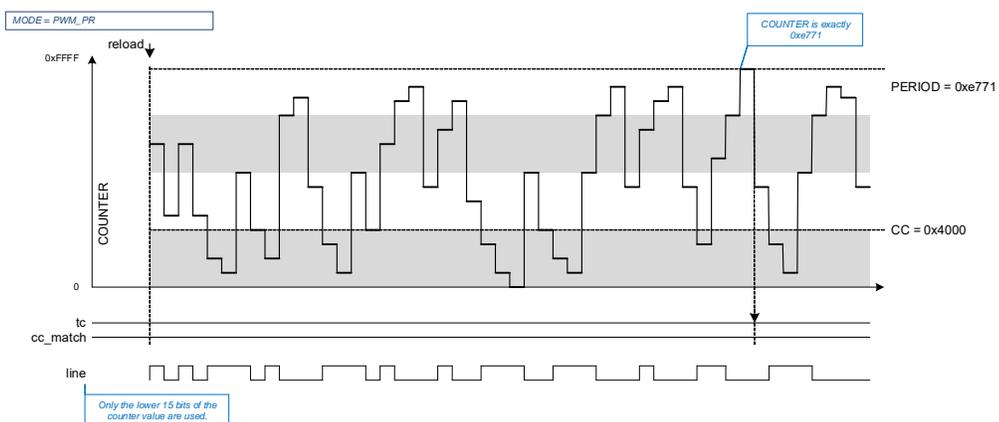
Note that the LFSR produces a deterministic number sequence (given a specific counter initialization value). Therefore, it is possible to calculate the COUNTER value after a certain number of LFSR iterations, n. This calculated COUNTER value can be used as PERIOD value, and the tc event will be generated after precisely n counter clocks.

Figure 18-43 illustrates PWM\_PR functionality.

**Notes:**

- The grey shaded areas represent the counter region in which the line value is '1', for a CC value of 0x4000. There are two areas, because only the lower 15 bits of the counter value are used.
- When CC is set to 0x4000, roughly one-half of the counter clocks will result in a line value of '1'.

Figure 18-43. PWM\_PR Output



### 18.3.6.1 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 COUNTER\_ENABLED field of the TCPWM\_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM\_CNT\_CTRL register.
3. Set the required period (16 bit) in the TCPWM\_CNT\_PERIOD register and buffer period value in the TCPWM\_CNT\_PERIOD\_BUFF register to swap values, if required.
4. Set the 16-bit compare value in the TCPWM\_CNT\_CC register and the buffer compare value in the TCPWM\_CNT\_CC\_BUFF register to swap 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 swap).
7. Set the TCPWM\_CNT\_TR\_CTRL1 register to select the edge that causes the event (reload, start, kill, and swap).
8. line\_out and line\_compl\_out 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.
10. Enable the counter by writing '1' to the COUNTER\_ENABLED field of the TCPWM\_CTRL register. A reload trigger must be provided through firmware (TCPWM\_CMD register) to start the counter if the hardware reload signal is not enabled.

## 18.4 TCPWM Registers

Table 18-31. List of TCPWM Registers

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

# 19. LCD Direct Drive



The PSoC 4 Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PSoC device to directly drive STN and TN segment LCDs.

## 19.1 Features

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

- Supports up to eight commons and 64 - Number of commons' segment lines.
- Supports Type A (standard) and Type B (low-power) drive waveforms
- Any GPIO can be configured as a common or segment
- Supports four drive methods:
  - PWM at 1/2 bias
  - PWM at 1/3 bias
  - PWM at 1/4 bias
  - PWM at 1/5 bias
- Operates in active and sleep modes
- Digital contrast control

## 19.2 LCD Segment Drive Overview

A segmented LCD panel has the liquid crystal material between two sets of electrodes and various polarization and reflector layers. The two electrodes of an individual segment are called commons (COM) or backplanes and segment electrodes (SEG). From an electrical perspective, an LCD segment can be considered as a capacitive load; the COM/SEG electrodes can be considered as the rows and columns in a matrix of segments. The opacity of an LCD segment is controlled by varying the root-mean-square (RMS) voltage across the corresponding COM/SEG pair.

The following terms/voltages are used in this chapter to describe LCD drive:

- **$V_{RMSOFF}$** : The voltage that the LCD driver can realize on segments that are intended to be off.
- **$V_{RMSON}$** : The voltage that the LCD driver can realize on segments that are intended to be on.
- **Discrimination Ratio (D)**: The ratio of  $V_{RMSON}$  and  $V_{RMSOFF}$  that the LCD driver can realize. This depends on the type of waveforms applied to the LCD panel. Higher discrimination ratio results in higher contrast.

Liquid crystal material does not tolerate long term exposure to DC voltage. Therefore, any waveforms applied to the panel must produce a 0-V DC component on every segment (on or off). Typically, LCD drivers apply waveforms to the COM and SEG electrodes that are generated by switching between multiple voltages. The following terms are used to define these waveforms:

- **Duty**: A driver is said to operate in 1/M duty when it drives 'M' number of COM electrodes. Each COM electrode is effectively driven 1/M of the time.
- **Bias**: A driver is said to use 1/B bias when its waveforms use voltage steps of  $(1/B) \times V_{DRV}$ .  $V_{DRV}$  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.

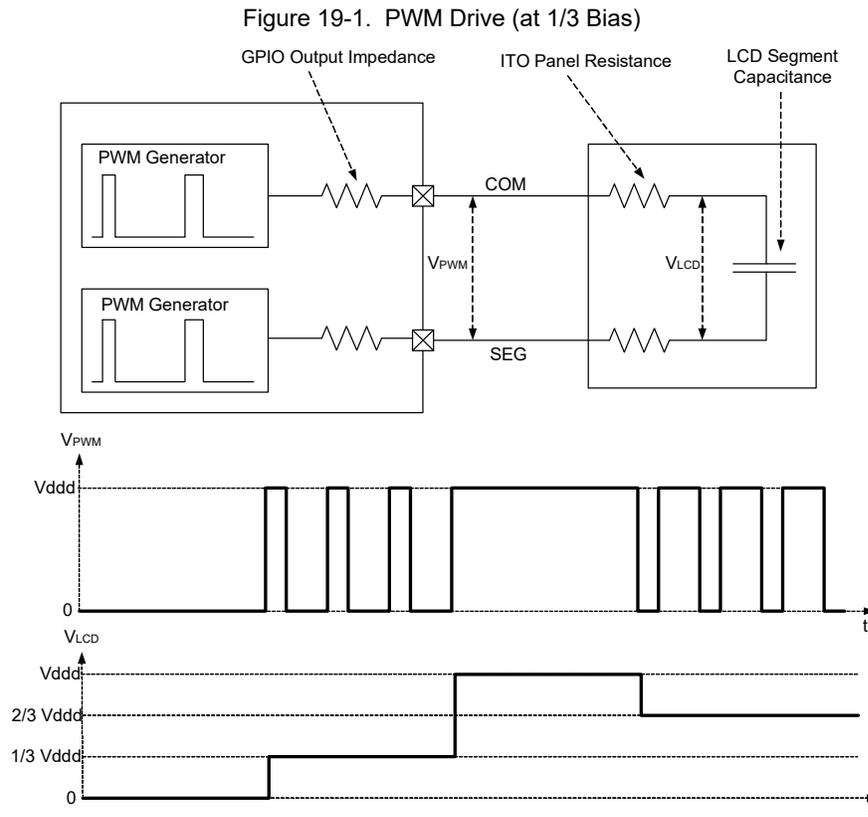
## 19.2.1 Drive Modes

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
- PWM drive at 1/5 bias

### 19.2.1.1 PWM Drive

In PWM drive mode, multi-voltage drive signals are generated using a PWM output signal together with the intrinsic resistance and capacitance of the LCD. [Figure 19-1](#) illustrates this.



The output waveform of the drive electronics is a PWM waveform. With the Indium Tin Oxide (ITO) panel resistance and the segment capacitance to filter the PWM, the voltage across the LCD segment is an analog voltage, as shown in [Figure 19-1](#). This figure illustrates the generation of a 1/3 bias waveform (four commons and voltage steps of  $V_{DD}/3$ ).

The PWM is derived from the IMO clock (high-speed operation). The generated analog voltage typically runs at very low frequency (~ 50 Hz) for segment LCD driving.

Figure 19-2 and Figure 19-3 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/2 bias and 1/4 duty. Only COM0/COM1 and SEG0/SEG1 are drawn for demonstration purpose. Similarly, Figure 19-4 and Figure 19-5 illustrate the Type A and Type B waveforms for COM and SEG electrodes for 1/3 bias and 1/4 duty.

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

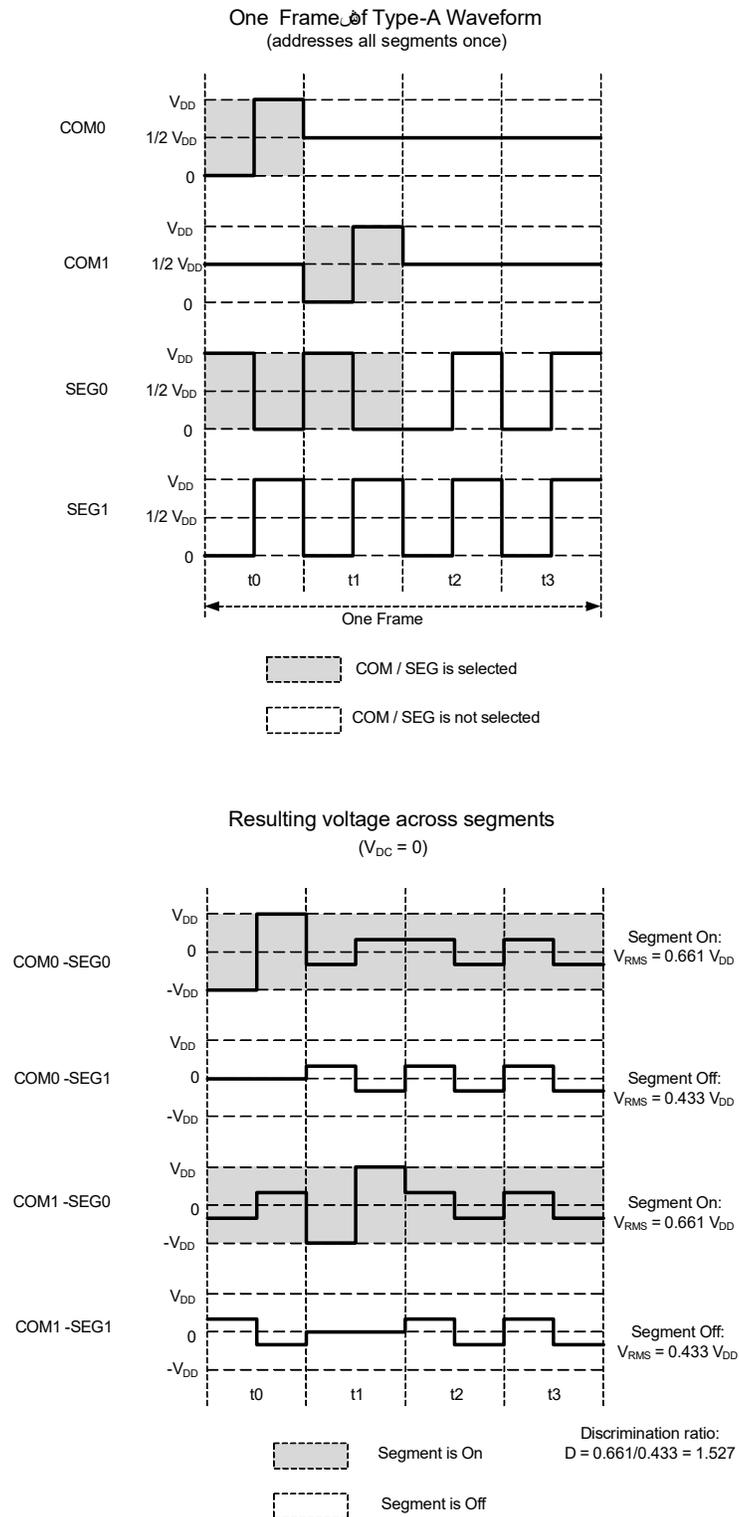


Figure 19-3. PWM1/2 Type-B Waveform Example

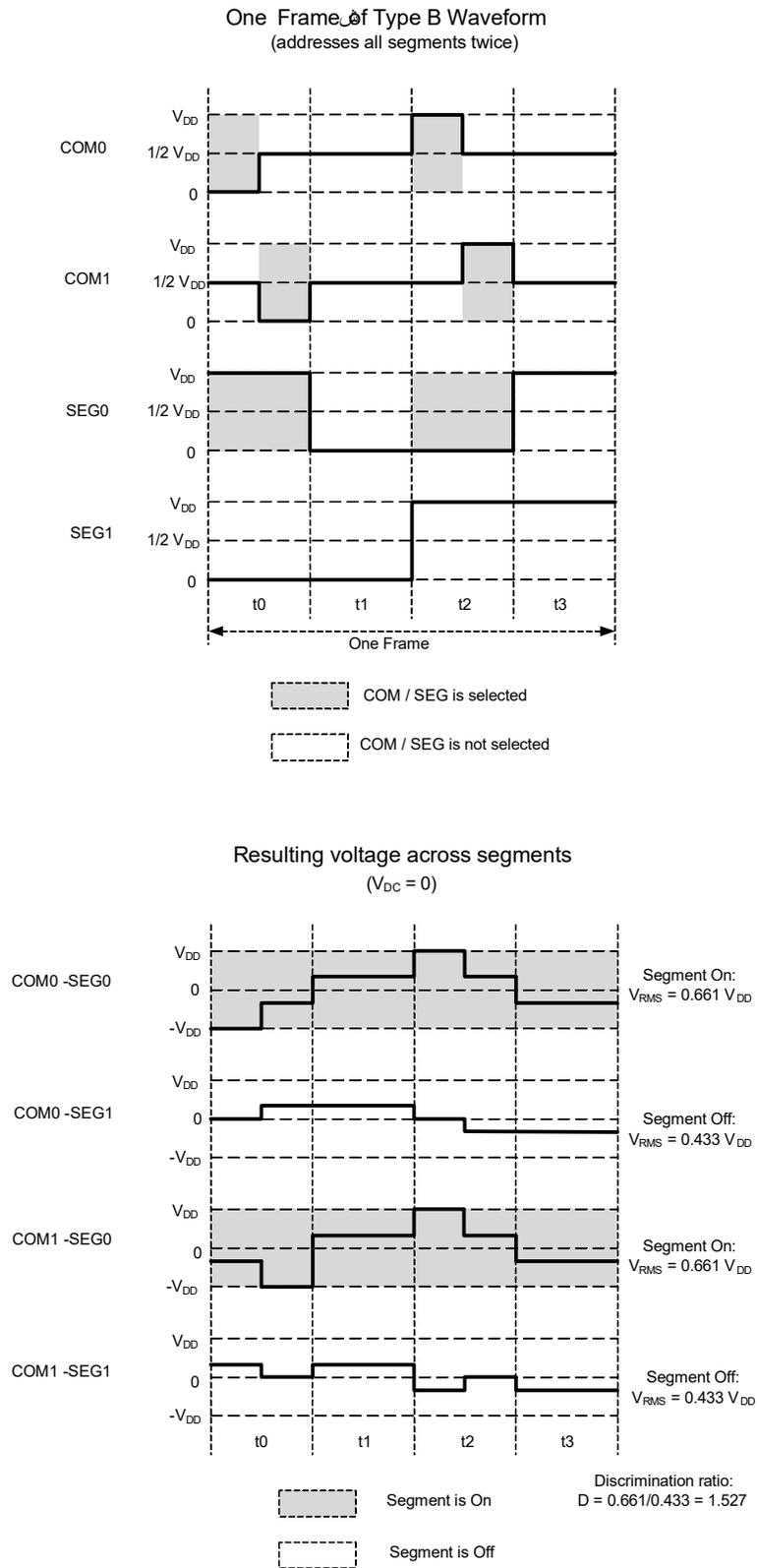


Figure 19-4. PWM1/3 Type-A Waveform Example

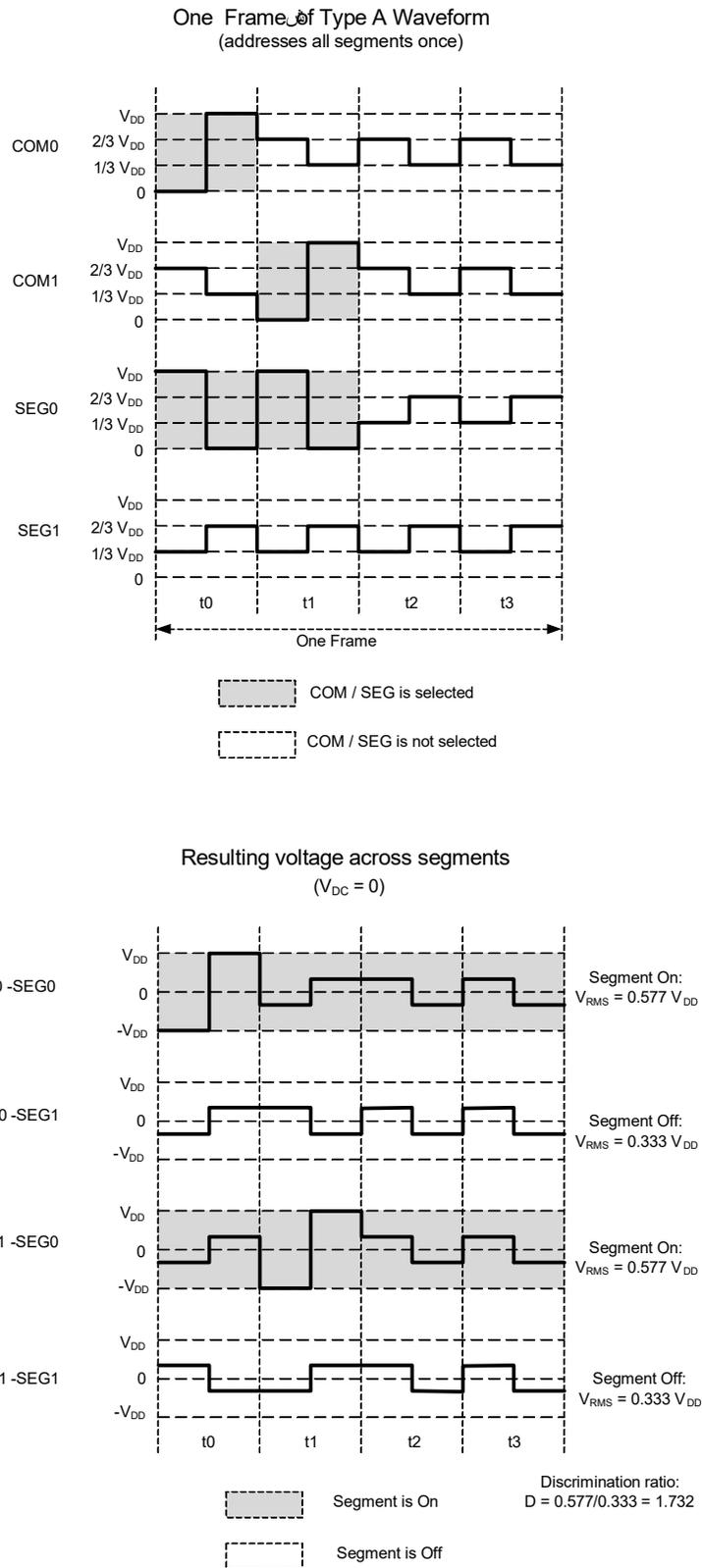
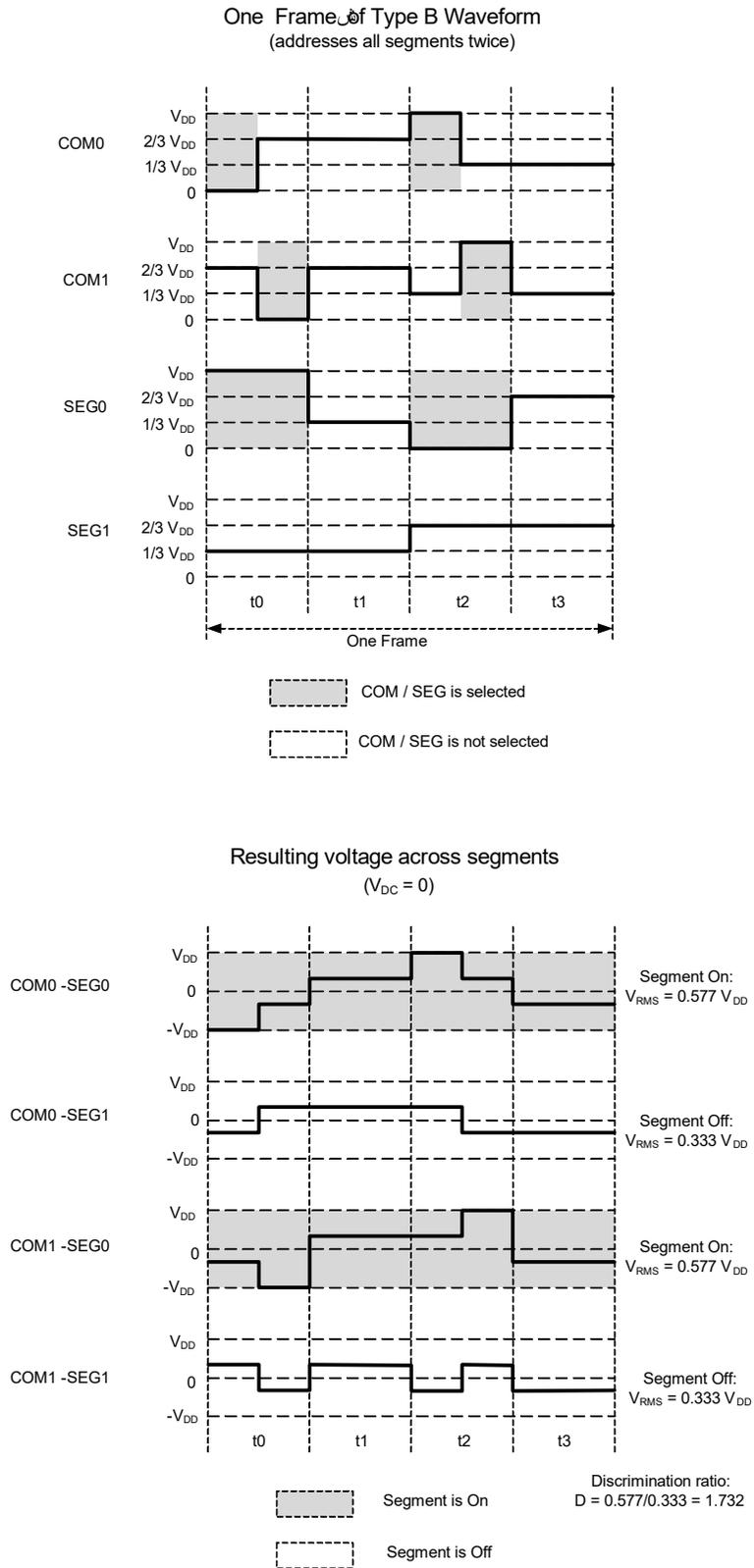


Figure 19-5. PWM1/3 Type-B Waveform Example



The effective RMS voltage for ON and OFF segments can be calculated easily using [Equation 19-1](#) and [Equation 19-2](#).

$$RMS(OFF) = \sqrt{\frac{2(B-2)^2 + 2(M-1)}{2M}} \times \left(\frac{V_{DRV}}{B}\right) \tag{Equation 19-1}$$

$$RMS(ON) = \sqrt{\frac{2B^2 + 2(M-1)}{2M}} \times \left(\frac{V_{DRV}}{B}\right) \tag{Equation 19-2}$$

Where, B is the bias and M is the duty (number of COMs).

For example, if the number of COMs is four, the resulting discrimination ratios (D) for 1/2 and 1/3 biases are 1.528 and 1.732, respectively. 1/3 bias offers better discrimination ratio in two and three COM drives also. Therefore, 1/3 bias offers better contrast than 1/2 bias and is recommended for most applications. They offer better discrimination ratio especially when used in designs with more than four COM drives.

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

The 1/2 bias mode has the advantage that PWM is only required on the COM signals; the SEG signals use only logic levels, as shown in [Figure 19-2](#) and [Figure 19-3](#).

### 19.2.2 Recommended Bias Settings

Table 19-1. Recommended Usage of Bias Settings

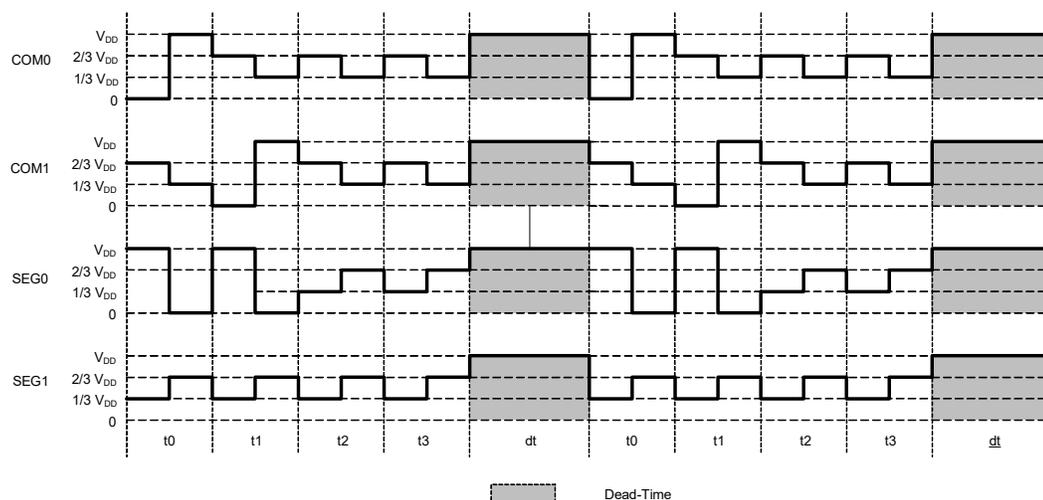
| Display Type       | Bias Setting              |
|--------------------|---------------------------|
| Four COM TN Glass  | PWM 1/3 bias              |
| Four COM STN Glass | PWM 1/3 bias              |
| Eight COM, STN     | PWM 1/4 bias and 1/5 bias |

### 19.2.3 Digital Contrast Control

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

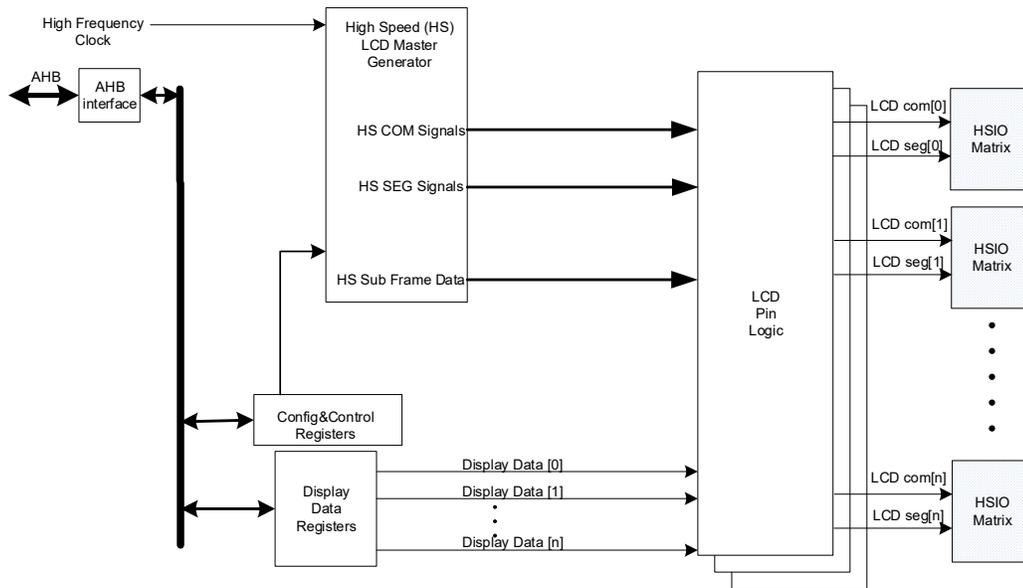
Figure 19-6. Dead-Time Contrast Control

Two Frames of of Type A Waveform with Dead-time  
(Example for 1/4<sup>th</sup> Duty and 1/3<sup>rd</sup> bias)



## 19.3 Block Diagram

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



### 19.3.1 How it Works

The LCD pin logic block routes the COM and SEG outputs from the LCD waveform generator to the corresponding I/O matrices. Any GPIO can be used as either a COM or SEG. This configurable pin assignment for COM or SEG is implemented in the GPIO and I/O matrix; see [“High-Speed I/O Matrix” on page 72](#).

The LCD controller operates in two device power modes: active and sleep. The LCD controller should not be enabled in deep sleep mode because operation is not possible without the IMO. The LCD controller is unpowered in hibernate and stop modes.

### 19.3.2 High-Speed Master Generator

The master generator has the following features and characteristics:

- Register bit configuring the block for either Type A or Type B drive waveforms (LCD\_MODE bit in LCD\_CONTROL register).
- Register bits to select the number of COMs (COM\_NUM field in LCD\_CONTROL register). The available values are 2, 3, 4, and 8.
- Operating mode configuration bits enabled to select one of the following:
  - PWM 1/2 bias
  - PWM 1/3 bias
  - PWM 1/4 bias
  - PWM 1/5 bias
  - Off/disabled
 OP\_MODE and BIAS fields in LCD\_CONTROL bits select the drive mode.
- A counter to generate the sub-frame timing. The SUBFR\_DIV field in the LCD\_DIVIDER register determines the duration of each sub-frame. If the divide value written into this counter is C, the sub-frame period is  $4 \times (C+1)$ . The high-speed generator has a 16-bit counter.
- A counter to generate the dead time period. These counters have the same number of bits as the sub-frame period counters and use the same clocks. DEAD\_DIV field in the LCD\_DIVIDER register controls the dead time period.

### 19.3.3 LCD Pin Logic

The LCD pin logic uses the sub-frame signal from the multiplexer to choose the display data. This pin logic will be replicated for each LCD pin.

### 19.3.4 Display Data Registers

Each LCD segment pin is part of an LCD port with its own display data register, LCD\_DATA<sub>x</sub>. 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\_DATA<sub>x</sub> registers are 32-bit wide and store the ON/OFF data for all SEG-COM combination enabled in the design. LCD\_DATA0<sub>x</sub> holds SEG-COM data for COM0 to COM3 and LCD\_DATA1<sub>x</sub> holds SEG-COM data for COM4 to COM7. The bits [4i+3:4i] (where 'i' is the pin number) of each LCD\_DATA0<sub>x</sub> register represent the ON/OFF data for Pin[i] in Port[x] and COM[3,2,1,0] combinations, as shown in Table 19-2. The LCD\_DATA<sub>x</sub> register should be programmed according to the display data of each frame. The display data registers are Memory Mapped I/O (MMIO) and accessed through the AHB slave interface.

Table 19-2. SEG-COM Mapping in LCD\_DATA0<sub>x</sub> Registers (each SEG is a pin of the LCD port)

| BITS[31:28] = PIN_7[3:0] |            |            |            | BITS[27:24] = PIN_6[3:0] |            |            |            |
|--------------------------|------------|------------|------------|--------------------------|------------|------------|------------|
| PIN_7-COM3               | PIN_7-COM2 | PIN_7-COM1 | PIN_7-COM0 | PIN_6-COM3               | PIN_6-COM2 | PIN_6-COM1 | PIN_6-COM0 |
| BITS[23:20] = PIN_5[3:0] |            |            |            | BITS[19:16] = PIN_4[3:0] |            |            |            |
| PIN_5-COM3               | PIN_5-COM2 | PIN_5-COM1 | PIN_5-COM0 | PIN_4-COM3               | PIN_4-COM2 | PIN_4-COM1 | PIN_4-COM0 |
| BITS[15:12] = PIN_3[3:0] |            |            |            | BITS[11:8] = PIN_2[3:0]  |            |            |            |
| PIN_3-COM3               | PIN_3-COM2 | PIN_3-COM1 | PIN_3-COM0 | PIN_2-COM3               | PIN_2-COM2 | PIN_2-COM1 | PIN_2-COM0 |
| BITS[7:3] = PIN_1[3:0]   |            |            |            | BITS[3:0] = PIN_0[3:0]   |            |            |            |
| PIN_1-COM3               | PIN_1-COM2 | PIN_1-COM1 | PIN_1-COM0 | PIN_0-COM3               | PIN_0-COM2 | PIN_0-COM1 | PIN_0-COM0 |

## 19.4 Register List

Table 19-3. LCD Direct Drive Register List

| Register Name          | Description   |
|------------------------|---|
| LCD_ID                 | This register includes the information of LCD controller' ID and revision number        |
| LCD_DIVIDER            | This register controls the sub-frame and dead-time period                               |
| LCD_CONTROL            | This register is used to configure high-speed and low-speed generators                  |
| LCD_DATA0 <sub>x</sub> | LCD port pin data register for COM0 to COM3; x = port number, eight ports are available |
| LCD_DATA1 <sub>x</sub> | LCD port pin data register for COM4 to COM7; x = port number, eight ports are available |

## 20. Inter-IC Sound Bus



The Inter-IC Sound Bus (I<sup>2</sup>S) is a serial bus interface standard used to connect digital audio devices together. The specification is from Philips<sup>®</sup> Semiconductor (I<sup>2</sup>S bus specification: February 1986, revised June 5, 1996). In addition to the standard I<sup>2</sup>S format, the I<sup>2</sup>S block also supports the Left Justified (LJ) format and the Time Division Multiplexed (TDM) format.

### 20.1 Features

- Supports standard Philips I<sup>2</sup>S, LJ, and eight-channel TDM digital audio interface formats
- Supports master mode operation in all the digital audio formats
- Supports only Transmit (Tx) operations
- Supports operating from an external master clock provided through an audio codec
- Provides configurable clock divider registers to generate the required sample rates
- Supports data word length of 8-bit, 16-bit, 18-bit, 20-bit, 24-bit, and 32-bit per channel
- Supports channel length of 8-bit, 16-bit, 18-bit, 20-bit, 24-bit, and 32-bit per channel (channel length fixed at 32-bit in TDM format)
- Provides hardware FIFO buffer, for the Tx block
- Supports CPU-based data transfers

### 20.2 Architecture

Figure 20-1. I<sup>2</sup>S Block Diagram

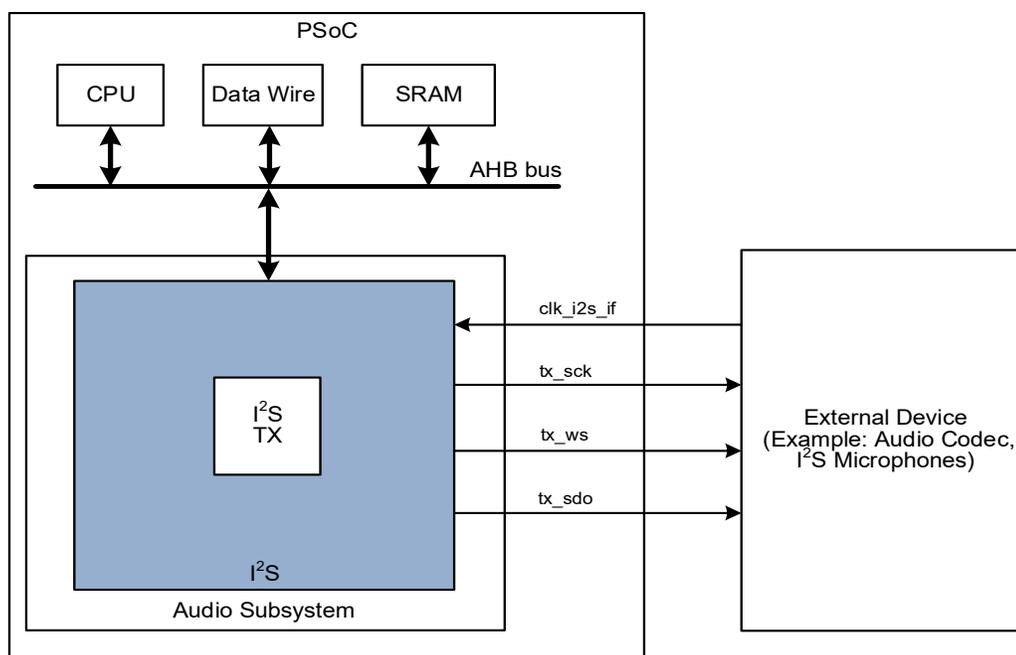


Figure 20-1 shows the high-level block diagram of the I<sup>2</sup>S block, which consists of only I<sup>2</sup>S Transmitter (Tx) sub-block. The digital audio interface format and master mode configuration can be done independently for the Tx block. The word select (ws) and serial data clock (sck) are generated by the I<sup>2</sup>S block in the PSoC 4100S Max device. The I<sup>2</sup>S block configuration, control, and status registers, along with the FIFO data buffers are accessible through the AHB bus. CPU can access the I<sup>2</sup>S registers through the AHB interface. Refer to the [device datasheet](#) for information on port pin assignments of the I<sup>2</sup>S block signals and AC/DC electrical specifications.

## 20.3 Digital Audio Interface Formats

The I<sup>2</sup>S block supports the following digital audio interface formats:

- Standard I<sup>2</sup>S format
- Left Justified format
- Time Division Multiplexed (TDM) format

The Tx sub-block can be configured to support one of the above formats in master mode. The I2S\_MODE bits in the I2S\_TX\_CTL register is used to configure the digital audio interface format. Only master mode is supported.

### 20.3.1 Standard I<sup>2</sup>S Format

Figure 20-2 shows the timing diagrams for the different word length and channel length combinations in the standard I<sup>2</sup>S digital audio format. In the standard I<sup>2</sup>S format, the word select signal (ws) is low for left channel data, and high for right channel data. The ws signal transitions one bit-clock (sck) early relative to the start of the left/right channel data. All the serial data (sd), ws signal transitions on the falling edge of the sck signal, and the read operations on the ws and sd lines are usually done on the rising edge of sck. Therefore, the I<sup>2</sup>S Tx block writes to the serial data (tx\_sdo) line on the falling edge of tx\_sck. The serial data is transmitted most significant bit (MSb) first. The ws/sck signals are generated by the I2S TX block.

The I<sup>2</sup>S block supports configurable word length and channel length selection options. The word length for the Tx block can be configured using the WORD\_LEN bits in the I2S\_TX\_CTL register. The channel length for the Tx block can be configured using the CH\_LEN bits in the I2S\_TX\_CTL register.

In the Tx block, when the channel length is greater than the word length, the unused bits can be transmitted either as '0' or '1'. This selection is made using the OVHDATA bit in the I2S\_TX\_CTL register.

Figure 20-2. Standard I<sup>2</sup>S Format (Word Length and Channel Length Combination Timing Diagrams)

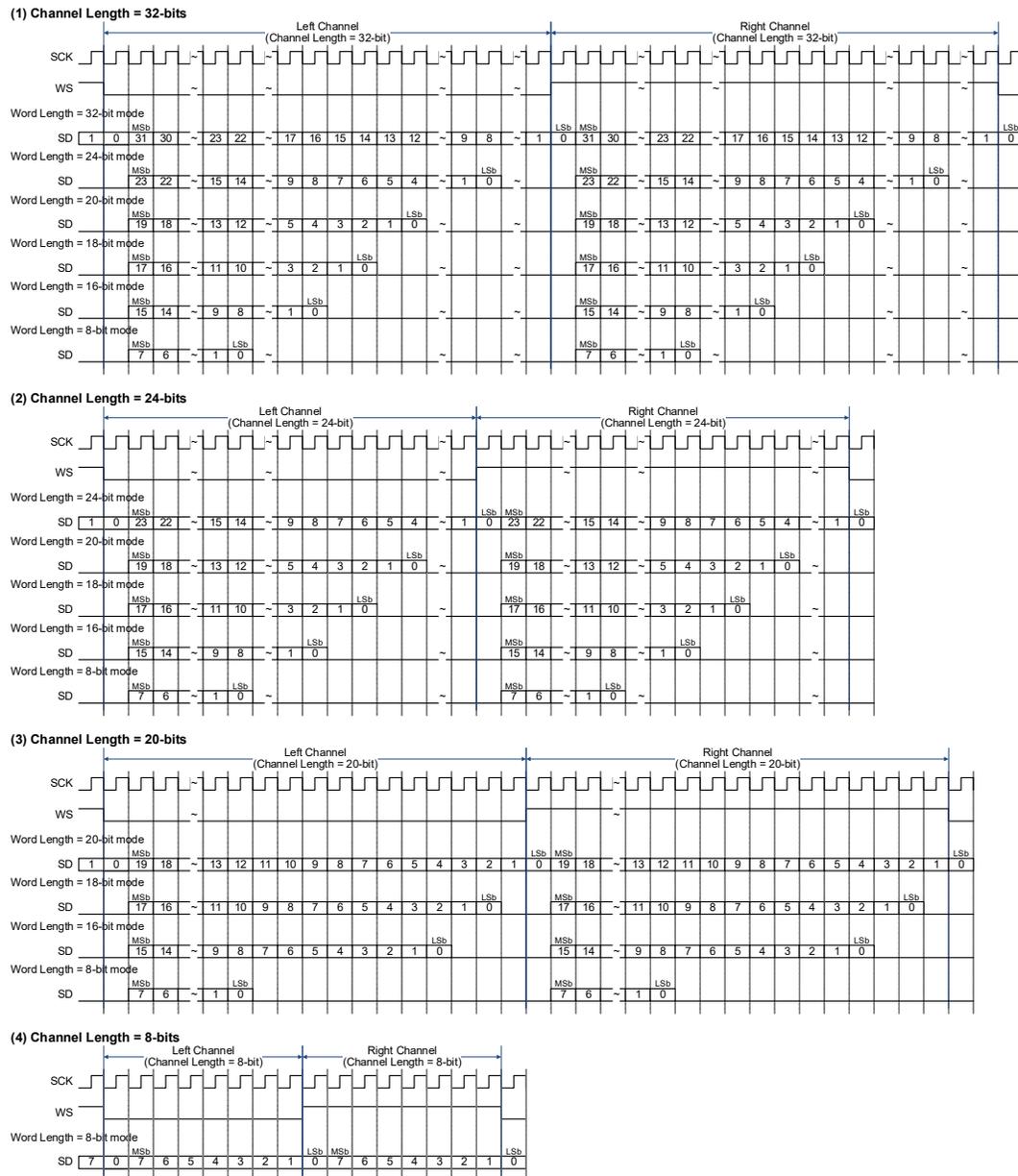


Table 20-1 lists the supported word length and channel length combinations.

Table 20-1. Word Length and Channel Length Combinations

|                |        | Word Length |         |         |         |         |         |
|----------------|--------|-------------|---------|---------|---------|---------|---------|
|                |        | 8-bit       | 16-bit  | 18-bit  | 20-bit  | 24-bit  | 32-bit  |
| Channel Length | 32-bit | Valid       | Valid   | Valid   | Valid   | Valid   | Valid   |
|                | 24-bit | Valid       | Valid   | Valid   | Valid   | Valid   | Invalid |
|                | 30-bit | Valid       | Valid   | Valid   | Valid   | Invalid | Invalid |
|                | 18-bit | Valid       | Valid   | Valid   | Invalid | Invalid | Invalid |
|                | 16-bit | Valid       | Valid   | Invalid | Invalid | Invalid | Invalid |
|                | 8-bit  | Valid       | Invalid | Invalid | Invalid | Invalid | Invalid |

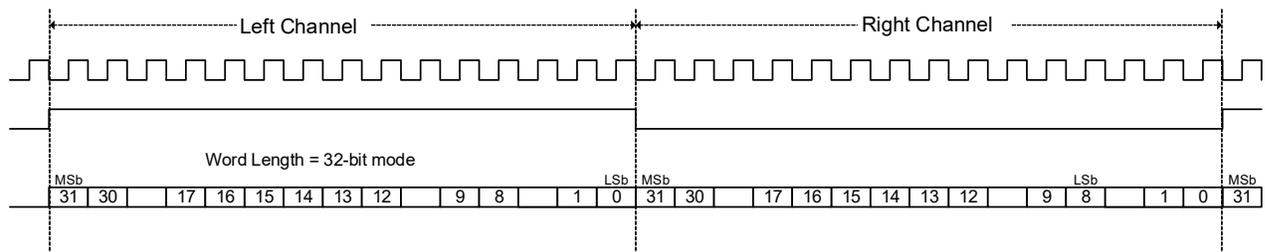
### 20.3.2 Left Justified (LJ) Format

Figure 20-3 shows the timing diagrams for the Left Justified interface format using the 32-bit channel length and 32-bit word length configuration as an example. The only differences between the standard I<sup>2</sup>S and LJ formats are:

- In the standard I<sup>2</sup>S format, WS signal is low for left channel data and high for right channel data. In the LJ format, WS signal is high for left channel data and low for right channel data.
- In the standard I<sup>2</sup>S format, WS signal transitions one bit-clock (sck) early relative to the start of the channel data (coincides with LSb of the previous channel). In the LJ format, there is no early transition, and the WS signal transitions coincide with the start of the channel data.

Apart from these differences, all the features explained in the standard I<sup>2</sup>S format section apply to the LJ format as well.

Figure 20-3. Left Justified Digital Audio Format

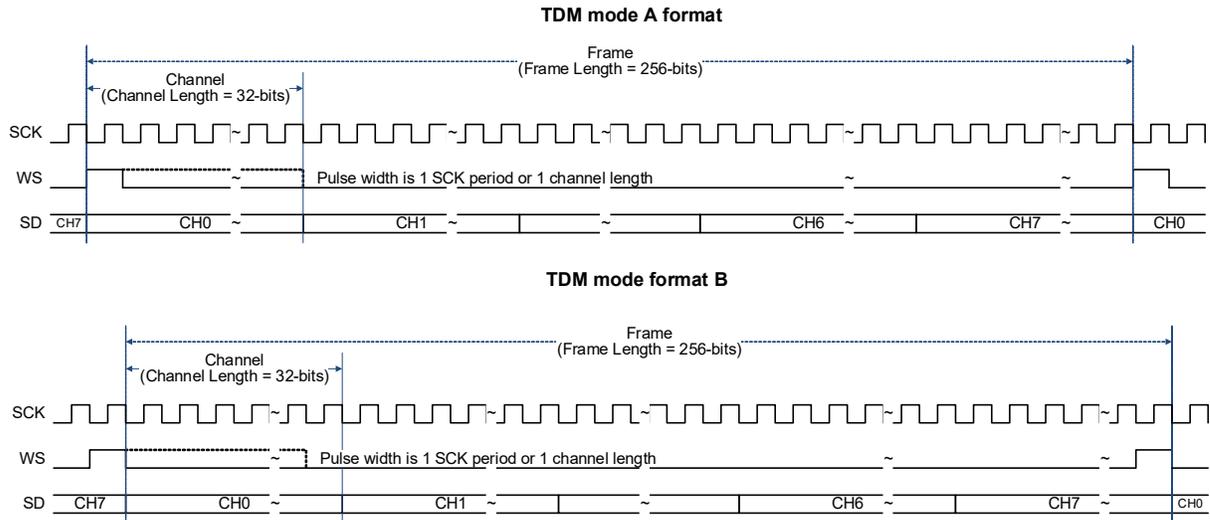


### 20.3.3 Time Division Multiplexed (TDM) Format

Figure 20-4 shows the timing diagrams for the two types of TDM formats supported by the I<sup>2</sup>S block. The differences between the standard I<sup>2</sup>S/LJ formats and the TDM format are as follows:

- Standard I<sup>2</sup>S/LJ formats support only two channels (left/right) per frame, while TDM format supports up to eight channels per frame.
- In the TDM format, channel length for all eight channels is fixed at 32 bits. In the standard I<sup>2</sup>S/LJ formats, the channel length is configurable. The word length per channel is configurable similar to the standard I<sup>2</sup>S and the data is also transmitted most significant bit first. Similar to I<sup>2</sup>S, when the word length per channel is less than the 32-bit channel length for Tx block, the OVHDATA bit in the I2S\_TX\_CTL register is used to fill the unused least significant channel data bits with either all zeros or all ones.
- In the TDM format, all eight channels of data are always present in a frame, and thus the frame width is fixed at 256 bits. You have the option to configure the number of active channels in a frame by configuring the CH\_NR bits in the I2S\_TX\_CTL register. In the standard I<sup>2</sup>S/LJ format, the CH\_NR should always be configured for two channels. The number of active channels in the TDM format can be less than or equal to eight channels. The unused (inactive) channels always follow the active channels in a frame. As an example, if CH\_NR is set for four channels, CH0 to CH3 are the active channels and CH4 to CH7 are the unused channels. The OVHDATA bit in the I2S\_TX\_CTL register is used to fill the unused channels with either all zeros or all ones.
- The pulse width of the word select (WS) signal in the TDM format can be configured to be either one bit clock (sck) wide or one channel wide. The selection is made using the WS\_PULSE bit in the I2S\_TX\_CTL register. The pulse width is fixed to one channel width in the I<sup>2</sup>S/LJ format.
- Two types of TDM formats are supported. In TDM mode A, the WS rising edge signal to signify the start of frame coincides with the start of CH0 data. In TDM mode B, the WS rising edge signal to signify the start of frame is one bit clock (sck) early, relative to the start of CH0 data (coincides with the last bit of the previous frame). The selection between the two TDM formats is made using the I2S\_MODE bits in the I2S\_TX\_CTL register.

Figure 20-4. TDM Digital Audio Interface Format



## 20.4 Clocking Features

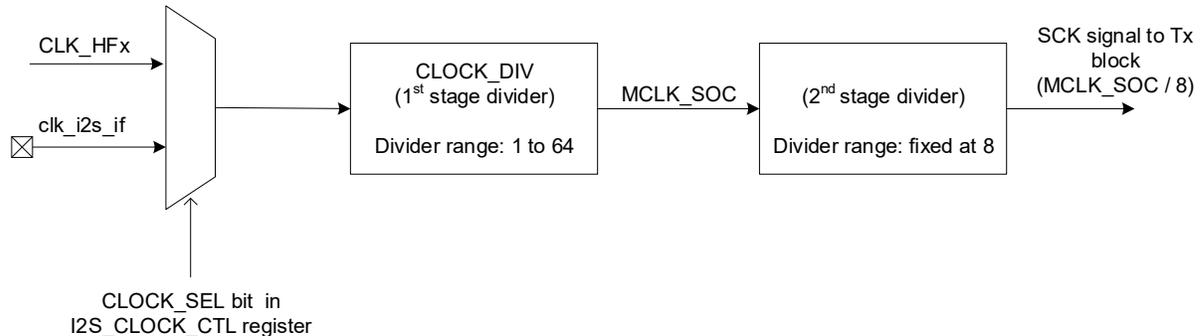
The I<sup>2</sup>S unit has three clock inputs.

Table 20-2. Clock Inputs

| Signal        | DESCRIPTION   |
|---------------|---|
| clk_sys_i2s   | System clock. This clock is used for control, status and interrupt registers.   |
| clk_audio_i2s | I <sup>2</sup> S internal clock. This clock is used for I <sup>2</sup> S transmitter (Tx) block; it is asynchronous with the clk_sys_i2s. This clock is connected to the CLK_HF[1] high-frequency clock in the device. Refer to the <a href="#">Clocking System chapter on page 88</a> for more details on high frequency clocks. |
| clk_i2s_if    | I <sup>2</sup> S external clock. This clock is provided from an external I <sup>2</sup> S bus host through a port pin. It is used in place of the clk_audio_i2s clock to synchronize I <sup>2</sup> S data to the clock used by the external I <sup>2</sup> S bus host.   |

Figure 20-5 shows the clocking divider structure in the I<sup>2</sup>S block. The sck and ws signals are generated either using the clk\_audio\_i2s internal clock or the clk\_i2s\_if external clock. Refer to the [device datasheet](#) for the port pin assignment of clk\_i2s\_if clock. The CLOCK\_SEL bit in the I2S\_CLOCK\_CTL register controls the selection between internal and external clocks.

Figure 20-5. Clocking Divider Structure



There are two stages of clock dividers in the I<sup>2</sup>S block as follows:

- The first stage clock divider is used to generate the internal I<sup>2</sup>S master clock (MCLK\_SOC). The input clock to the first stage divider is either clk\_audio\_i2s or clk\_i2s\_if. The first stage clock divider is configured using the CLOCK\_DIV bits in I2S\_CLOCK\_CTL register. Divider values from 1 to 64 are supported.
- The second stage clock divider is used to generate the sck signals. The input clock is the output from the first stage clock divider. This divider value is fixed at '8' (FTX\_SCK = FMCLK\_SOC/8). The word select (ws) signal frequency depends on the sck frequency, and the configured channel length value.

Table 20-3 gives an example of the clock divider settings for operating the I<sup>2</sup>S block at the standard sampling rates in the standard I<sup>2</sup>S format. Note that the first stage divider values in the table are the register field values – the actual divider values are one more than the configured register values as explained in the clock divider section. Refer to the [device datasheet](#) for details on maximum values of SCK frequency, and the output sampling rates.

Table 20-3. I<sup>2</sup>S Divider Values for Standard Audio Sampling Rates in Standard I<sup>2</sup>S Format

| Sampling Rate (SR) (kHz) | WORD_LEN (bits) | SCK (2*WORD_LEN*SR) (MHz) | CLK_HF1 (or clk_i2s_if) (MHz) | (CLK_HF[1])/SCK (Total Divider Ratio) | CLK_CLOCK_DIV (First Divider) | Second Stage Divider (Fixed at 8) |
|--------------------------|-----------------|---------------------------|-------------------------------|---------------------------------------|-------------------------------|-----------------------------------|
| 8                        | 32              | 0.512                     | 49.152                        | 96                                    | 11                            | 8                                 |
| 16                       | 32              | 1.024                     |                               | 48                                    | 5                             |                                   |
| 32                       | 32              | 2.048                     |                               | 24                                    | 2                             |                                   |
| 48                       | 32              | 3.072                     |                               | 16                                    | 1                             |                                   |
| 44.1                     | 32              | 2.8224                    | 45.1584                       | 32                                    | 3                             |                                   |

## 20.5 FIFO Buffer

The I<sup>2</sup>S block has a FIFO buffer for the Tx block. The ordering format of the channel data in Tx FIFO depends on the configured digital audio format. This ordering format should be considered when writing to the Tx FIFO. In the standard I<sup>2</sup>S and LJ digital audio formats, the ordering of the data is (L, R, L, R, L, ...) where L refers to the left channel data and R refers to the right channel data. In the TDM format with the number of active channels set to four, the data order will be (CH0, CH1, CH2, CH3, CH0, CH1, CH2, CH3, CH0,.....). If the number of active channels is set to eight, the cycle will repeat after CH0–CH7 data.

**I<sup>2</sup>S Tx FIFO:** The I<sup>2</sup>S Tx block has a hardware FIFO of depth 256 elements where each element is 32-bit wide. In addition to this 256-element FIFO, the I<sup>2</sup>S block has an internal transmit buffer that can store four 32-bit data to be transmitted. This four-element buffer is used as an intermediary to hold data to be transferred on the I<sup>2</sup>S bus, and is not exposed to the AHB BUS interface.

The TX FIFO can be paused by setting the TX\_PAUSE bit in I2S\_CMD. When the TX\_PAUSE bit is set, the data sent over I<sup>2</sup>S is "0", instead of TX FIFO data. To resume normal operation, the TX\_PAUSE bit must be cleared.

The I2S\_TX\_FIFO\_CTL register is used for FIFO control operations. The TRIGGER\_LEVEL bits in the I2S\_TX\_FIFO\_CTL register can be used to generate a transmit trigger event when the Tx FIFO has less entries than the value configured in the TRIGGER\_LEVEL bits.

The FIFO freeze operation can be enabled by setting the FREEZE bit in the I2S\_TX\_FIFO\_CTL register. When the FREEZE bit is set and the Tx block is operational (TX\_START bit in I2S\_CMD is set), hardware reads from the Tx FIFO do not remove the FIFO entries. Also, the Tx FIFO read pointer will not be advanced. Any writes to the I2S\_TX\_FIFO register will increment the Tx FIFO write pointer; when the Tx FIFO becomes full, the internal write pointer stops incrementing. The freeze operation may be used for firmware debug purposes. This operation is not intended for normal operation. To return to normal operation after using the freeze operation, the I<sup>2</sup>S must be reset by clearing the TX\_ENABLED bit in the I2S\_CTL register, and then setting the bit again.

The CLEAR bit in the I2S\_TX\_FIFO\_CTL register is used to clear the Tx FIFO by resetting the read/write pointers associated with the FIFO. Write accesses to the Tx FIFO using the I2S\_TX\_FIFO\_WR or I2S\_TX\_FIFO\_WR\_SILENT registers are not allowed while the CLEAR bit is set.

The I2S\_TX\_FIFO\_STATUS register provides FIFO status information. This includes number of used entries in the Tx FIFO and the current values of the Tx FIFO read/write pointers. This register can be used for debug purposes. The I<sup>2</sup>S Tx FIFO read pointer is updated whenever the data is transferred from the Tx FIFO to the internal transmit buffer. Tx FIFO write pointer is updated whenever the data is written to the I2S\_TX\_FIFO\_WR register, through CPU.

For Tx FIFO data writes using the CPU, the hardware can be used to trigger an interrupt event for any of the FIFO conditions such as TX\_TRIGGER, TX\_NOT\_FULL, and TX\_EMPTY. As part of the interrupt handler, the CPU can write to the I2S\_TX\_FIFO\_WR register. The recommended method is to write (256 - TRIGGER\_LEVEL) words to the I2S\_TX\_FIFO\_WR register every time the TX\_TRIGGER interrupt event is triggered. In addition, interrupt events can be generated for FIFO overflow/underflow conditions.

The data in the I2S\_TX\_FIFO is always right-aligned. The I2S\_TX\_FIFO\_WR format for different word length configurations is provided in Figure 20-6.

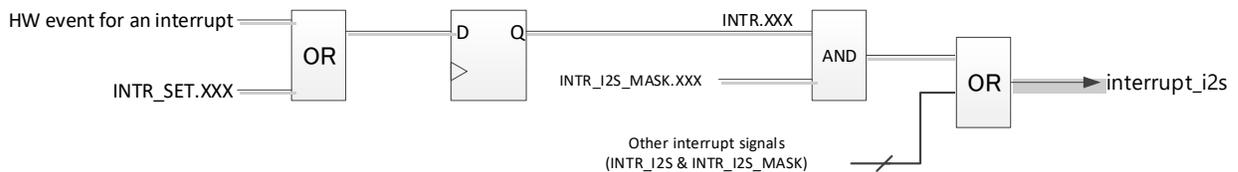
Figure 20-6. I2S\_TX\_FIFO\_WR Register Format for Different Word Lengths

|                           |  | write data format of I2S_TX_FIFO_WR |    |    |    |    |    |    |    |    |    |     |    |    |    |     |    |     |    |     |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |     |
|---------------------------|--|-------------------------------------|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----|----|-----|----|-----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|-----|
|                           |  | 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 |   |   |   |     |
| Word Length = 24-bit mode |  | fixed "0"                           |    |    |    |    |    |    |    |    |    | MSb | 23 | 22 | 21 | 20  | 19 | 18  | 17 | 16  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | LSb |
| Word Length = 20-bit mode |  | fixed "0"                           |    |    |    |    |    |    |    |    |    |     |    |    |    | MSb | 19 | 18  | 17 | 16  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | LSb |
| Word Length = 18-bit mode |  | fixed "0"                           |    |    |    |    |    |    |    |    |    |     |    |    |    |     |    | MSb | 17 | 16  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | LSb |
| Word Length = 16-bit mode |  | fixed "0"                           |    |    |    |    |    |    |    |    |    |     |    |    |    |     |    |     |    | MSb | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | LSb |

## 20.6 Interrupt Support

The I<sup>2</sup>S block has one interrupt output signal that goes to the interrupt controller in the CPU. Refer to the [Interrupts chapter on page 52](#) for details on the vector number of the I<sup>2</sup>S interrupt and the procedure to configure the interrupt priority, vector address, and enabling/disabling. The I<sup>2</sup>S interrupt can be triggered under any of the following events - TX\_TRIGGER, TX\_NOT\_FULL, TX\_EMPTY, TX\_OVERFLOW, TX\_UNDERFLOW, TX\_WD. Each of the interrupt events can be individually enabled/disabled to generate an interrupt condition. The I2S\_INTR\_MASK register is used to enable the required events by writing '1' to the corresponding bit. Irrespective of the INTR\_MASK settings, if any of the events occur, the corresponding event status bit will be set by the hardware in the I2S\_INTR register. The I2S\_INTR\_MASKED register is the bitwise AND of the I2S\_INTR\_MASK and I2S\_INTR registers. The final I<sup>2</sup>S interrupt signal is the logical OR of all the bits in the I2S\_INTR\_MASKED register. So only those events that are enabled in the I2S\_INTR\_MASK register are propagated as interrupt events to the interrupt controller. Interrupts can also be triggered in software by writing to the corresponding bits in I2S\_INTR\_SET register. Figure 20-7 illustrates the interrupt signal generation.

Figure 20-7. Interrupt Signal Generation



In the interrupt service routine (ISR), the I2S\_INTR\_MASKED register should be read to know the events that triggered the interrupt event. Multiple events can trigger the interrupt because the final interrupt signal is the logical OR output of the events. The ISR should do the tasks corresponding to each interrupt event that was triggered. At the end of the ISR, the value read in the I2S\_INTR\_MASKED register earlier should be written to the I2S\_INTR register to clear the bits whose interrupt events were processed in the ISR. Unless the bits are not cleared by writing '1' to the I2S\_INTR register, the interrupt signal will always be high. A dummy read of the I2S\_INTR register should be done for the earlier register write to take effect.

# 21. CRYPTO



This chapter explains the PSoC 4 CRYPTO module that includes AES block cipher, SHA hash, CRC, pseudo random number generation, and true random number generation functionality. In the PSoC 4 MCU, cryptographic functions varies by part number. Refer to the [device datasheet](#) to determine available cryptographic function on the device.

## 21.1 Features

The PSoC 4 CRYPTO block provides the cryptographic functions and offers these features:

- AES functionality (block cipher), per FIPS 197 standard
  - Forward block cipher (plaintext (input text) to ciphertext (encrypted text)) with 128/192/256-bit key
  - Inverse block cipher (ciphertext (encrypted text) to plaintext (input text)) with 128/192/256-bit key
- SHA functionality (hash), per FIPS 180-4 standard
  - SHA1
  - SHA224, SHA256
- CRC functionality
  - Programmable polynomial of up to 32-bits
- Pseudo random number generator (PRNG)
- True random number generator (TRNG)

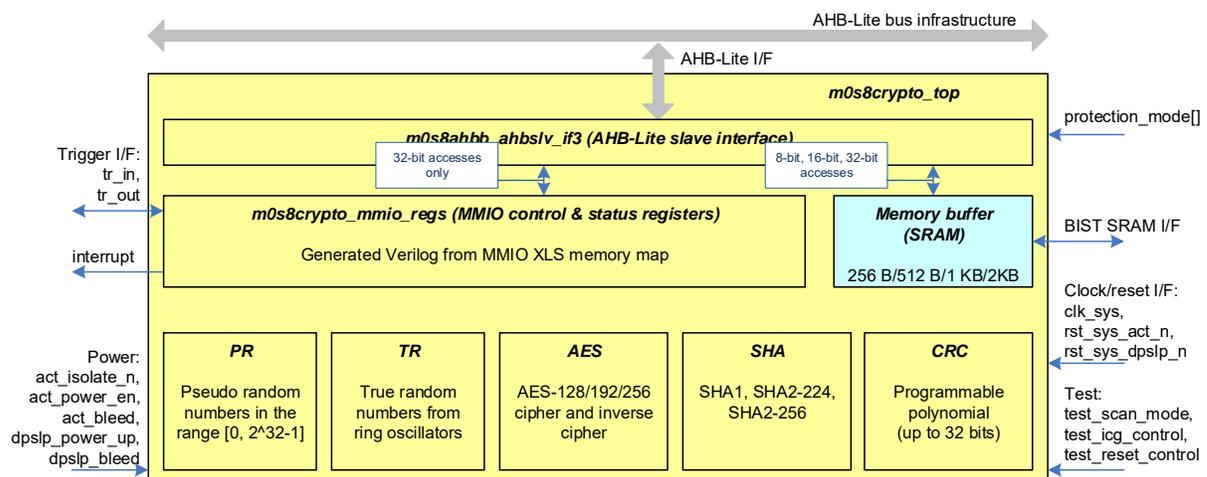
**Note** The CRYPTO module can only execute one of these functions at any given time.

The CRYPTO block is connected to the bus infrastructure as an AHB-Lite slave; it can receive AHB-Lite transfers (from the CPU or another bus master), but it cannot initiate AHB-Lite transfers.

## 21.2 Overview

Figure 21-1 illustrates an overview of the CRYPTO module.

Figure 21-1. CRYPTO Block Overview



The CRYPTO block has the following components:

- A standard AHB-Lite slave interface.
- A component that contains the MMIO control and status registers.
- A trigger interface allows to trigger from and trigger to peripheral.
- A memory buffer, for internal operation operand data
- A Pseudo Random (PR) number generator component.
- A True Random (TR) number generator component.
- An Advanced Encryption Standard (AES) component, provides AES block cipher functionality.
- A Secure Hash Algorithm (SHA) component, provides SHA hash functionality.
- A Cyclic Redundancy Check (CRC) component, provides CRC functionality.

## 21.3 Trigger Interface

CRYPTO block operation can be software-controlled by MMIO registers or hardware-controlled by trigger interface. In both cases, the MMIO registers must be configured as per the operation to be performed.

In case of hardware-controlled operation, the rising edge of the input trigger signal starts an operation. When the operation completes, a pulse is generated on the output trigger signal to indicate operation completion.

**Note** Output trigger can be generated for both software initiated operation and hardware initiated operation.

## 21.4 Memory Buffer

The CRYPTO block uses registers to capture its MMIO control and status information and uses SRAMs to capture its operand data. The memory buffer is implemented using system SRAM and maintain their information in DeepSleep power mode (only SRAM array is powered) also. The SRAM memory region size is 2 KB.

The operand data in the memory buffer are under software control and specified by SRC\_CTL0, SRC\_CTL1, DST\_CTL0 and DST\_CTL1 registers. SRC\_CTL0 and SRC\_CTL1 are used for source operands and DST\_CTL0 and DST\_CTL1 are used for the destination operands.

For example, an AES block cipher operation has two source operands:

- SRC\_CTL0 specifies the offset of the start key (symmetric key).
- SRC\_CTL1 specifies the offset of the plaintext (input text).

An AES block cipher operation has two destination operands:

- DST\_CTL0 specifies the offset of the last round key.
- DST\_CTL1 specifies the offset of the ciphertext (encrypted text).

Only a single operation can be performed at a time (for example, it is not possible to perform an AES block cipher and SHA hash function simultaneously). However, it is possible to write the source operand data of the next operation in the memory buffer, when the current operation is active.

**Note** MMIO access to the SRAM memory has higher priority over the operation access to the SRAM memory.

### 21.4.1 Access Control

The CRYPTO block's memory buffer is a memory region with CPU *execution* mode restrictions. The CPU is either in privileged execution mode or user execution mode.

- In privileged execution mode, the CPU can access all MMIO registers and memory regions.
- In user execution mode, the CPU can only access user MMIO registers and user memory regions. The CPU cannot access privileged MMIO registers and privileged memory regions.

The memory buffer is partitioned under control of a (privileged) MMIO register. Typically, during chip boot, the partition boundary is copied from the flash supervisory rows into the MMIO register. The partitioning results in:

- A privileged memory region of the memory buffer.
- A user memory region of the memory buffer.

The memory buffer's privileged and user memory regions are subject to the chip protection mode and CPU execution mode access restrictions.

Table 21-1 summarizes the CPU and DAP access to the IP's privileged and user MMIO registers and memory regions.

Table 21-1. CPU and DAP Access Control to IP's MMIO Registers and Memory Regions

| Protection Mode | CPU Privileged Execution Mode | CPU User Execution Mode | DAP       |
|-----------------|-------------------------------|-------------------------|-----------|
| VIRGIN          | Access                        | Access                  | Access    |
| OPEN            | Access                        | User Only               | User Only |
| PROTECTED       | Access                        | User Only               | No Access |
| KILL            | Access                        | User Only               | No Access |
| BOOT            | Access                        | Access                  | No Access |

## 21.5 Pseudo Random Number Generator

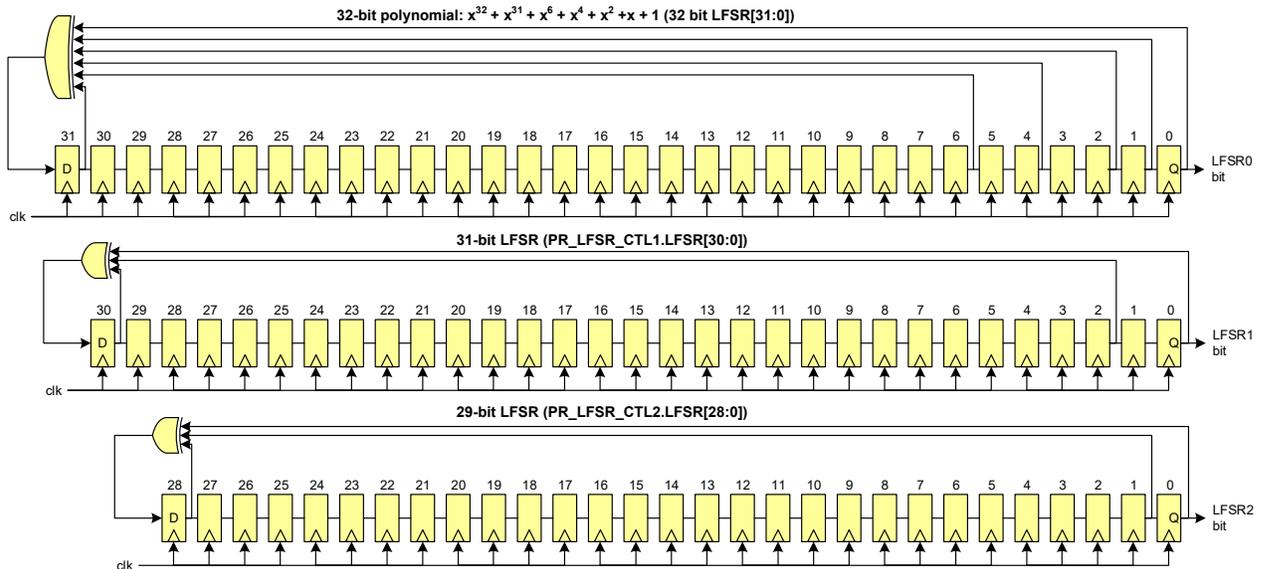
The pseudo random number generator component generates pseudo random numbers from 1 to 32 bits value controlled by the register PR\_CTL.MAX. The generator is based on three Fibonacci based Linear Feedback Shift Registers (LFSRs).

The following three irreducible fixed polynomials (with minimum feedback) are used:

- 32-bit polynomial:  $x^{32} + x^{30} + x^{26} + x^{25} + 1$
- 31-bit polynomial:  $x^{31} + x^{28} + 1$
- 29-bit polynomial:  $x^{29} + x^{27} + 1$

Figure 21-2 illustrates the LFSR functionality.

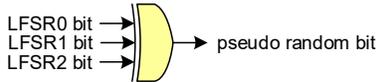
Figure 21-2. LFSR Function



Software initializes the LFSRs with non-zero seed values. The MMIO registers PR\_LFSR\_CTL0, PR\_LFSR\_CTL1, and PR\_LFSR\_CTL2 are provided for this purpose. At any time, the state of these MMIO registers can be read to retrieve the state of the LFSRs. The 32-bit LFSR generates a repeating bit sequence of  $2^{32} - 1$  bits, the 31-bit LFSR generates a repeating bit sequence of  $2^{31} - 1$  and the 29-bit LFSR generates a repeating bit sequence of  $2^{29} - 1$ .

The final pseudo random bit is the XOR of the three bits that are generated by the individual LFSRs.

Figure 21-3. Pseudo Random Bit Generation from LFSR



As the numbers  $2^{32}-1$ ,  $2^{31}-1$  and  $2^{29}-1$  are relatively prime, the XOR output is a repeating bit sequence of roughly  $2^{32+31+29}$ . A pseudo random number of  $n$  bits "pr[n-1:0]" uses  $n$  pseudo random bits from the pseudo random number generator. The pseudo random number generator component generate the result from 1 to 32 bits value controlled by the register PR\_CTL.MAX.

To generate a pseudo random number result, the following calculation is performed.

```
MAX_PLUS1[32:0] = PR_CTL.MAX[31:0] + 1;
product[63:0] = MAX_PLUS1[32:0] * pr[32:1] + PR_CTL.MAX[31:0] * pr[0];
result = product[63:32];
```

The result is provided through MMIO register PR\_RESULT.

### 21.5.1 Configuring PRNG

Pseudo random number generation is under software control. It involves the following steps:

1. Enable the IP.
2. Initialize the LFSRs with non-zero seed values.
3. Configure the range [0, PR\_CTL.MAX[31:0]].
4. Start a pseudo random number generation operation.
5. Retrieve the results when the operation has completed. Waiting for the operation to be completed can be done by either polling or through an ISR.

## 21.6 True Random Number Generator (TRNG)

The Arm Cortex-M0+ CPU in PSoC 4100S Max device supports the true random number generator component (TRNG) component. The TRNG component generates true random numbers. The TRNG number generation can be limited by the programmable bit size of TRNG.

The TRNG relies on up to six ring oscillators to provide physical noise sources. A ring oscillator consists of a series of inverters connected in a feedback loop to form a ring. Due to (temperature) sensitivity of the inverter delays, jitter is introduced on a ring's oscillating signal. The jittered oscillating signal is sampled to produce a "digitized analog signal" (DAS). This is done for all multiple ring oscillators.

To increase entropy and to reduce bias in DAS bits, the DAS bits are further post-processed. Post-processing involves two steps:

- An optional reduction step (up to six DAS bits and over one or multiple DAS bit periods) to increase entropy.
- An optional "von Neumann correction" step to reduce a '0' or '1' bias.
  - This correction step processes pairs of reduction bits as produced by the previous step. Given two reduced bits  $r_0$  and  $r_1$  (with  $r_0$  being produced before  $r_1$ ), the correction step is defined as follows:
    - $\{r_0, r_1\} = \{0, 0\}$ : no bit is produced
    - $\{r_0, r_1\} = \{0, 1\}$ : a '0' bit is produced (bit  $r_0$ )

- $\{r0, r1\} = \{1, 0\}$ : a '1' bit is produced (bit r0)
- $\{r0, r1\} = \{1, 1\}$ : no bit is produced

The correction step only produces a bit on a '0' to '1' or '1' to '0' transition. Note that for a random input bit sequence, the correction step produces an output bit sequence of roughly  $\frac{1}{4}$  the frequency of the input bit sequence (the input reduction bits are processed in non-overlapping pairs and only half of the pair encodings result in an output bit).

Post-processing produces bit samples that are considered true random bit samples. The true random bit samples are shifted into a register, to provide random values of up to 32 bits.

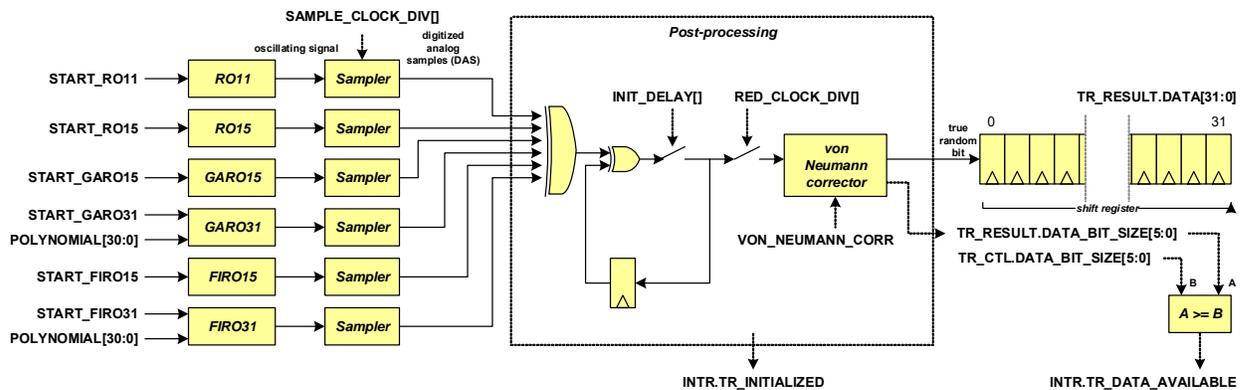
As a result of high switching activity, ring oscillators consume a significant amount of power. Therefore, when the TRNG functionality is disabled, the ring is "broken" to prevent switching. When the TRNG functionality is enabled, a ring oscillator initially has predictable behavior. However, over time, infinitesimal environmental (temperature) changes cause an increasing deviation from this predictable behavior.

- DURING the initial delay, the ring oscillator is NOT a reliable physical noise source.
- AFTER an initial delay, the same ring oscillator will show different oscillation behavior and provides a reliable physical noise source.

Therefore, the DAS bits can be dropped during an initialization period (INIT\_DELAY[]). The INIT\_DELAY field can be configured using the TRNG\_CTL0 register. It is advised to drop few samples after starting the oscillator.

Figure 21-4 illustrates an overview of the TRNG component.

Figure 21-4. TRNG Block Diagram



Note that when a ring oscillator is stopped, the synchronization logic is reset. Therefore, the ring oscillator contributes a constant '0' to the reduction step of the post-processing.

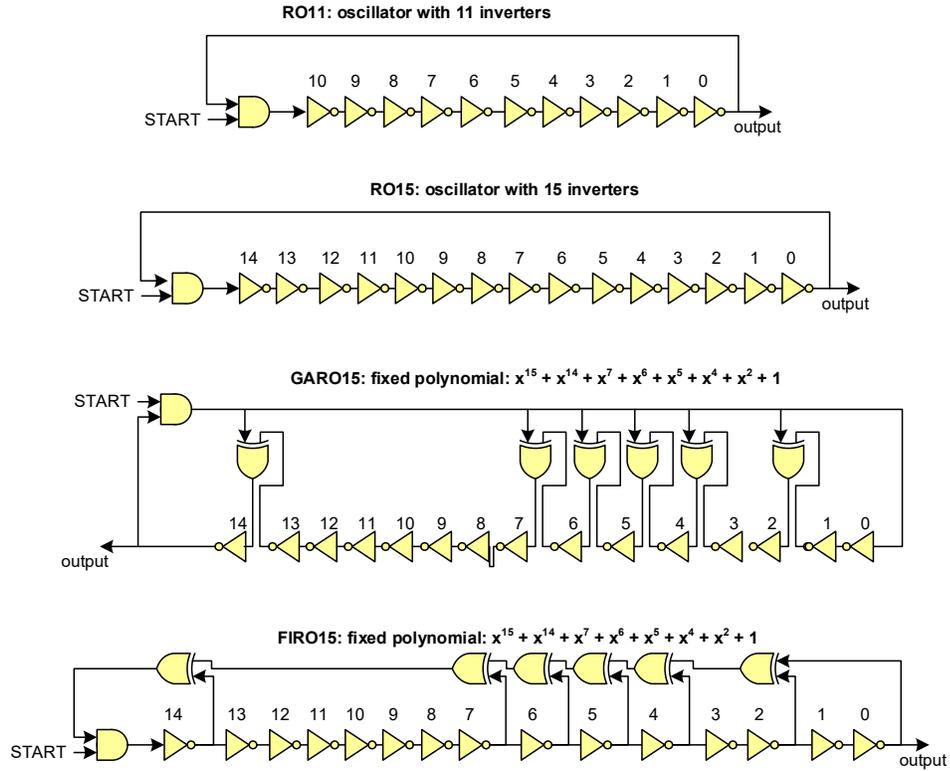
The TRNG relies on up to six ring oscillators:

- RO11: A fixed ring oscillator consisting of 11 inverters.
- RO15: A fixed ring oscillator consisting of 15 inverters.
- GARO15: A fixed Galois based ring oscillator of 15 inverters.
- GARO31: A flexible Galois based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides the flexibility in the oscillator feedback.
- FIRO15: A fixed Galois based ring oscillator of 15 inverters.
- FIRO31: A flexible Galois based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides the flexibility in the oscillator feedback.

Each ring oscillator can be started or stopped. When stopped, the ring is "broken" to prevent switching.

Figure 21-5 illustrates the schematics of the fixed ring oscillators.

Figure 21-5. Ring Oscillator Type 1



The START signals originate from the MMIO register field. The flexible Galois and Fibonacci based ring oscillators rely on programmable polynomials to specify the oscillator feedback. This allows for rings of 1, 3, 5, ..., 31 inverters (an odd number is required to generate an oscillating signal).

Figure 21-6 illustrates an overview of the Galois based ring oscillator.

Figure 21-6. Ring Oscillator Types 2

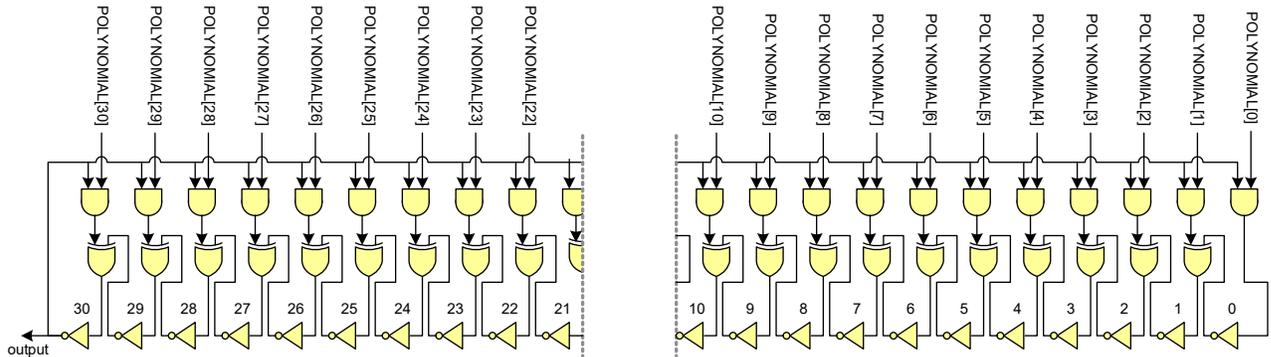
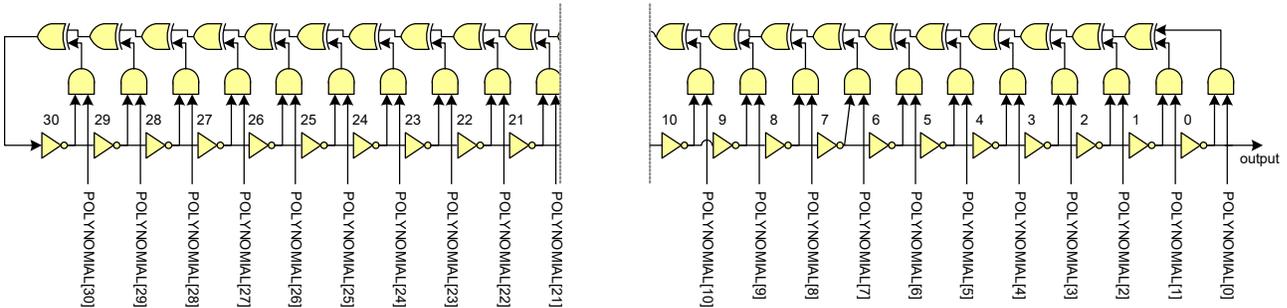


Figure 21-7 illustrates an overview of the Fibonacci based ring oscillator.

Figure 21-7. Ring Oscillator Types 3



The TRNG has a build-in health monitor that performs tests on the digitized noise source to detect deviations from the intended behavior. For example, the health monitor detects “stuck at” faults in the digitized analog samples.

The health monitor tests one out of three selected digitized bit streams:

- DAS bitstream. This is XOR of the digitized analog samples.
- RED bitstream. This is the bitstream of reduction bits
- TR bitstream. This is the bitstream of true random bits.

The health monitor performs two different tests: Repetition Count Test and Adaptive Proportion Test.

- **Repetition Count Test:** This test checks for the repetition (specified by REP\_COUNT) of the same bit value ('0' or '1') in a bit stream. A detection indicates that a specific active bit value has repeated for a pre-programmed count (specified by CUTOFF\_COUNT). The test uses a counter to maintain the number of repetitions of the active bit value in REP\_COUNT.

A detection stops the test and sets the associated interrupt status field. When the test is stopped, REP\_COUNT equals CUTOFF\_COUNT. The TRNG block can also be configured to stop on the repetition count detection.

- **Adaptive Proportion Test:** This test checks for a disproportionate occurrence of a specific bit value ('0' or '1') in a bit stream. A detection indicates that a specific active bit value has occurred a pre-programmed number of times (specified by CUTOFF\_COUNT) in a bit sequence of a specific bit window size (specified by WINDOW\_SIZE). The test uses a counter to maintain an index in the current window (specified by WINDOW\_INDEX) and a counter to maintain the number of occurrences of the active bit value (specified by OCC\_COUNT).

A detection stops the test and sets the associated interrupt status field. When the test is stopped, OCC\_COUNT equals CUTOFF\_COUNT and the WINDOW\_INDEX identifies the bit sequence index on which the detection occurred. The TRNG block can also be configured to stop on the adaptive proportion detect.

## 21.7 Advance Encryption Standard (AES)

The Arm Cortex-M0+ CPU in PSoC 4100S Max device supports advance encryption standards (AES) component that performs a block cipher or inverse block cipher as per the AES standard (FIPS 197).

- The block cipher translates a 128-bit block of plaintext (input text) data into a 128-bit block of ciphertext (encrypted text) data.
- The inverse block cipher translates a 128-bit block of ciphertext (encrypted text) data into a 128-bit block of plaintext (input text) data

AES is a symmetric block cipher, it means the cipher and inverse cipher keys are the same. The component supports 128-bit, 192-bit and 256-bit keys. The AES algorithm generates round keys "on-the-fly" from the start round key as provided by SW. Round key generation is reversed for the cipher and inverse cipher.

- The block cipher uses the symmetric key as the (start) round key for the first cipher round. The round key for second cipher round is derived from the symmetric key. The round key for the third cipher round is derived from the round key of the second cipher round, and so forth.
- The *inverse* block cipher uses the round key of the final cipher round as the (start) round key for the first inverse cipher round. The round key for the second *inverse* cipher round is derived from the round key of the first inverse cipher round, and so forth.
- The round key of the final inverse cipher round is the same as the symmetric key.

The generated round keys are only dependent on the start key of the first round (in case of the block cipher this is the symmetric key). It is possible to derive the start round key for the inverse block cipher from the symmetric key, by performing a (forward) block cipher with arbitrary plaintext (input text) data.

The number of (inverse) cipher rounds depends on the key size. [Table 21-2](#) provides the number of rounds.

Table 21-2. AES Key vs Cipher Rounds

| AES Key Size | Cipher Rounds (inverse) |
|--------------|-------------------------|
| 128-bit      | 10 rounds               |
| 192-bit      | 12 rounds               |
| 256-bit      | 14 rounds               |

All AES operand data is provided by the memory buffer. For example, an AES block cipher operation has two source operands:

- SRC\_CTL0 specifies the offset of the start key (symmetric key).
- SRC\_CTL1 specifies the offset of the plaintext (input text).

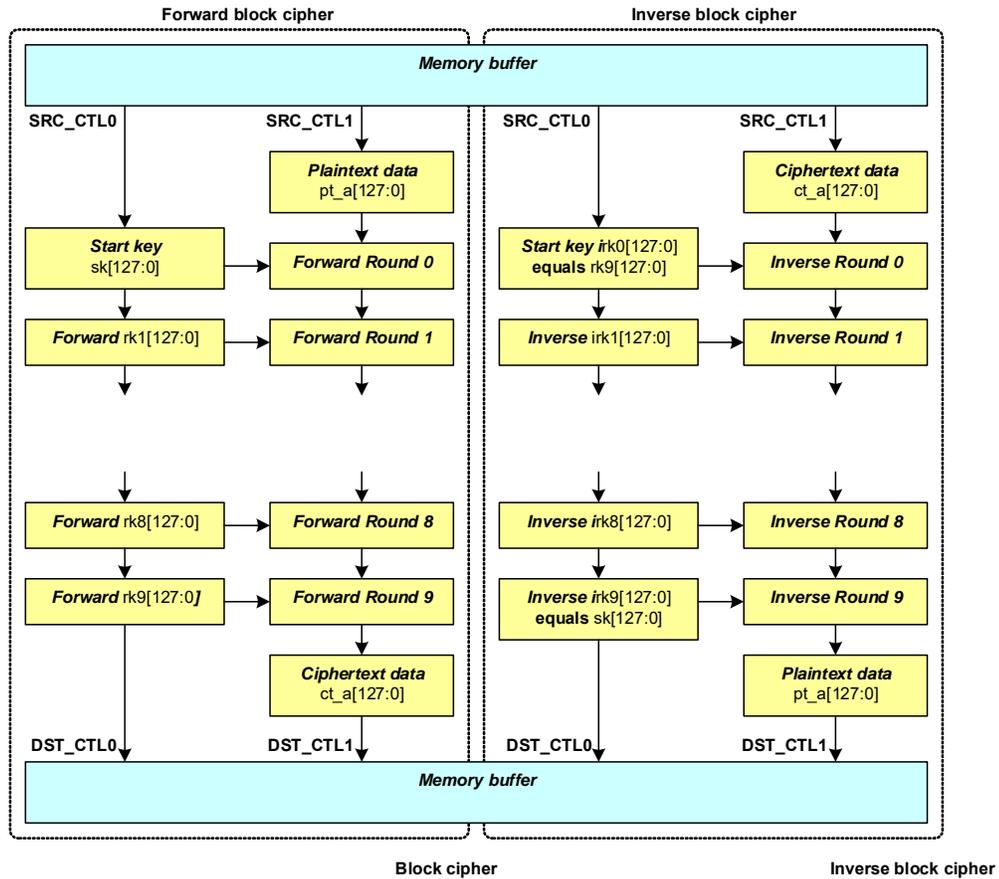
An AES block cipher operation has two destination operands:

- DST\_CTL0 specifies the offset of the last round key.
- DST\_CTL1 specifies the offset of the ciphertext (encrypted text).

**Note** To reduce space needed in the buffer it can overwrite the "start key" with the "last round key". Similarly, it is also allowed to overwrite the input data with the output data.

Figure 21-8 illustrates the AES-128 cipher.

Figure 21-8. AES Block Diagram



The block diagrams shows how a forward block cipher translates a plaintext (input text) “pt\_a[127:0]” into ciphertext (encrypted text) “ct\_a[127:0]” using a 128-bit start key “sk[127:0]”. Besides the plaintext (input text), the component produces the round key of the last cipher round “rk9[127:0]”. If the round key of the last cipher round “rk9[127:0]” is used as the start key for an inverse block cipher on ciphertext (encrypted text) “ct\_a[127:0]”, the original plaintext (input text) “pt\_a[127:0]” is produced. Furthermore, the round key of the last inverse cipher round is the same as the forward block cipher key “sk[127:0]”.

### 21.7.1 Modes of Operation

AES operation modes are defined by the NIST (National Institute of Standards and Technology) publication 800-30A. It recommends five modes of operation for use with an underlying symmetric key block cipher algorithm:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

The device supports ECB mode of operation i.e., the core AES algorithm. Other modes of operation involve pre and/post XOR operations of an initialization vector and plaintext (input text)/ciphertext (encrypted text), these operations can be implemented in software to implement other AES operation modes.

## 21.7.2 Configuring AES

AES is under software control. It involves the following steps:

1. Enable the IP.
2. Initialize the memory buffer with the start key.
3. Initialize the memory buffer with plaintext (input text)/ciphertext (encrypted text) data.
4. Specify the memory buffer offsets of the source and destination operands.
5. Specify the key size.
6. Start a forward/inverse cipher operation.
7. Retrieve the results from the memory buffer when the operation has completed.

## 21.8 SHA

This device supports SHA component that performs a Secure Hash Algorithm (SHA) per the SHA standard (FIPS 180-4). The SHA algorithm calculates a fixed length hash value from a variable length message. The hash value is used to produce a message digest or signature.

It is computationally impossible to change the message without changing the signature. A given message always produces the same hash value. To prevent repetitions, a counter may be included in the message. The SHA component supports a subset of the algorithms in the SHA standard as shown in [Table 21-3](#).

Table 21-3. SHA Scheme

| Algorithm | Block Size | Word Size | Hash Size | Message Digest Size |
|-----------|------------|-----------|-----------|---------------------|
| SHA1      | 512 bits   | 32 bits   | 160 bits  | 160 bits            |
| SHA224    | 512 bits   | 32 bits   | 256 bits  | 224 bits            |
| SHA256    | 512 bits   | 32 bits   | 256 bits  | 256 bits            |

The memory buffer provides a message block on which the hash function is performed (SRC\_CTL0) and the initial hash value (SRC\_CTL1). The message block must be laid out in little endian format.

The memory buffer provides working space for message schedule round constants (DST\_CTL0). The HW derives these constants from the message. The round constants working space size depends on the algorithm as shown in [Table 21-4](#).

Table 21-4. SHA Scheme vs Constant

| Algorithm | Working Space for Round Constant |
|-----------|----------------------------------|
| SHA1      | 320 bytes                        |
| SHA224    | 256 bytes                        |
| SHA256    | 256 bytes                        |

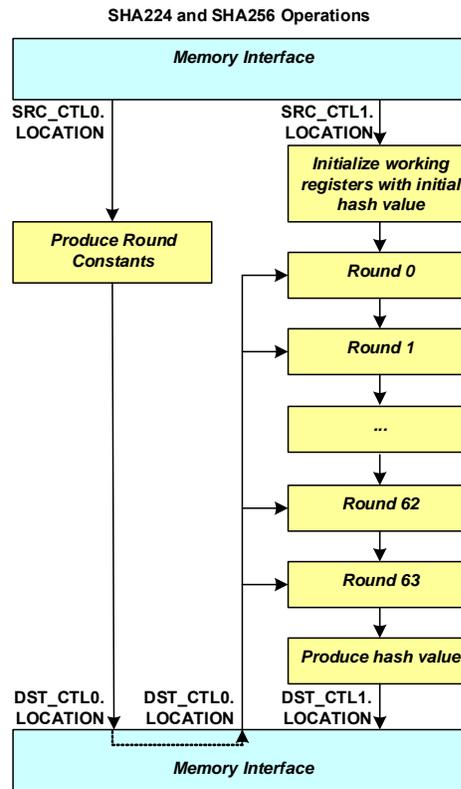
The memory buffer is used for the produced hash value (DST\_CTL1). The SHA preprocessed message consists of multiple 512-bit blocks. The SHA component processes a single 512-bit block at a time. The first SHA operation on the first message block uses the initial SHA hash value (as defined by the standard), subsequent SHA operations on successive message blocks use the produced hash value of the previous SHA operation. The initial SHA hash value is written in the memory buffer by the SW.

The SHA operation on the last message block produces the final hash value. The message digest is a subset of the final hash value (SHA224) or the complete final hash value (SHA1 and SHA256).

Note that the difference between SHA224 and SHA256 is entirely in software, i.e., they are the same for the SHA logic.

Figure 21-9 illustrates the SHA224 operation.

Figure 21-9. SHA Block Diagram



### 21.8.1 Configuring SHA

SHA is under software control. It involves the following steps:

1. Enable the IP.
2. Initialize the memory buffer with initial hash value.
3. Initialize the memory buffer with the message block.
4. Specify the memory buffer offsets of the source and destination operands.
5. Specify the hash operation.
6. Start a hash operation.
7. Retrieve the results from the memory buffer when the operation has completed.

## 21.9 CRC

The PSoC 4100S Max device supports CRC component that performs a cyclic redundancy check with a programmable polynomial of up to 32 bits.

The memory buffer provides the data on which the CRC is performed (SRC\_CTL0 specifies the offset of the data in the buffer). The data must be laid out in little endian format (least significant Byte of a multi-Byte word should be located at the lowest memory address of the word). The MMIO register field CRC\_DATA\_CTL.DATA\_SIZE[11:0] specifies the Byte size of the data. The MMIO register field CRC\_DATA\_CTL.DATA\_XOR[7:0] specifies a byte pattern with which each data Byte is XOR'd. This allows for inversion of the data Byte value. The MMIO register field CRC\_DATA\_CTL.DATA\_REVERSE allows for bit reversal of the data Byte (this provides support for serial interfaces that transfer Bytes in most-significant-bit first and least-significant bit first configurations).

The MMIO register field CRC\_POL\_CTL.POLYNOMIAL[31:0] specifies the polynomial. The polynomial specification omits the high order bit and should be left aligned.

For example, popular 32-bit and 16-bit CRC polynomials are specified as follows:

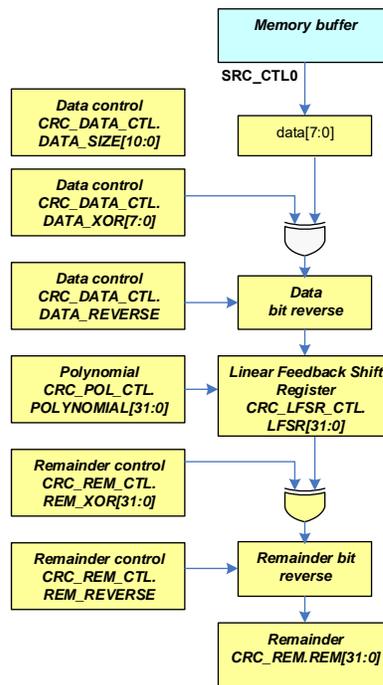
- CRC32:  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$   
CRC\_POL\_CTL.POLYNOMIAL[31:0] = 0x04c11db7
- CRC16-CCITT:  $x^{16} + x^{12} + x^5 + 1$   
CRC\_POL\_CTL.POLYNOMIAL[31:0] = (0x1021 << 16)
- CRC16:  $x^{16} + x^{15} + x^2 + 1$   
CRC\_POL\_CTL.POLYNOMIAL[31:0] = (0x8005 << 16)

The MMIO register field CRC\_LFSR\_CTL.LFSR[31:0] holds the state of the CRC calculation. Before the CRC operation, this field should be initialized with the CRC seed value.

The MMIO register field CRC\_REM.REM[31:0] holds the result of the CRC calculation, and is derived from the end state of the CRC calculation (CRC\_LFSR\_CTL.LFSR[31:0]). The MMIO register field CRC\_REM\_CTL.REM\_XOR[31:0] specifies a 32-bit pattern with which the end state is XOR'd. The MMIO register field CRC\_REM\_CTL.REM\_REVERSE allows for bit reversal of the XOR'd state.

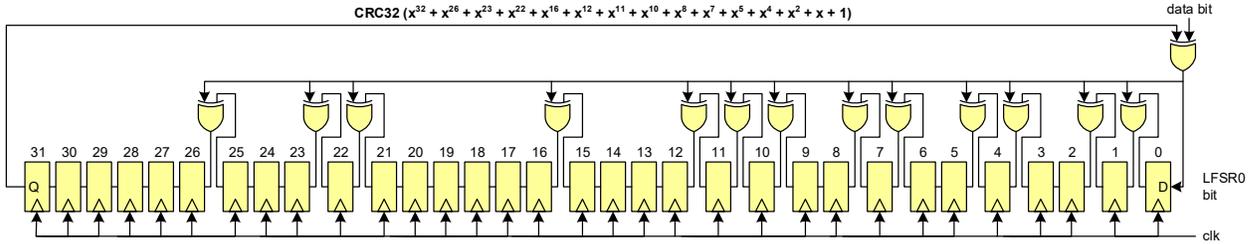
Figure 21-10 illustrates the CRC functionality.

Figure 21-10. CRC Block Diagram



The Linear Feedback Shift Register functionality operates on the LFSR state. It uses the programmed polynomial and consumes a data bit for each iteration (eight iterations are performed per cycle to provide a throughput of one data Byte per cycle). Figure 21-11 illustrates the functionality for the CRC32 polynomial ( $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ ).

Figure 21-11. CRC Polynomial



Different CRC algorithms require different seed values and have different requirements w.r.t XOR functionality and bit reversal. Table 21-5 provides the proper settings for the CRC32, CRC16-CCITT and CRC16 algorithms. The table also provides the remainder after the algorithm has been performed on a five-byte array {0x12, 0x34, 0x56, 0x78, 0x9a}.

Table 21-5. CRC Scheme Settings

| MMIO Register Field       | CRC32      | CRC 16-CCITT | CRC16      |
|---------------------------|------------|--------------|------------|
| CRC_POL_CTL.POLYNOMIAL    | 0x04c11db7 | 0x10210000   | 0x80050000 |
| CRC_DATA_CTL.DATA_REVERSE | 1          | 0            | 1          |
| CRC_DATA_CTL.DATA_XOR     | 0x00       | 0x00         | 0x00       |
| CRC_LFSR_CTL.LFSR (seed)  | 0xffffffff | 0xffff0000   | 0xffff0000 |
| CRC_REM_CTL.REM_REVERSE   | 1          | 0            | 1          |
| CRC_REM_CTL.REM_XOR       | 0xffffffff | 0x00000000   | 0x00000000 |
| CRC_REM.REM               | 0x3c4687af | 0xf8a00000   | 0x000048d0 |

### 21.9.1 Configuring CRC

CRC is under software control. It involves the following steps:

1. Enable the IP.
2. Initialize the memory buffer with the data.
3. Specify the size of the data array.
4. Specify the memory buffer offsets of the source operand (data).
5. Specify the processing of the data (XOR mask and bit reversal).
6. Specify the polynomial.
7. Specify the LFSR seed values.
8. Specify the processing of the remainder (XOR mask and bit reversal).
9. Start a CRC operation.
10. Retrieve the results from the CRC\_REM MMIO register when the operation has completed.

## 21.10 Power Modes

The VU register-file registers and memory buffer content are maintained in DeepSleep power mode and lose state in Hibernate power mode.

# Section E: Analog System

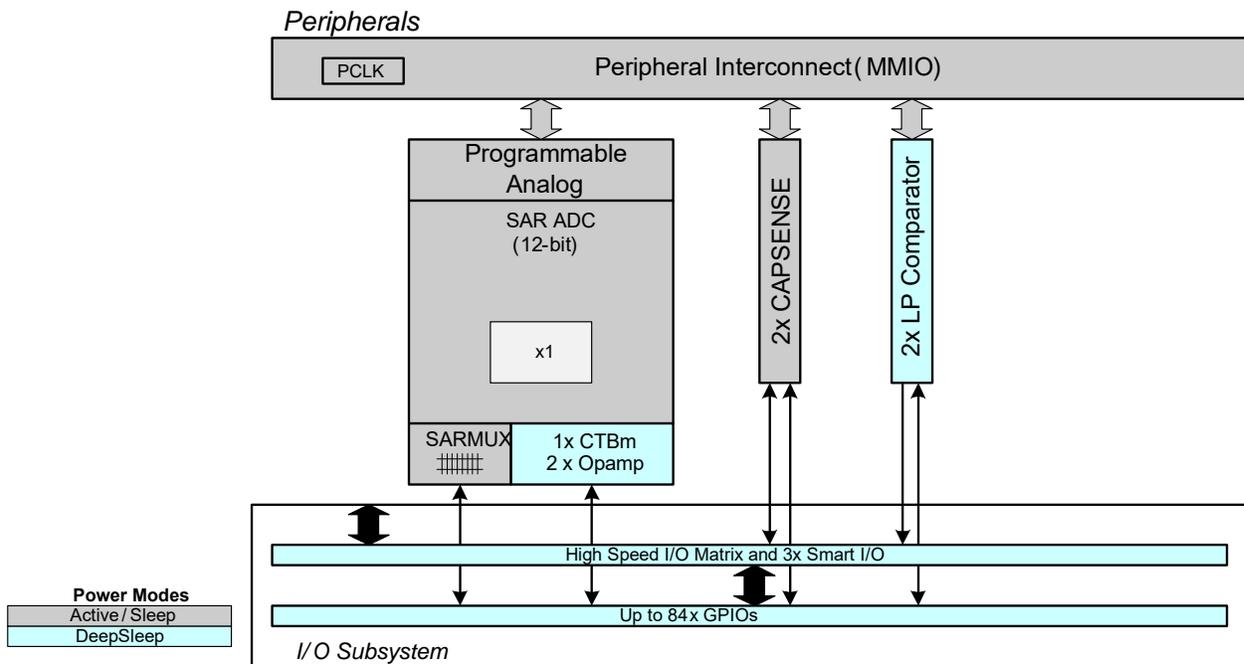


This section encompasses the following chapter:

- SAR ADC chapter on page 315
- Low-Power Comparator chapter on page 339
- Continuous Time Block mini (CTBm) chapter on page 344
- CapSense chapter on page 353
- Temperature Sensor chapter on page 354
- Analog Routing chapter on page 357

## Top Level Architecture

Analog System Block Diagram



## 22. 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 22-1](#)):

- SARMUX
- SAR ADC core
- SARREF
- SARSEQ

The SAR ADC core is a fast 12-bit ADC with sampling rate of 1 Msps. Preceding the SAR ADC is the SARMUX, which can route external pins and internal signals (AMUXBUS-A/B, CTBm, temperature sensor output) to the 16 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 result from each channel is double-buffered and a complete scan may be configured to generate an interrupt at the end of the scan. The sequencer may also be configured to flag overflow, collision, and saturation errors that can be configured to assert an interrupt.

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

### 22.1 Features

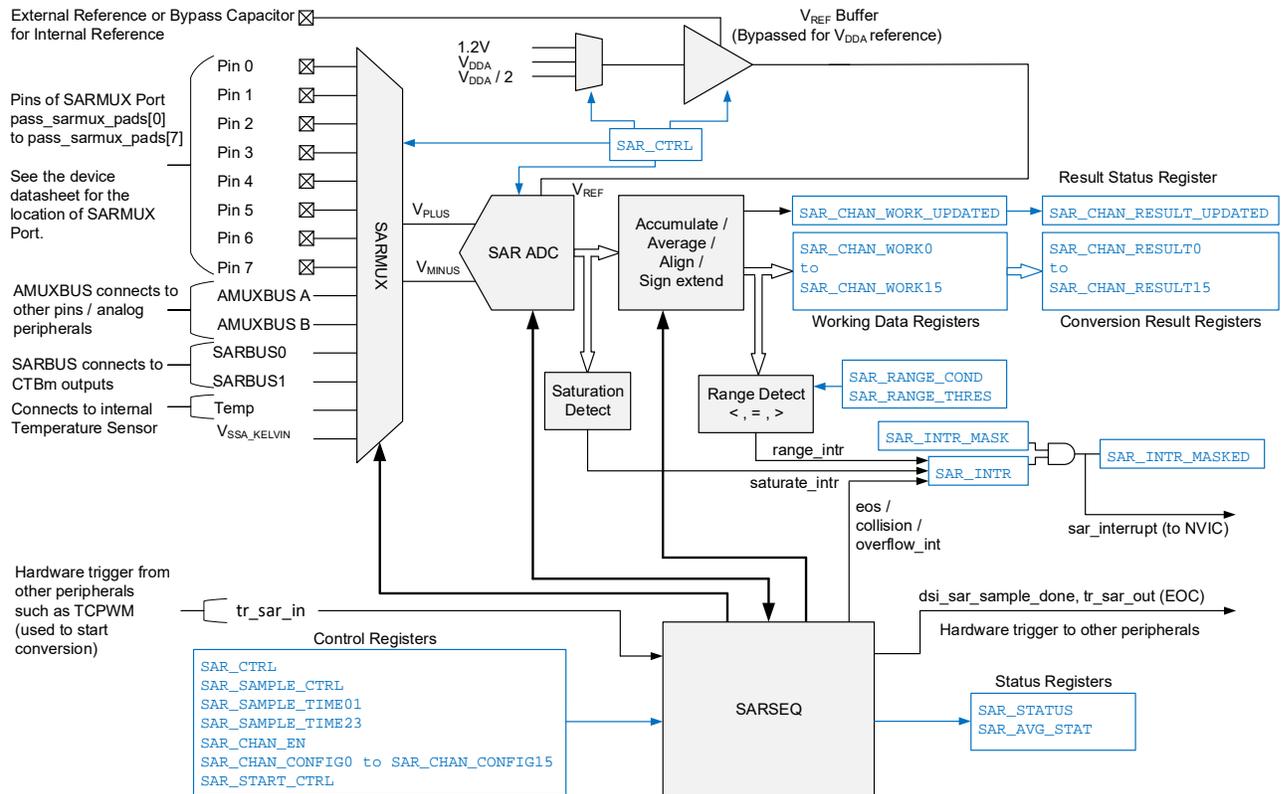
The SAR ADC block provides the following features:

- Operates across the entire device power supply range
- Maximum 1 Msps sample rate
- 16 individually configurable channels that can scan 13 unique input sources 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, CTBm comparator, low-power comparator, and by SAR end of conversion signal
  - Hardware/firmware trigger (one shot), and free-running (continuous conversion) modes
- Hardware averaging support
  - First order accumulate
  - Samples averaging from 2 to 256 (powers of 2)
- Results represented in 16-bit sign extended values

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

## 22.2 Block Diagram

Figure 22-1. Block Diagram



## 22.3 How it Works

This section includes the following contents:

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

### 22.3.1 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

#### 22.3.1.1 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  $V_{minus}$  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  $2 \times V_{REF}$ .

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.

#### 22.3.1.2 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  $V_n$  and the ADC reference is  $V_{REF}$ , the range on the positive input is  $V_n \pm V_{REF}$ . This criteria applies for both single-ended and differential modes. In single-ended mode,  $V_n$  is connected to  $V_{SSA}$ ,  $V_{REF}$  or an external input.

Note that  $V_n \pm V_{REF}$  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.

### 22.3.1.3 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 22-1. Result Data Format

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

### 22.3.1.4 Negative Input Selection

The negative input connection choice affects the voltage range, SNR, and effective resolution (Table 22-2). 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 22-2. Negative Input Selection Comparison

| Single-ended/Differential | Signed/Unsigned  | SARMUX Vminus | SARMUX Vplus Range                                | Result Register         | Maximum SNR |
|---------------------------|------------------|---------------|---|-------------------------|-------------|
| Single-ended              | N/A <sup>a</sup> | $V_{SSA}$     | $+V_{REF}$<br>$V_{SSA} = 0$                       | 0x7FF<br>0x000          | Better      |
| Single-ended              | Unsigned         | $V_{REF}$     | $+2 \times V_{REF}$<br>$V_{REF}$<br>$V_{SSA} = 0$ | 0xFFF<br>0x800<br>0     | Good        |
| Single-ended              | Signed           | $V_{REF}$     | $+2 \times V_{REF}$<br>$V_{REF}$<br>$V_{SSA} = 0$ | 0x7FF<br>0x000<br>0x800 | Good        |

Table 22-2. Negative Input Selection Comparison (continued)

| Single-ended/<br>Differential | Signed/Unsigned | SARMUX<br>Vminus | SARMUX<br>Vplus Range                                | Result Register         | Maximum SNR |
|-------------------------------|-----------------|------------------|--|-------------------------|-------------|
| Single-ended                  | Unsigned        | Vx               | Vx + V <sub>REF</sub><br>Vx<br>Vx - V <sub>REF</sub> | 0xFFF<br>0x800<br>0     | Best        |
| Single-ended                  | Signed          | Vx               | Vx + V <sub>REF</sub><br>Vx<br>Vx - V <sub>REF</sub> | 0x7FF<br>0x000<br>0x800 | Best        |
| Differential                  | Unsigned        | Vx               | Vx + V <sub>REF</sub><br>Vx<br>Vx - V <sub>REF</sub> | 0xFFF<br>0x800<br>0     | Best        |
| Differential                  | Signed          | Vx               | Vx + V <sub>REF</sub><br>Vx<br>Vx - V <sub>REF</sub> | 0x7FF<br>0x000<br>0x800 | Best        |

a. For single-ended mode with V<sub>minus</sub> connected to V<sub>SSA</sub>, conversions are effectively 11-bit because voltages cannot swing below V<sub>SSA</sub> 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<sub>REF</sub> to the negative input of the SAR ADC; then, the input range can be from 0 to 2 × V<sub>REF</sub>.

Note that single-ended conversions with V<sub>minus</sub> 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.

### 22.3.1.5 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:

$$\text{Conversion time (sar\_clk)} = \text{resolution (bit)} + 2$$

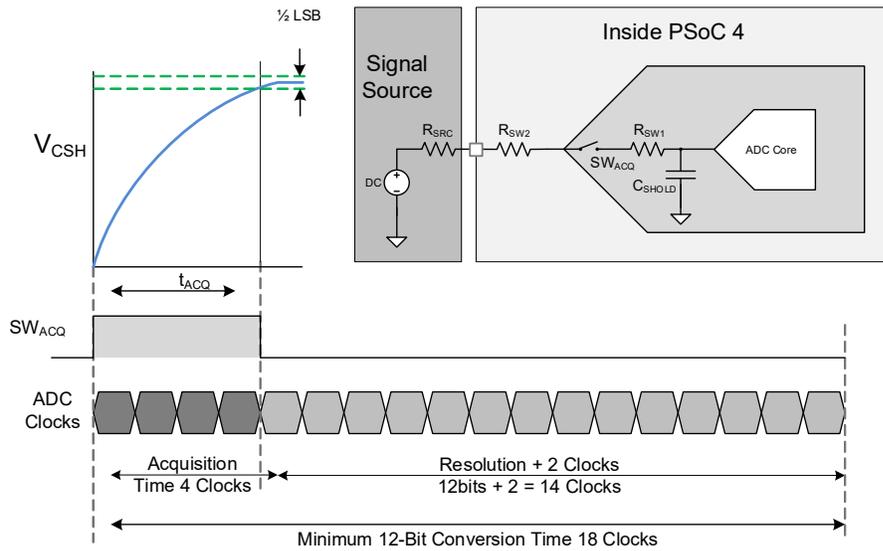
$$\text{Total acquisition and conversion time (sar\_clk)} = \text{acquisition time} + \text{resolution (bit)} + 2$$

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.

### 22.3.1.6 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 22-2. 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 22-2](#). The recommended value of acquisition time is:

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

Where:

$$C_{SHOLD} \sim 10 \text{ pF}$$

$R_{SW2} + R_{SW1} = \sim 2.2\text{k}$  when SARMUX port pin is used, with other routing paths additional resistance will be applicable, depending on the routing path (see [“Analog Routing” on page 357](#) for details).

$R_{SRC}$  = series resistance of the signal source

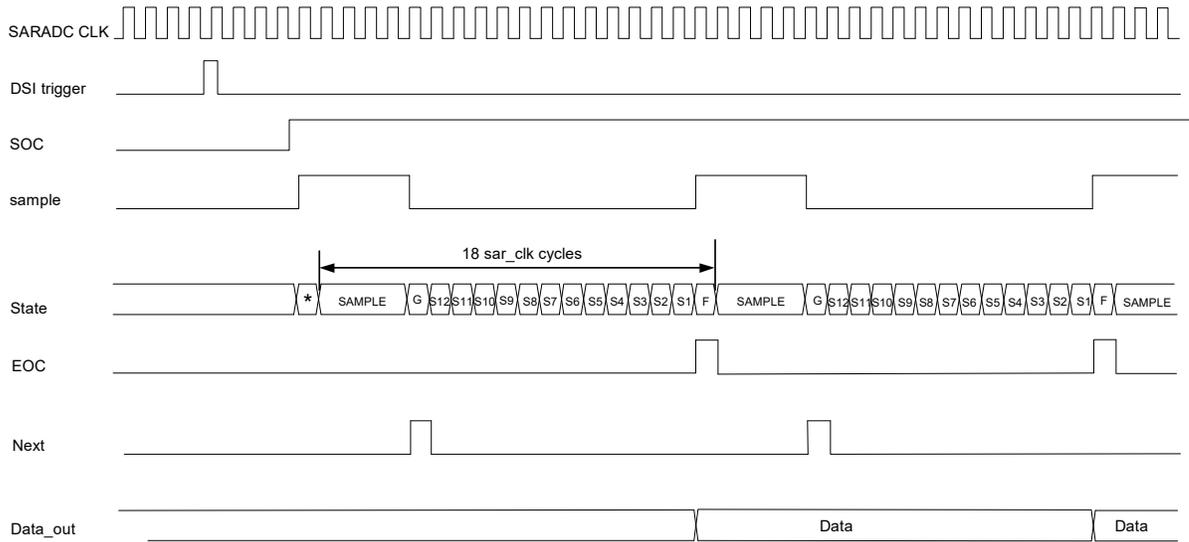
### 22.3.1.7 SAR ADC Clock

SAR ADC clock frequency must be between 1 MHz and 18 MHz, which comes from the HFCLK via a clock divider. Note that a fractional divider is not supported for 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 22-2](#)), which is 194 ns.

### 22.3.1.8 SAR ADC Timing

Figure 22-3 shows the SAR ADC timing diagram. A 12-bit resolution conversion needs 14 clocks (one bit needs one sar\_clk, plus two excess sar\_clk for G and F state). With acquisition time equal to four sar\_clk cycles by default, 18 clock sar\_clk cycles are required for total ADC acquisition and conversion. After sample (acquisition), it will output the next pulse. The SARMUX can route to another pin and signal; this will be done automatically with sequencer control (see “SARSEQ” on page 324 for details).

Figure 22-3. SAR ADC Timing

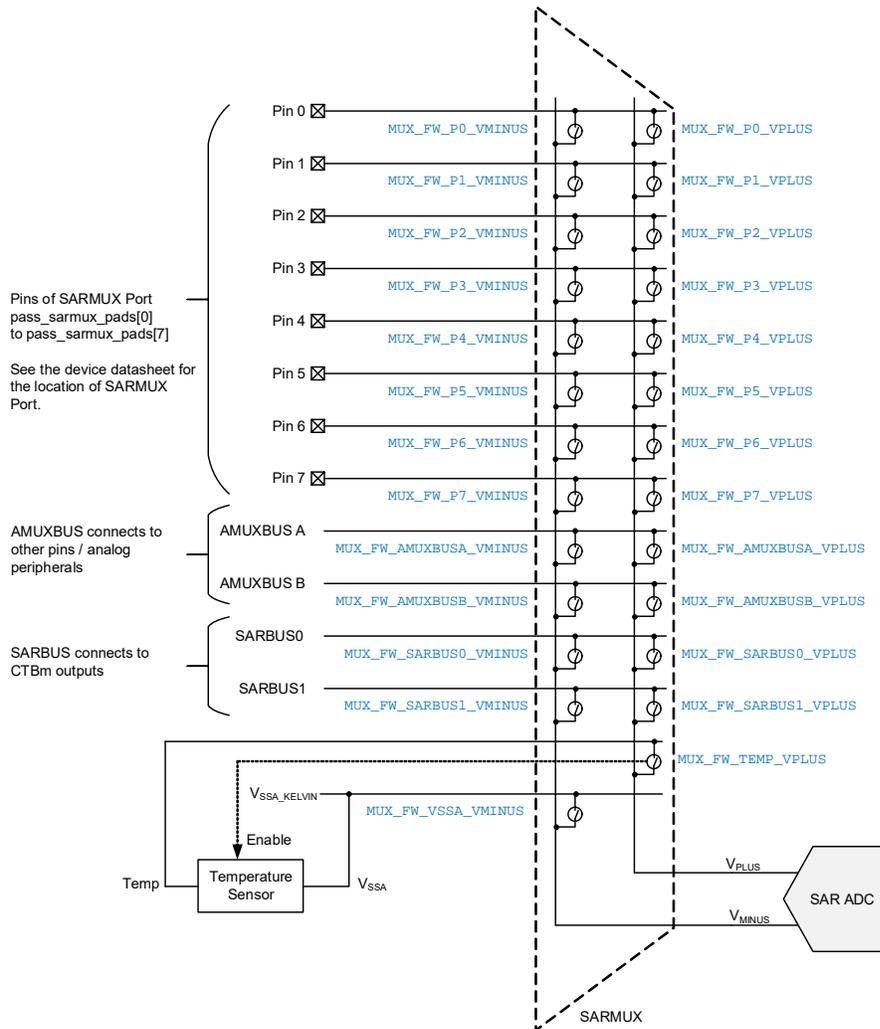


### 22.3.2 SARMUX

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

- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ) or firmware
- Charge pump inside:
  - If  $V_{DDA} < 4.0$  V, charge pump should be turned on to reduce switch resistance
  - If  $V_{DDA} \geq 4.0$  V, charge pump is turned off and delivers  $V_{DDA}$  as its output
- Multiple inputs:
  - Analog signals from pins (port 2)
  - Temperature sensor output
  - CTBm output via sarbus0/1 (not fast enough to sample at 1 Msp/s)
  - AMUXBUS\_A/B (not fast enough to sample at 1 Msp/s)

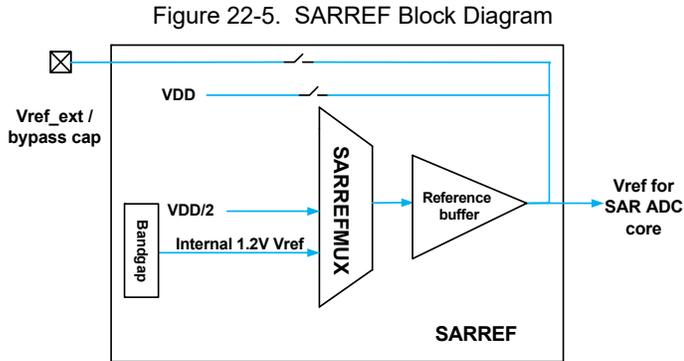
Figure 22-4. SARMUX



### 22.3.3 SARREF

The main features of SARREF are:

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



#### 22.3.3.1 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 1.2-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].

#### 22.3.3.2 Bypass Capacitors

The internal references, 1.2 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 1.2-V internal  $V_{REF}$ , the maximum SAR ADC clock frequency is 1.6 MHz. When using an external reference, it is recommended that an external capacitor is used. Bypass capacitors can be enabled by setting SAR\_CTRL [7]. [Table 22-3](#) lists different reference modes and its maximum frequency/sample rate for 12-bit continuous mode operation.

Table 22-3. Reference Modes

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

1.2-V internal  $V_{REF}$  startup time varies with the different bypass capacitor size, [Table 22-4](#) 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 1.2-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 22-4. Bypass Capacitor Values

| Internal $V_{REF}$ Startup Time                                | Maximum Specification |
|--|-----------------------|
| Startup time for reference with external capacitor (1 $\mu$ F) | 2 ms                  |
| Startup time for reference with external capacitor (100 nF)    | 200 $\mu$ s           |

### 22.3.3.3 Input Range versus Reference

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

### 22.3.4 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 the SAR ADC core (such as resolution, acquisition time, and reference)
- Receives data from the SAR ADC and does pre-processing (average, range detect)
- Uses double buffer to store the result so the CPU can safely read the results of the last scan while the next scan is in progress.

The features of SARSEQ are:

- 16 channels can be individually enabled as an automatic scan without CPU intervention
- An additional 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 TCPWM block, CTBm comparator, low-power comparator, SAR ADC end of conversion signal, and by firmware
- Hardware averaging support
  - First order accumulate
  - From 2 to 256 samples averaging (powers of 2)
  - Results in 16-bit representation
- Double buffering of output data
  - Left or right adjusted results
  - Results in working register and result register
- Interrupt generation
  - Finished scan conversion
  - Channel saturation detect in all control modes
  - Over range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect
- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single ended, or differential, averaging

### 22.3.4.1 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.

Channel configuration SAR\_CHAN\_CONFIG register has an enable bit (AVG\_EN) to enable averaging.

In global configuration, AVG\_CNT (SAR\_SMAPLE\_CTRL [6:4]) specifies the number of samples (N) according to this formula:

$$N = 2^{(AVG\_CNT + 1)} \quad N \text{ range} = [2..256]$$

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; if 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).

### 22.3.4.2 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 329 for details.

### 22.3.4.3 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.

#### 22.3.4.4 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 trigger. 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.

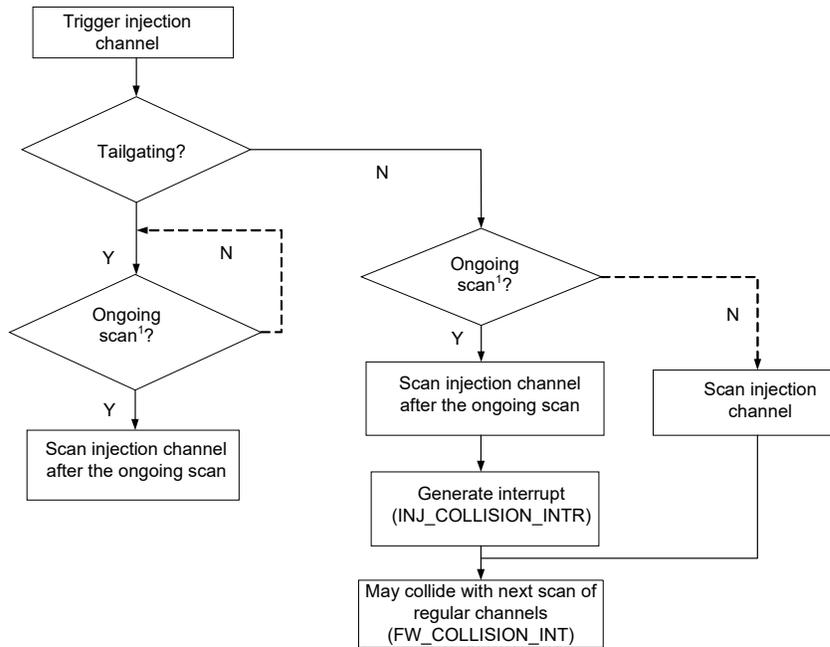
These features are described in detail in [“Global SARSEQ Configuration” on page 334](#), [“Channel Configurations” on page 335](#), and [“SAR Interrupts” on page 328](#).

#### 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\_COLLISION\_INTR is raised). As a result, the next scan of regular channels is postponed until the injection scan is finished, thus causing jitter on a regular scan.

Figure 22-6. Injection Channel Flow Chart



<sup>1</sup> 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\_COLLISION\_INTR).

Figure 22-7 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 22-7. Injection Channel Enabled with Tailgating



### 22.3.5 SAR Interrupts

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.

#### 22.3.5.1 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 GPIO by setting the EOS\_DSI\_OUT\_EN bit in SAR\_SAMPLE\_CTRL [31]. The EOS\_INTR signal is maintained for two system clock cycles. These cycles coincide with the data\_valid signal for the last channel of the scan (if selected).

EOS\_INTR can be masked by making the EOS\_MASK bit 0 in the 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.

#### 22.3.5.2 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.

#### 22.3.5.3 Collision Interrupt

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

There are three collision interrupts: for the firmware trigger (FW\_COLLISION\_INTR), for the external trigger (DSI\_COLLISION\_INTR), and for the injection channel (INJ\_COLLISION\_INTR). These interrupts allow the firmware to identify which trigger collided with an ongoing scan.

When the external trigger is used in level mode, 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.

#### 22.3.5.4 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.

#### 22.3.5.5 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).

#### 22.3.5.6 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\_SATURATE\_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.

#### 22.3.5.7 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).

### 22.3.6 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.
- An external trigger can be TCPWM outputs, CTBm comparator outputs, low-power comparator outputs, and SAR ADC's end-of-sampling and end-of-conversion signals. Hardware trigger is enabled by writing '1' to the DSI\_TRIGGER\_EN bit of the SAR\_SAMPLE\_CTRL register. Signal for the trigger is selected using the PERI\_TR\_GROUP2\_TR\_OUT\_CTL0 register in PSoC 4.

Table 22-5. Hardware Trigger Source Selection in PSoC 4100S Max

| PERI_TR_GROUP2_TR_OUT_CTL0[6:0] | Trigger Source                     |
|---------------------------------|------------------------------------|
| 0                               | Hardwired to 0 (firmware trigger)  |
| 1                               | TCPWM 0 Overflow                   |
| 2                               | TCPWM 1 Overflow                   |
| 3                               | TCPWM 2 Overflow                   |
| 4                               | TCPWM 3 Overflow                   |
| 5                               | TCPWM 4 Overflow                   |
| 6                               | TCPWM 5 Overflow                   |
| 7                               | TCPWM 6 Overflow                   |
| 8                               | TCPWM 7 Overflow                   |
| 9                               | TCPWM 0 Compare Match              |
| 10                              | TCPWM 1 Compare Match              |
| 11                              | TCPWM 2 Compare Match              |
| 12                              | TCPWM 3 Compare Match              |
| 13                              | TCPWM 4 Compare Match              |
| 14                              | TCPWM 5 Compare Match              |
| 15                              | TCPWM 6 Compare Match              |
| 16                              | TCPWM 7 Compare Match              |
| 17                              | TCPWM 0 Underflow                  |
| 18                              | TCPWM 1 Underflow                  |
| 19                              | TCPWM 2 Underflow                  |
| 20                              | TCPWM 3 Underflow                  |
| 21                              | TCPWM 4 Underflow                  |
| 22                              | TCPWM 5 Underflow                  |
| 23                              | TCPWM 6 Underflow                  |
| 24                              | TCPWM 7 Underflow                  |
| 25                              | TCPWM 0 Line Output                |
| 26                              | TCPWM 1 Line Output                |
| 27                              | TCPWM 2 Line Output                |
| 28                              | TCPWM 3 Line Output                |
| 29                              | TCPWM 4 Line Output                |
| 30                              | TCPWM 5 Line Output                |
| 31                              | TCPWM 6 Line Output                |
| 32                              | TCPWM 7 Line Output                |
| 33                              | SAR ADC Sample Done (sdone) Signal |

Table 22-5. Hardware Trigger Source Selection in PSoC 4100S Max (continued)

| PERI_TR_GROUP2_TR_OUT_CTL0[6:0] | Trigger Source                         |
|---------------------------------|--|
| 34                              | SAR ADC End of Conversion (eoc) Signal |
| 35                              | CTBm Comparator 0 Output               |
| 36                              | CTBm Comparator 1 Output               |
| 37                              | LPCOMP 0 Output                        |
| 38                              | LPCOMP 1 Output                        |

- 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 an external trigger coincides with a firmware trigger, the external trigger is handled first and a separate scan is done for the firmware trigger (and a collision interrupt is set). When an external trigger coincides with a continuous trigger, both triggers are effectively handled at the same time (a collision interrupt may be set for the external trigger).

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

### 22.3.6.1 External Trigger Configuration

- Synchronization

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

- Trigger Level

The trigger can either be a pulse or a level; this is indicated by the configuration bit, DSI\_TRIGGER\_LEVEL in the SAR\_SAMPLE\_CTRL register. If it is a level, then the SAR starts new scans for as long as the trigger signal remains high. When the trigger signal is a pulse input, a positive edge detected on the trigger signal triggers a new scan.

- Transmission Time

After the 'dsi\_trigger' is raised, it takes some transmission time before the SAR ADC is told to start sampling. With different DSI\_SYNC\_TRIGGER and DSI\_TRIGGER\_LEVEL configuration, the transmission time is different; Table 22-6 shows the maximum time. Two trigger pulse intervals should be longer than the transmission time, otherwise, the second trigger is ignored.

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

Table 22-6. External Trigger Maximum Time

| Maximum External_TRIGGER Transmission Time      | Bypass Sync<br>DSI_SYNC_TRIGGER=0 | Enable Sync<br>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 22-7. 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 should be longer than the transmission time (as specified in Table 22-6); otherwise, the second trigger pulse will be ignored.          |

### 22.3.7 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 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.

### 22.3.8 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 22-8. ICONT\_LV for Low Power Consumption

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

In addition to controlling the power of the SAR ADC core, the power consumed by 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 22-9 for details.

Table 22-9. SAR VREF Power Options

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

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

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

Use registers to configure the SAR ADC; this is the most common usage. Detailed register bit definition is available in the [PSoC 4100S Max: PSoC 4 Registers TRM](#).

#### 22.3.9.1 SARMUX Analog Routing

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

##### Sequencer Control

It is essential that the appropriate hardware control bits in 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 22-10](#) 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 22-10. PORT\_ADDR and PIN\_ADDR

| PORT_ADDR | PIN_ADDR | Description                    |
|-----------|----------|--------------------------------|
| 0         | 0..7     | 8 dedicated pins of the SARMUX |
| 1         | X        | sarbus0 <sup>a</sup>           |
| 1         | X        | sarbus1 <sup>a</sup>           |
| 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 344](#) for more information. When PORT\_ADDR=1, sarbus0 connects to positive terminal of SAR ADC regardless of the value of PIN\_ADDR; sarbus1 can only connect to the negative terminal of SAR ADC when differential mode is enabled and PORT\_ADDR=1.

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

For single-ended conversions, NEG\_SEL (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 318 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) 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 318 for details.

**22.3.9.2 Global SARSEQ Configuration**

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

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 22-11. Global Configuration Registers

| Configurations                                | Control Registers                                    | Detailed Reference                                |
|---|--|---|
| Reference selection                           | SAR_CTRL[6:4]  | <a href="#">22.3.3.1 Reference Options</a>        |
| Signed/unsigned selection                     | SAR_SAMPLE_CTRL [3:2]                                | <a href="#">22.3.1.3 Result Data Format</a>       |
| Data left/right alignment                     | SAR_SAMPLE_CTRL [1]                                  | <a href="#">22.3.1.3 Result Data Format</a>       |
| Negative input selection in single-ended mode | SAR_CTRL[11:9]                                       | <a href="#">22.3.1.4 Negative Input Selection</a> |
| Resolution                                    | SAR_SAMPLE_CTRL[0] <sup>a</sup>                      | <a href="#">22.3.1.5 Resolution</a>               |
| Acquisition time                              | SAR_SAMPLE_TIME01 [25:0]<br>SAR_SAMPLE_TIME32 [25:0] | <a href="#">22.3.1.6 Acquisition Time</a>         |
| Averaging count                               | SAR_SAMPLE_CTRL[7:4]                                 | <a href="#">22.3.4.1 Averaging</a>                |
| Range detection                               | SAR_RANGE_THRES [31:0]<br>SAR_RANGE_COND [31:30]     | <a href="#">22.3.4.2 Range Detection</a>          |

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.

### 22.3.9.3 Channel Configurations

Channel configuration includes:

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

As a general rule, 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 22-12. Channel Configuration Registers

| Configurations             | Registers                | Detailed Reference  |
|----------------------------|--------------------------|---|
| Single-ended/differential  | SAR_CHANx_CONFIG [8]     | <a href="#">22.3.1.1 Single-ended and Differential Mode</a> |
| Acquisition time selection | SAR_CHANx_CONFIG [13:12] | <a href="#">22.3.1.6 Acquisition Time</a>                   |
| Resolution selection       | SAR_CHANx_CONFIG [9]     | <a href="#">22.3.1.5 Resolution</a>                         |
| Average enable             | SAR_CHANx_CONFIG [10]    | <a href="#">22.3.4.1 Averaging</a>                          |

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 22-13. Resolution

| Average | SUB_RESOLUTION<br>SAR_SAMPLE_CTRL[0] | Register Mode Resolution<br>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             |

### 22.3.9.4 Channel Enables

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

### 22.3.9.5 *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 [22.3.5 SAR Interrupts](#) for details.

### 22.3.9.6 *Trigger*

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

- Firmware trigger: SAR\_START\_CTRL [0]
- External trigger: dsi\_trigger
- Continuous trigger: SAR\_SAMPLE\_CTRL [16]

See [22.3.6 Trigger](#) for details.

### 22.3.9.7 *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.

### 22.3.9.8 *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 [22.3.4.4 Injection Channel](#) for details.

### 22.3.10 Temperature Sensor Configuration

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

A pin or signal can be routed to the SAR ADC in three ways. [Table 22-14](#) 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 22-14. Route Temperature to SAR ADC

| Control Methods | Setup   |
|-----------------|---|
| Sequencer       | DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8])<br>VREF_SEL = 0 (SAR_CTRL[6:4])<br>PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4])<br>PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0])<br>SWITCH_DISABLE = 0 (SAR_CTRL[30])<br>SAR_MUX_SWITCH0[16] = 1<br>SAR_MUX_SWITCH0[17] = 1<br>SAR_MUX_SWITCH_HW_CTRL[16]= 1<br>SAR_MUX_SWITCH_HW_CTRL[17]= 1<br>NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0 <sup>a</sup> |
| Firmware        | DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8])<br>VREF_SEL = 0 (SAR_CTRL[6:4])<br>SWITCH_DISABLE = 1 (SAR_CTRL[30])<br>SAR_MUX_SWITCH0[16] = 1<br>SAR_MUX_SWITCH0[17] = 1<br>SAR_MUX_SWITCH_HW_CTRL[16]= 0<br>SAR_MUX_SWITCH_HW_CTRL[17]= 0<br>NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0 <sup>a</sup>  |

a. For temperature sensor, override NEL\_SEG (SAR\_CTRL [11:9]) to '0'.

## 22.4 Registers

| Name                     | Offset | Qty. | Width | Description   |
|--------------------------|--------|------|-------|---|
| SAR_CTRL                 | 0x0000 | 1    | 32    | Global configuration register<br>Analog control register  |
| SAR_SAMPLE_CTRL          | 0x0004 | 1    | 32    | Global configuration register<br>Sample control register  |
| SAR_SAMPLE_TIME01        | 0x0010 | 1    | 32    | Global configuration register<br>Sample time specification ST0 and ST1  |
| SAR_SAMPLE_TIME23        | 0x0014 | 1    | 32    | Global configuration register<br>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   |

# 23. Low-Power Comparator



PSoC 4 devices have two low-power comparators. These comparators can perform fast analog signal comparison in all system power modes. Refer to the [Power Modes chapter on page 104](#) for details on various device power modes. The positive and negative inputs can be connected to dedicated GPIO pins or to AMUXBUS-A/AMUXBUS-B. The comparator output can be read by the CPU through a status register, used as an interrupt or wakeup source or routed to a GPIO.

## 23.1 Features

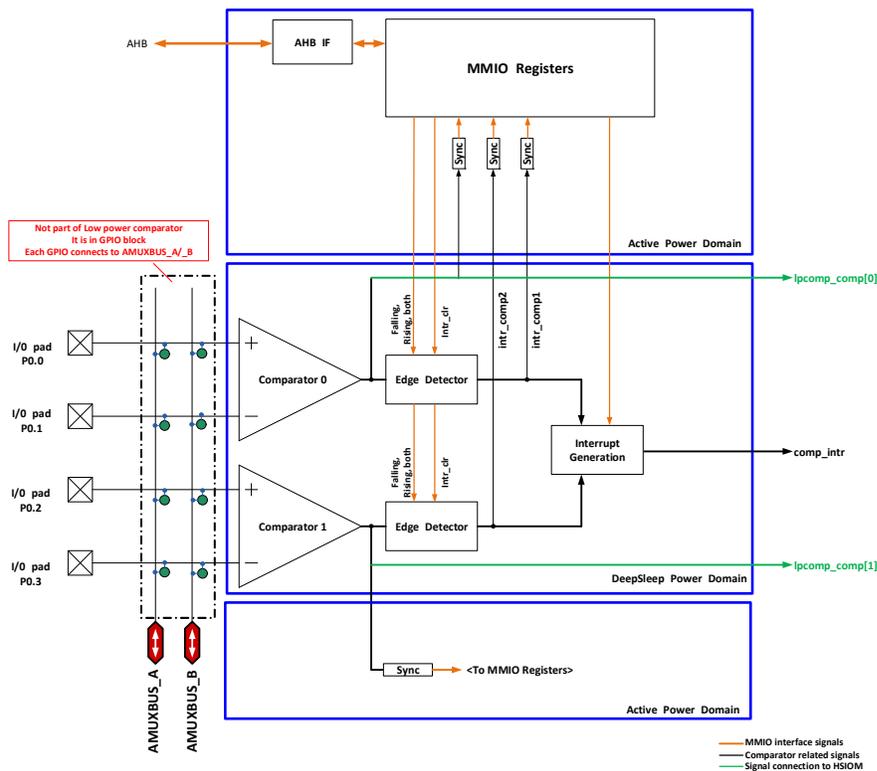
PSoC 4 comparators have the following features:

- Configurable positive and negative inputs
- Programmable power and speed
- Ultra low-power mode support (<4  $\mu$ A)
- Optional 10-mV input hysteresis
- Low-input offset voltage (<4 mV after trim)
- Wakeup source in Deep-Sleep mode

## 23.2 Block Diagram

Figure 23-1 shows the block diagram for the low-power comparator.

Figure 23-1. Low-Power Comparator Block Diagram



## 23.3 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.

### 23.3.1 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 23-1](#), note that P0.0 and P0.1 connect to positive and negative inputs of Comparator 0; P0.2 and P0.3 connect to the inputs of Comparator 1. Also, note that the AMUXBUS nets do not have a direct connection to the comparator inputs. Therefore, the comparator connection is routed to the AMUXBUS nets through the corresponding input pin. These input pins will not be available for other purposes when using AMUXBUS for comparator connections. They should be left open in designs that use AMUXBUS for comparator input connection. Note that AMUXBUS connections are not available in Deep-Sleep mode. If Deep-Sleep operation is required, the 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 66](#) for more details on connecting the GPIO to AMUXBUS A/B or setting up the GPIO for comparator input.

### 23.3.2 Output and Interrupt Configuration

The output of Comparator0 and Comparator1 are available in the OUT1 bit [6] and OUT2 bit [14], respectively, in the LPCOMP\_CONFIG register (see [Table 23-1](#)). The comparator outputs are synchronized to SYSCLK before latching them to the OUTx bits in the LPCOMP\_CONFIG register. The output of each comparator is connected to a corresponding edge detector block. This block determines the edge that triggers the interrupt. The edge selection and interrupt enable is configured using the INTTYPE1 bits [5:4] and INTTYPE2 bits [13:12] in the LPCOMP\_CONFIG register. Using the INTTYPEx bits, the interrupt type can be selected to disabled, rising edge, falling edge, or both edges, as described in [Table 23-1](#).

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 72](#) 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.

During an edge event, the comparator will trigger an interrupt (intr\_comp1/intr\_comp2 signals in [Figure 23-1](#)). 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 23-1](#)), 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 52](#) 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..

The LPCOMP interrupt (comp1\_intr/comp2\_intr) is synchronous with SYSCLK. Clearing comp1\_intr/comp2\_intr are all synchronous.

LPCOMP\_INTR\_SET register bits [1:0] can be used to assert an interrupt for software debugging.

In Deep-Sleep mode, the 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 23-1. 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<br>00: Disabled<br>01: Rising Edge<br>10: Falling Edge<br>11: Both rising and falling edges |
| LPCOMP_CONFIG[13:12] | INTTYPE2 | Sets on which edge Comparator1 will trigger an IRQ<br>00: Disabled<br>01: Rising Edge<br>10: Falling Edge<br>11: Both rising and falling edges |
| LPCOMP_INTR[0]       | COMP1    | Comparator0 Interrupt: hardware sets this interrupt when Comparator0 triggers. Write a '1' to clear the interrupt                              |
| LPCOMP_INTR[1]       | COMP2    | Comparator2 Interrupt: hardware sets this interrupt when Comparator1 triggers. Write a '1' to clear the interrupt                              |
| LPCOMP_INTR_SET[0]   | COMP1    | Write a '1' to trigger the software interrupt for Comparator0  |
| LPCOMP_INTR_SET[1]   | COMP2    | Write a 1 to trigger the software interrupt for Comparator1  |

### 23.3.3 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 23-2](#).

**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 23-2. Comparator Power Mode Selection Bits MODE1 and MODE2

| Register[Bit_Pos]  | Bit_Name | Description   |
|--------------------|----------|---|
| LPCOMP_CONFIG[1:0] | MODE1    | Comparator0 power mode selection<br>00: Slow operating mode (uses less power)<br>01: Fast operating mode (uses more power)<br>10: Ultra low-power operating mode (uses lowest possible power) |
| LPCOMP_CONFIG[9:8] | MODE2    | Comparator1 power mode selection<br>00: Slow operating mode (uses less power)<br>01: Fast operating mode (uses more power)<br>10: Ultra low-power operating mode (uses lowest possible power) |
| LPCOMP_CONFIG[7]   | ENABLE1  | Comparator0 enable bit<br>0: Disables Comparator0<br>1: Enables Comparator0   |
| LPCOMP_CONFIG[15]  | ENABLE2  | Comparator1 enable bit<br>0: Disables Comparator1<br>1: Enables Comparator1   |

### 23.3.4 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 23-3](#).

Table 23-3. Hysteresis Control Bits HYST1 and HYST2

| Register[Bit_Pos] | Bit_Name | Description   |
|-------------------|----------|---|
| LPCOMP_CONFIG[2]  | HYST1    | Enable/Disable 10 mV hysteresis to Comparator0<br>- 0: Enable Hysteresis<br>- 1: Disable Hysteresis |
| LPCOMP_CONFIG[10] | HYST2    | Enable/Disable 10 mV hysteresis to Comparator1<br>- 0: Enable Hysteresis<br>- 1: Disable Hysteresis |

### 23.3.5 Wakeup from Low-Power Modes

The comparator is operational in the device's low-power modes, including Sleep and Deep-Sleep modes. The comparator output interrupt can wake the device from Sleep and Deep-Sleep modes. The comparator should be enabled in the LPCOMP\_CONFIG register, the INTTYPE<sub>x</sub> bits in the LPCOMP\_CONFIG register should not be set to disabled, and the INTR\_MASK<sub>x</sub> bit should be set in the LPCOMP\_INTR\_MASK register for the corresponding comparator to wake the device from low-power modes. Comparisons involving AMUXBUS connections are not available in DeepSleep mode.

In the Deep-Sleep power mode, a compare event on either Comparator0 or Comparator1 output will generate a wakeup interrupt. The INTTYPE<sub>x</sub> 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.

### 23.3.6 Comparator Clock

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

### 23.3.7 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 [4:0] 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[4:0] until the output switches direction again. The TRIMB bit tuning is valid only for slow mode of comparator operation. TRIMB bit[4] 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  $V_{ref}$ .

## 23.4 Register Summary

Table 23-4. 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   |

## 24. 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. PSoC 4 device has one CTBm block, which can be operational in Deep-Sleep power mode.

### 24.1 Features

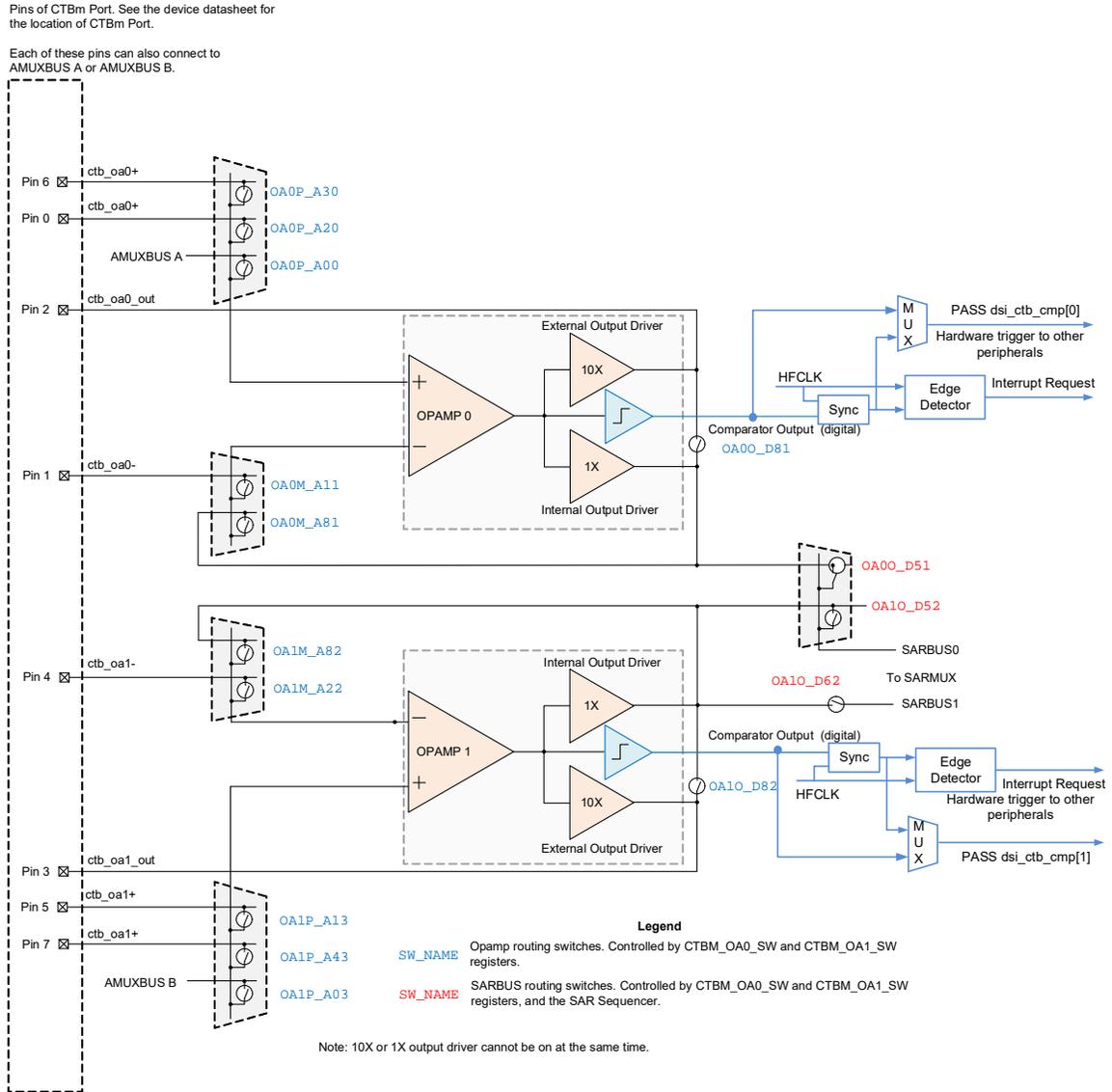
The opamps in the PSoC 4 CTBm block have the following features:

- 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

## 24.2 Block Diagram

Figure 24-1 shows the block diagram for the CTBm block available in PSoC 4 devices.

Figure 24-1. CTBm Block Diagram



## 24.3 How It Works

As the block diagram shows, 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 24-1](#); 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 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 CTBM\_OA\_RES0\_CTRL [11] bit for opamp0 and CTBM\_OA\_RES1\_CTRL [11] bit for opamp1.

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

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

Follow these steps to set up a comparator:

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

### 24.3.1 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 CTBM\_OA\_RESx\_CTRL. The slew rate and gain bandwidth are maximum in high-power mode and minimum in low-power mode. Note that power mode configuration also affects the maximum output drive capability ( $I_{OUT}$ ) in 1X mode. See [Table 24-1](#) for details. See the [device datasheet](#) for gain bandwidth, slew rate, and  $I_{OUT}$  specifications in various power modes.

### 24.3.2 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 24-1](#).

Table 24-1. Output Driver versus Power Mode

| Power Mode $I_{OUT}$ Drive Capability | CTBM_OA_RESx_CTRL[1:0] |             |             |           |
|---------------------------------------|------------------------|-------------|-------------|-----------|
|                                       | 00 (disable)           | 01 (low)    | 10 (medium) | 11 (high) |
| External Driver (10X)                 | Off                    | 10 mA       | 10 mA       | 10 mA     |
| Internal Driver (1X)                  | Off                    | 100 $\mu$ A | 400 $\mu$ A | 1 mA      |

The CTBM\_OA\_RESx\_CTRL[2] bit is used to select between the 10X and 1X output capability (0: 1X, 1: 10X). If the output of the opamp is connected to the SAR ADC, it is recommended to choose the 1X output driver. If the output of the opamp is connected to an external pin, 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 CTBM\_OAx\_SW [21] to '1'. However, Cypress does not guarantee performance in this case.

Table 24-2 summarizes the bits used to configure the opamp output drive strength and power modes.

Table 24-2. Output Strength and Power Mode Configuration in CTBM Registers

| Register[Bit_Pos]       | Bit_Name          | Description   |
|-------------------------|-------------------|---|
| CTBM_CTB_CTRL[31]       | ENABLE            | CTBM power mode selection<br>0: CTBM is disabled<br>1: CTBM is enabled  |
| CTBM_OA_RES0_CTRL [11]  | OA0_PUMP_EN       | Opamp0 pump enable bit<br>0: Opamp0 pump is disabled<br>1: Opamp0 pump is enabled   |
| CTBM_OA_RES1_CTRL [11]  | OA1_PUMP_EN       | Opamp1 pump enable bit<br>0: Opamp1 pump is disabled<br>1: Opamp1 pump is enabled   |
| CTBM_OA_RES0_CTRL [1:0] | OA0_PWR_MODE      | Opamp0 power mode select bits<br>00: Opamp0 is OFF<br>01: Opamp0 is in low power mode<br>10: Opamp0 is in medium power mode<br>11: Opamp0 is in high power mode |
| CTBM_OA_RES1_CTRL [1:0] | OA1_PWR_MODE      | Opamp1 power mode select bits<br>00: Opamp1 is OFF<br>01: Opamp1 is in low power mode<br>10: Opamp1 is in medium power mode<br>11: Opamp1 is in high power mode |
| CTBM_OA_RES0_CTRL [2]   | OA0_DRIVE_STR_SEL | Opamp0 output drive strength select bits<br>0: Opamp0 output drive strength is 1X<br>1: Opamp0 output drive strength is 10X                                     |
| CTBM_OA_RES1_CTRL [2]   | OA1_DRIVE_STR_SEL | Opamp1 output drive strength select bits<br>0: Opamp1 output drive strength is 1X<br>1: Opamp1 output drive strength is 10X                                     |

### 24.3.3 Compensation

Each opamp also has a programmable compensation capacitor block, which allows optimizing the stability of the opamp performance based on output load. The compensation of each opamp is controlled by the respective CTBM\_OAx\_COMP\_TRIM register, as explained in Table 24-3. Note that all the GBW slew rate specifications in the [device datasheet](#) are applied for all compensation trims.

Table 24-3. Opampx (Opamp0 or Opamp1) Compensation Bits in CTBm

| Register[Bit_Pos]       | Bit_Name      | Description  |
|-------------------------|---------------|--|
| CTBM_OAx_COMP_TRIM[1:0] | OAx_COMP_TRIM | OpampX compensation trim bits<br>00: No compensation<br>01: Minimum compensation, high speed, and low stability<br>10: Medium compensation, balanced speed, and stability<br>11: Maximum compensation, low speed, and high stability |

## 24.3.4 Switch Control

The CTBm has many switches to configure the opamp input and output. Most of them are controlled by configuring CTBm registers (CTBM\_OA0\_SW, CTBM\_OA1\_SW), except three switches, which are used to connect the output of opamps to SAR ADC through sarbus0 and sarbus1. They must be controlled by SAR ADC registers, and CTBm registers.

Switches can be closed by setting the corresponding bit in register CTBM\_OAx\_SW; clearing them will cause the corresponding switches to open. To open the switch, writeS '1' to CTBM\_OAx\_SW\_CLEAR, which clears the corresponding bit in CTBM\_OAx\_SW. See the *PSoC 4100S Max: PSoc 4 Registers TRM* for details on the switches and the connections they enable.

### 24.3.4.1 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.

**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 have three positive input options through analog switches: two external pins and one AMUXBUS line. See [Table 24-4](#) for details.

Table 24-4. Positive Input Selection

|        | Positive Input | Switch Control Bit | Description             |
|--------|----------------|--------------------|-------------------------|
| Opamp0 | AMUXBUS A      | CTBM_OA0_SW [0]    | 0: open 1: close switch |
|        | P1.0           | CTBM_OA0_SW [2]    | 0: open 1: close switch |
|        | P1.6           | CTBM_OA0_SW [3]    | 0: open 1: close switch |
| Opamp1 | AMUXBUS B      | CTBM_OA1_SW [0]    | 0: open 1: close switch |
|        | P1.5           | CTBM_OA1_SW [1]    | 0: open 1: close switch |
|        | P1.7           | CTBM_OA1_SW [4]    | 0: open 1: close switch |

- Negative input: Both opamp0 and opamp1 have two negative input options through analog switches: one external pin or output feedback, which is controlled by the CTBM\_OAx\_SW register. [Table 24-5](#) shows the control bits.

Table 24-5. Negative Input Selection

|        | Negative Input                                  | Switch Control Bit | Description             |
|--------|---|--------------------|-------------------------|
| Opamp0 | P1.1  | CTBM_OA0_SW [8]    | 0: open 1: close switch |
|        | Opamp0 output feedback through 1X output driver | CTBM_OA0_SW [14]   | 0: open 1: close switch |
| Opamp1 | P1.4  | CTBM_OA1_SW [8]    | 0: open 1: close switch |
|        | Opamp1 output feedback through 1X output driver | CTBM_OA1_SW [14]   | 0: open 1: close switch |

### 24.3.4.2 Output Configuration

Each opamp's output is connected directly to a fixed pin; no additional setup is needed. Optionally, it can be connected to sarbus0 or sarbus1 through three switches (SW1/2/3). The opamp0 output can be connected to sarbus0 and opamp1 can be connected to sarbus0 or sarbus1. sarbus0 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, and CTBm register together; the other switches can be controlled only by CTBm register.

The following truth tables ([Table 24-6](#), [Table 24-7](#), and [Table 24-8](#)) 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. CTBM\_SW\_HW\_CTRL bit [2] or [3] should be set when using the SAR register to control switches. CTBM\_OAx\_SW[18]/[19] can mask the other control bits – if CTBM\_OAx\_SW[18]/[19] = 0, SW[n] = 0.

The CTBM\_\_SW\_STATUS [30:28] register gives the current switch status of SW1/2/3.

Table 24-6. Truth Table of SW1 Control Logic

| PORT_ADDR | PIN_ADDR | CTBM_SW_HW_CTRL[2] | CTBM_OA0_SW[18] | SW1 |
|-----------|----------|--------------------|-----------------|-----|
| X         | X        | X                  | 0               | 0   |
| X         | 0        | 1                  | 1               | 0   |
| 0         | X        | 1                  | 1               | 0   |
| X         | X        | X                  | 1               | 1   |
| X         | X        | 0                  | 1               | 1   |
| 1         | 2        | X                  | 1               | 1   |

Table 24-7. Truth Table of SW2 Control Logic

| DIFFERENTIAL_EN | PORT_ADDR | PIN_ADDR | CTBM_SW_HW_CTRL[3] | CTBM_OA0_SW[18] | SW2 |
|-----------------|-----------|----------|--------------------|-----------------|-----|
| X               | X         | X        | X                  | 0               | 0   |
| X               | X         | 0        | 1                  | 1               | 0   |
| X               | 0         | X        | 1                  | 1               | 0   |
| 1               | X         | X        | X                  | 1               | 0   |
| X               | X         | X        | 0                  | 1               | 1   |
| X               | X         | X        | X                  | 1               | 1   |
| 0               | 1         | 3        | X                  | 1               | 1   |

#### 24.3.4.3 Comparator Mode

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

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

#### 24.3.4.4 Comparator Configuration

The hysteresis of 10 mV  $\pm$ 5 percent can be enabled in one direction (low to high). Input hysteresis can be enabled by setting CTBM\_OA\_RESx\_CTRL[5]. The two comparators also have three power modes: low, medium, and high, controlled by setting CTBM\_OA\_RESx\_CTRL [1:0]. Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ultra-low-power mode. Exact specifications for power consumption and response time are provided in the datasheet.

The synchronization of the comparator output with the system AHB clock can be configured in CTBM\_OA\_RESx\_CTRL[6].

The output state of comparator0 and comparator1 are stored in CTBM\_COMP\_STAT[0] and CTBM\_COMP\_STAT[16], respectively.

Table 24-8 summarizes various bits used to configure the comparator mode in the CTBM block.

Table 24-8. Comparator Mode and Configuration Register Settings

| Register[Bit_Pos]    | Bit_Name            | Description  |
|----------------------|---------------------|--|
| CTBM_OA_RESy_CTRL[4] | OAX_COMP_EN         | OpampX comparator enable bit<br>0: Comparator mode is disabled in opampX<br>1: Comparator mode is enabled in opampX      |
| CTBM_OA_RESx_CTRL[5] | OAX_HYST_EN         | OpampX Comparator hysteresis enable bit<br>0: Hysteresis is disabled in opampX<br>1: Hysteresis is enabled in opampX     |
| CTBM_OA_RESx_CTRL[6] | OAX_BYPASS_DSI_SYNC | OpampX bypass comparator output synchronization for DSI (trigger) output<br>0: Synchronize (level or pulse)<br>1: Bypass |
| CTBM_OA_RESx_CTRL[7] | OAX_DSI_LEVEL       | OpampX comparator DSI (trigger) output synchronization level<br>0: Pulse<br>1: Level                                     |

#### 24.3.4.5 Comparator Interrupt

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

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

Each interrupt has an interrupt mask bit in the CTBM\_INTR\_MASK register. By setting the interrupt mask low, the corresponding interrupt source is ignored. The CTBm comparator interrupt to the NVIC will be raised if logic AND of the interrupt flags in CTBM\_INTR registers and the corresponding interrupt masks in CTBM\_INTR\_MASK register is 1.

Writing a '1' to the CTBM\_INTR bit [1:0] can clear corresponding interrupt.

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

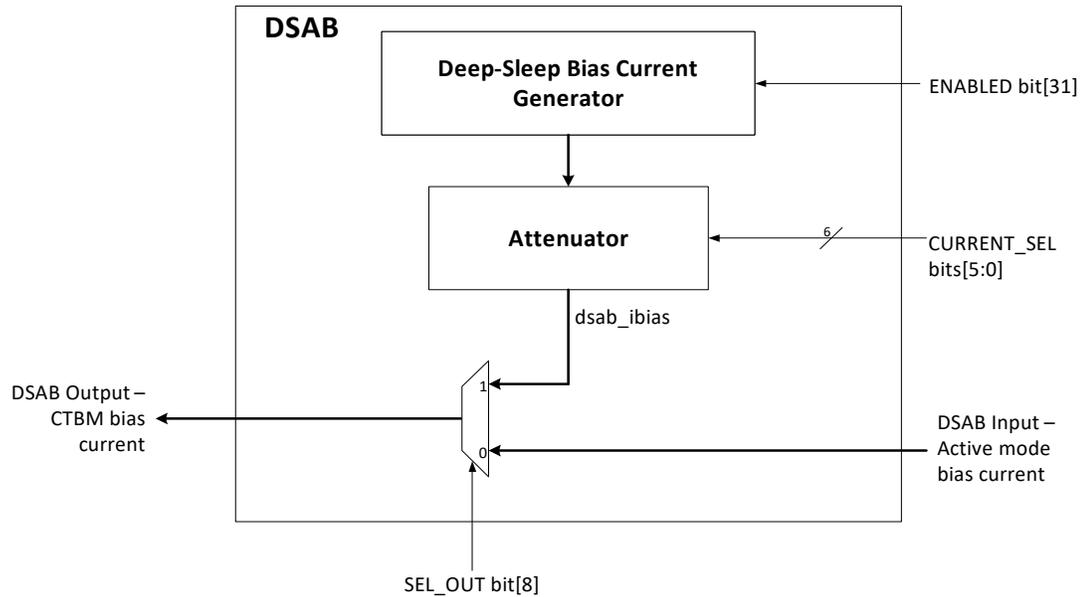
For verification and debug purposes, a set bit is provided for each interrupt in the CTBM\_INTR\_SET register. This bit allows the firmware to raise the interrupt without a real comparator switch event.

#### 24.3.4.6 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 104](#) 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 24-2 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  $CURRENT\_SEL \times 0.075 \mu A$  ( $\pm 5$  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 24-9 summarizes the bit configuration settings of the PASS\_DSAB\_DSAB\_CTRL register.

Figure 24-2. 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  $\sim 2.8$  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 CTBM\_CTB\_CTRL register. This bit enables both the opamps of the CTBM during deep sleep. The deep-sleep operation of the CTBm also requires the DSAB block to be enabled.

Table 24-9. DSAB and CTBM Deep-Sleep Configuration Register Settings

| Register[Bit_Pos]         | Bit_Name     | Description  |
|---------------------------|--------------|--|
| PASS_DSAB_DSAB_CTRL [5:0] | CURRENT_SEL  | Current selection for the dsab_ibias; $dsab\_ibias = CURRENT\_SEL \times 0.075 \mu A (\pm 5\%)$  |
| PASS_DSAB_DSAB_CTRL [8]   | SEL_OUT      | CTBm bias current selection<br>0: Bypass DSAB and use active mode bias current<br>1: Use dsab_ibias as the CTBm bias current   |
| PASS_DSAB_DSAB_CTRL [31]  | ENABLED      | Enable/disable DSAB bias generator<br>0: DSAB block is disabled and the CTBm bias current is connected to the Active mode bias current<br>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<br>0: Enabled<br>1: Disabled   |

## 24.4 Register Summary

Table 24-10. Register Summary

| Name                        | Description                     |
|-----------------------------|---------------------------------|
| CTBM0_CTRL                  | Global CTBM0 block enable       |
| CTBM0_OA_RES0_CTRL          | Opamp0 control register         |
| CTBM0_OA_RES1_CTRL          | Opamp1 control register         |
| CTBM0_COMP_STAT             | Comparator status               |
| CTBM0_INTR                  | Interrupt request register      |
| CTBM0_INTR_SET              | Interrupt request set register  |
| CTBM0_INTR_MASK             | Interrupt request mask          |
| CTBM0_INTR_MASKED           | Interrupt request masked        |
| CTBM0_OA0_SW                | Opamp0 switch control           |
| CTBM0_OA0_SW_CLEAR          | Opamp0 switch control clear     |
| CTBM0_OA1_SW                | Opamp1 switch control           |
| CTBM0_OA1_SW_CLEAR          | Opamp1 switch control clear     |
| CTBM0_SW_HW_CTRL            | CTBM0 hardware control enable   |
| CTBM0_SW_STATUS             | CTBM0 bus switch control status |
| CTBM0_OA0_OFFSET_TRIM       | Opamp0 trim control             |
| CTBM0_OA0_SLOPE_OFFSET_TRIM | Opamp0 trim control             |
| CTBM0_OA0_COMP_TRIM         | Opamp0 trim control             |
| CTBM0_OA1_OFFSET_TRIM       | Opamp1 trim control             |
| CTBM0_OA1_SLOPE_OFFSET_TRIM | Opamp1 trim control             |
| CTBM0_OA1_COMP_TRIM         | Opamp1 trim control             |
| PASS_DSAB_DSAB_CTRL         | DSAB control register           |
| PASS_DSAB_TRIM              | IBIAS trim register             |

## 25. CapSense



The CapSense system can measure the self-capacitance of an electrode or the mutual capacitance between a pair of electrodes. A new generation of CapSense IP has been introduced in this device, known as Multi Sense Converter (MSC), which has additional features compared to other PSoC 4 devices.

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 Eclipse IDE for ModusToolbox to implement CapSense designs. See the [PSoC 4 and PSoC 6 MCU CAP-SENSE 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

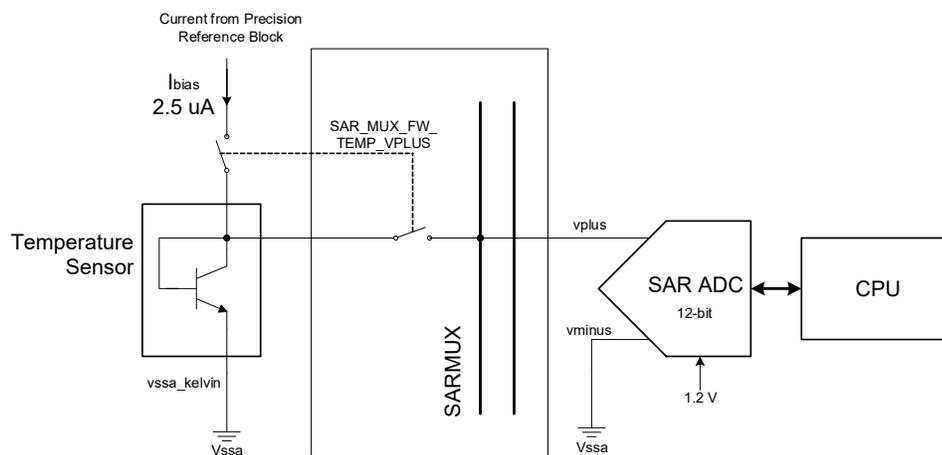
The temperature sensor has the following features:

- $\pm 5^\circ$  Celsius accuracy over temperature range  $-40^\circ\text{C}$  to  $+85^\circ\text{C}$
- $0.5^\circ$  Celsius/LSB resolution (without amplification) when using a 12-bit SAR ADC with a 1.2-V reference
- 10  $\mu\text{s}$  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  $^\circ\text{C}$  can be calculated from the ADC results as given in [Equation 26-1](#).

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

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 [Equation 26-2](#).

$$A = (\text{signed int})\left(2^{16}\left(\frac{100^{\circ}\text{C} - (-40^{\circ}\text{C})}{\text{SAR}_{100^{\circ}\text{C}} - \text{SAR}_{-40^{\circ}\text{C}}}\right)\right) \quad \text{Equation 26-2}$$

Where,

$\text{SAR}_{100\text{C}}$  = ADC counts at 100°C

$\text{SAR}_{-40\text{C}}$  = 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 ( $I_{\text{bias}}$ ) present in the chip. It is calculated as given in [Equation 26-3](#).

$$B = (\text{unsigned int})\left(2^6 \times 100^{\circ}\text{C} - \left(\frac{A \times \text{SAR}_{100\text{C}}}{2^{10}}\right)\right) \quad \text{Equation 26-3}$$

Where,

$\text{SAR}_{100\text{C}}$  = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH\_SAR\_TEMP\_OFFSET.

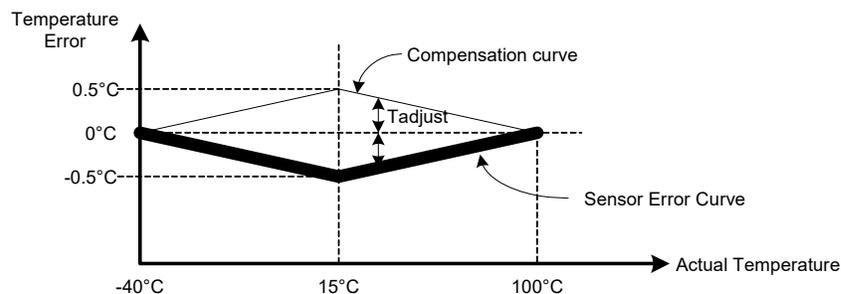
- $T_{\text{adjust}}$  is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating  $T_{\text{initial}} = (A \times \text{SAR}_{\text{out}} + 2^{10} \times B)$ . If it is greater than the center value (15°C), then  $T_{\text{adjust}}$  is given in [Equation 26-4](#).

$$T_{\text{adjust}} = \left(\frac{0.5^{\circ}\text{C}}{100^{\circ}\text{C} - 15^{\circ}\text{C}} \times (100^{\circ}\text{C} \times 2^{16} - T_{\text{initial}})\right) \quad \text{Equation 26-4}$$

If less than center value, then  $T_{\text{adjust}}$  is given in [Equation 26-5](#).

$$T_{\text{adjust}} = \left(\frac{0.5^{\circ}\text{C}}{40^{\circ}\text{C} + 15^{\circ}\text{C}} \times (40^{\circ}\text{C} \times 2^{16} - T_{\text{initial}})\right) \quad \text{Equation 26-5}$$

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 1.2-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, or firmware. See the [SAR ADC chapter on page 315](#) 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  $V_{NEG} = V_{SS}$ ,  $V_{REF} = 1.2\text{ V}$ , 12-bit resolution, and right-aligned result.
3. Enable the temperature sensor.
4. Get the digital output from the SAR ADC.
5. Fetch 'A' from SFLASH\_SAR\_TEMP\_MULTIPLIER and 'B' from SFLASH\_SAR\_TEMP\_OFFSET.
6. Calculate the die temperature using the linear equation (see [Equation 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  $V_{BE}$ :  $0xBC4B \times 0x595 = (-17333)_{10} \times (1429)_{10} = (-24768857)_{10}$
  - b. Multiply B and 1024:  $0x65B4 \times 0x400 = (26036)_{10} \times (1024)_{10} = (26660864)_{10}$
  - 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  $15^{\circ}\text{C}$  ( $0x1C$  - 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,  $\text{Temp} = 29.19219^{\circ}\text{C} \sim 29.2^{\circ}\text{C}$

## 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 <a href="#">Equation 26-1</a> .  |
| SFLASH_SAR_TEMP_OFFSET     | Constant 'B' as defined in <a href="#">Equation 26-1</a> .   |

# 27. Analog Routing



The PSoC 4 MCU has a flexible analog routing system that interconnects programmable analog blocks and GPIOs. Analog routing in the PSoC 4 MCU reduces external connections, allows last-minute feature changes, and eliminates delays caused by PCB spins by reconfiguring interconnection between analog blocks.

## 27.1 Features

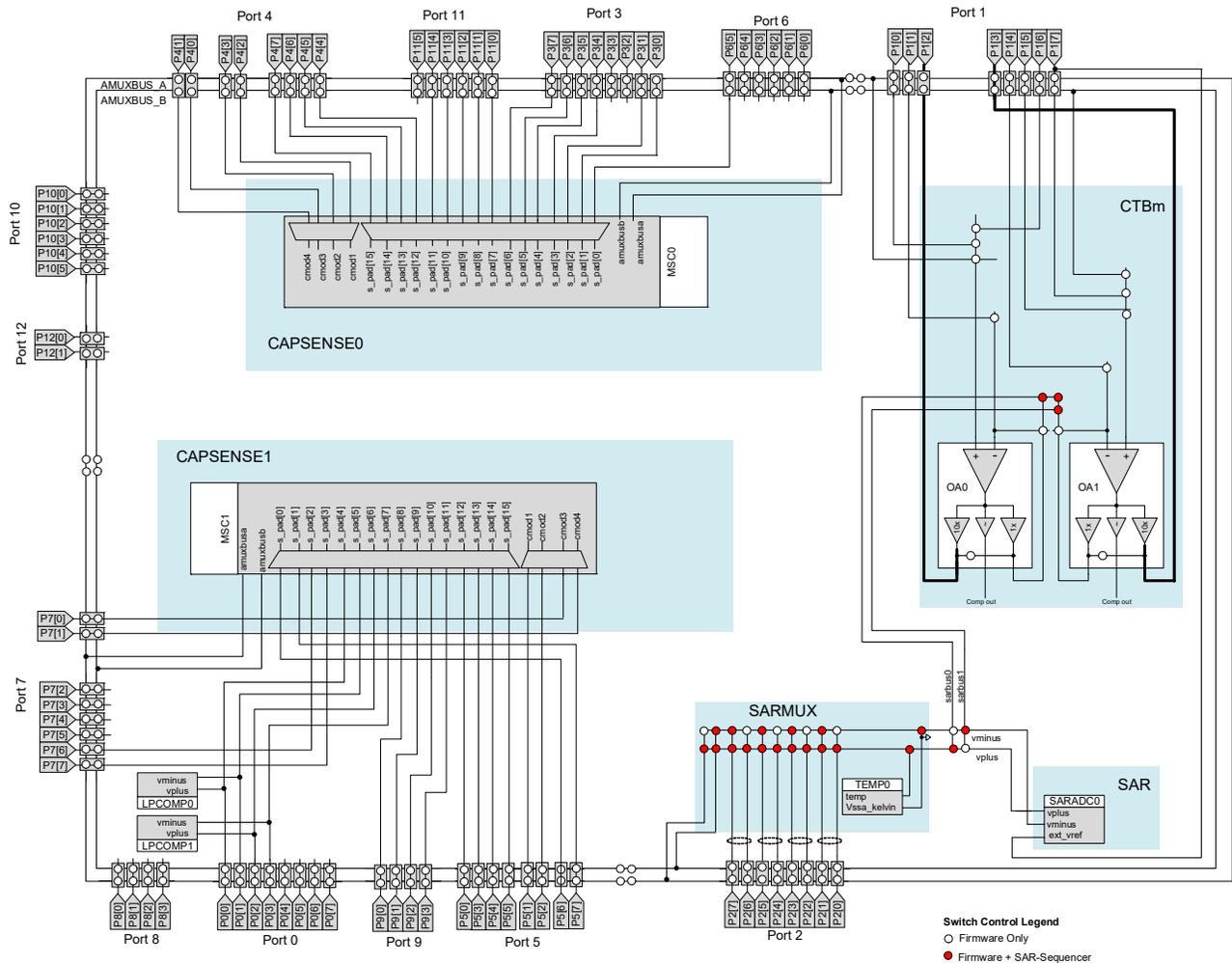
The PSoC 4 MCU analog routing has the following features:

- Flexible connections between different analog blocks and GPIOs
- Two system-wide analog mux buses (AMUXBUS) that interconnect all analog capable GPIOs
- Switches in the analog routing can be used to create analog multiplexers

## 27.2 Architecture

SARMUX has many switches that may be controlled by SARSEQ block (sequencer controller) or firmware. The sequencer is the hardware control method, which can be masked by the hardware control bit in the register, SARx\_MUX\_SWITCH\_CTRL. Different control methods have different control capability on the switches. See [Figure 27-1](#).

Figure 27-1. 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 27-1. The corresponding registers are: SAR\_CHANx\_CONFIG, SAR\_MUX\_SWITCH0, SAR\_CTRL, and SAR\_MUX\_SWITCH\_HW\_CTRL.

**Firmware control:** Programmable registers directly define the VPLUS/VMINUS connection. It can control every switch in SARMUX; see Figure 27-1. 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.

### 27.2.1 Analog Interconnection

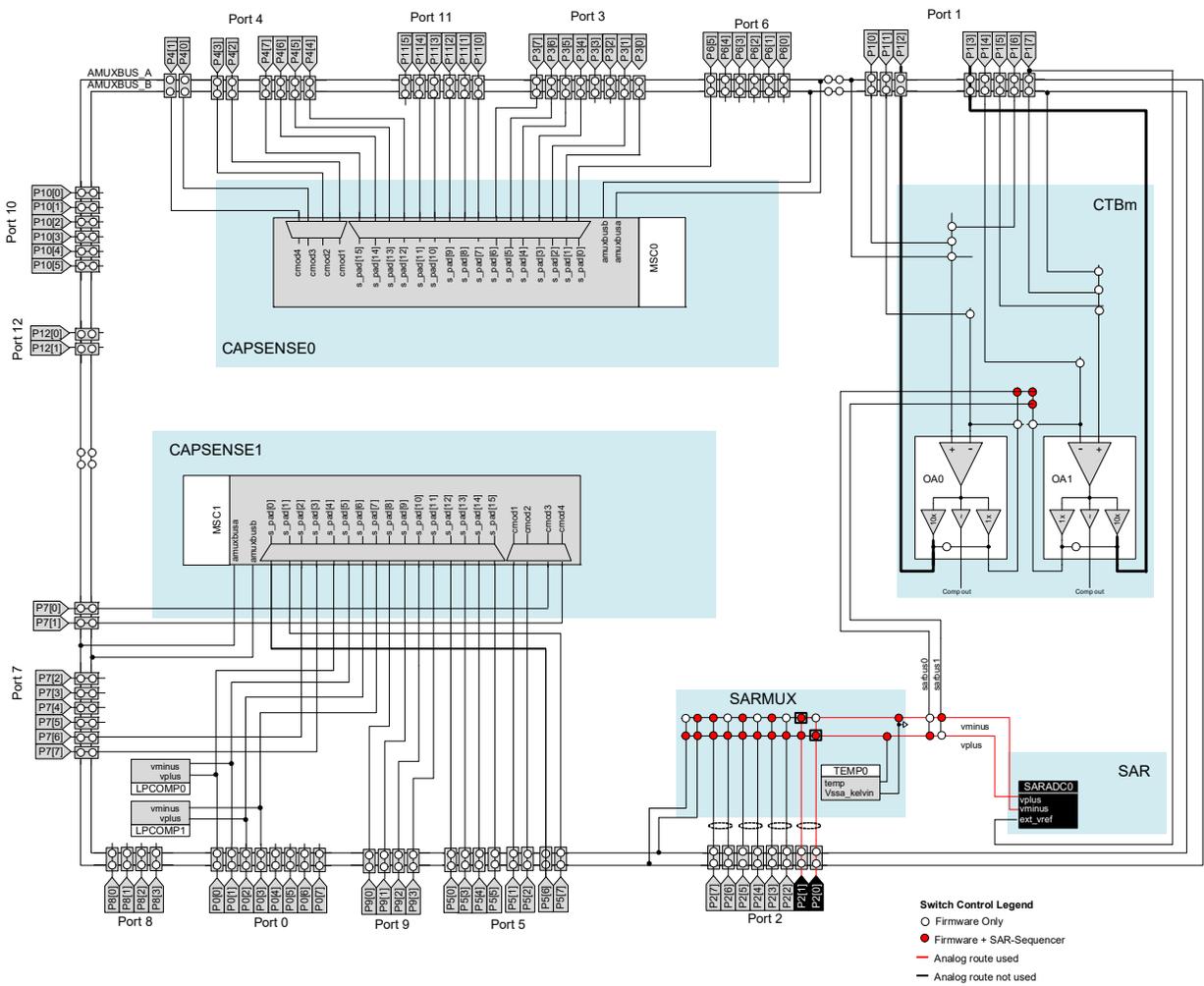
PSoC 4 analog interconnection is very flexible. SAR ADC can be connected to multiple inputs via SARMUX, including both external pins and internal signals. For example, it can connect to a neighboring block such as CTBm. It can also connect to other pins except port 2 through AMUXBUS\_A/B, at the expense 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 27-2 shows how two GPIOs that support SARMUX are connected to SAR ADC as a differential pair (Vplus/Vminus) via switches. These two switches can be controlled by sequencer, or firmware. The pins are arranged in adjacent pairs; for example, in SARMUX port P2[0] and P2[1], P2[2] and P2[3], and so on. If you need to use pins that are not paired as a differential pair, such as P2[1] and P2[2], the sequencer does not work; use firmware.

Figure 27-2. Input from External Pins

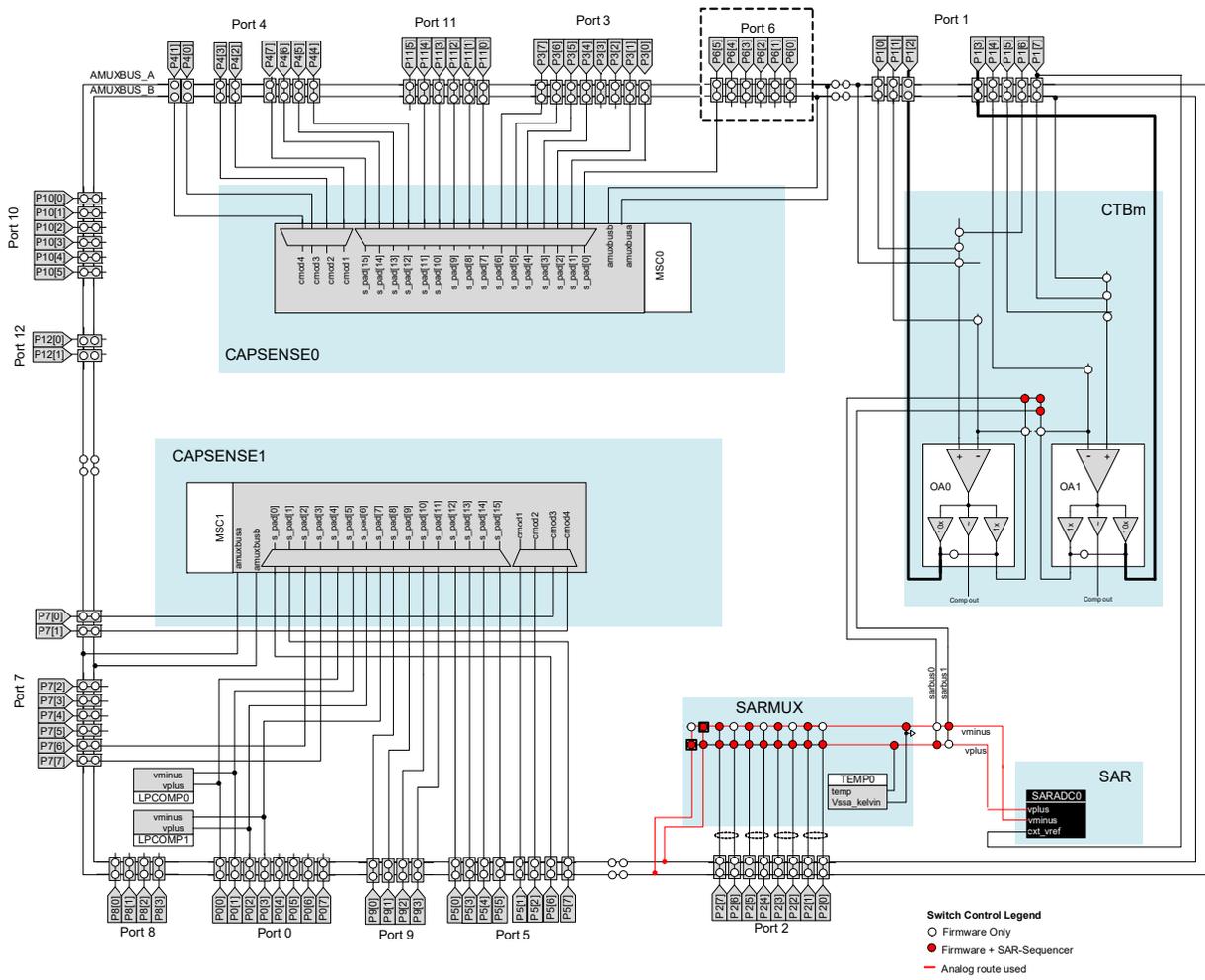


### Input from Analog Bus (AMUXBUS\_A/B)

Figure 27-3 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 AMUXBUS\_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 Msp/s. This is not recommended for external signals; the dedicated SARMUX port should be used, if possible.

Figure 27-3. Input from Analog Bus



### Input from CTBm Output via sarbus

SAR ADC can be connected to CTBm output via sarbus 0/1. **Figure 27-4** 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 27-5** 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. There are also additional switches, so it is not fast enough to sample at 1 Msp/s. However, the on-chip opamps add value for many applications.

Figure 27-4. Input from CTBm Output via sarbus

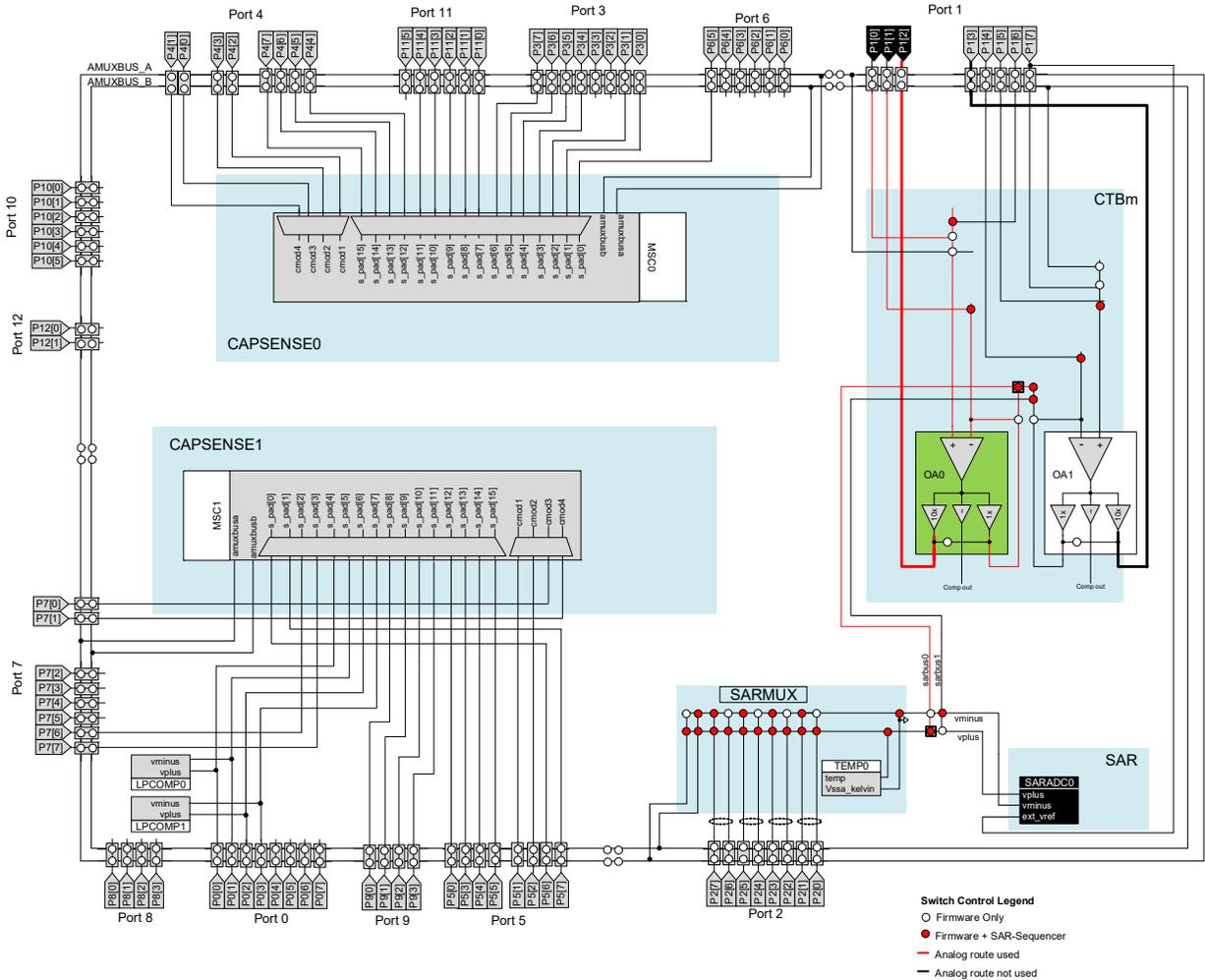
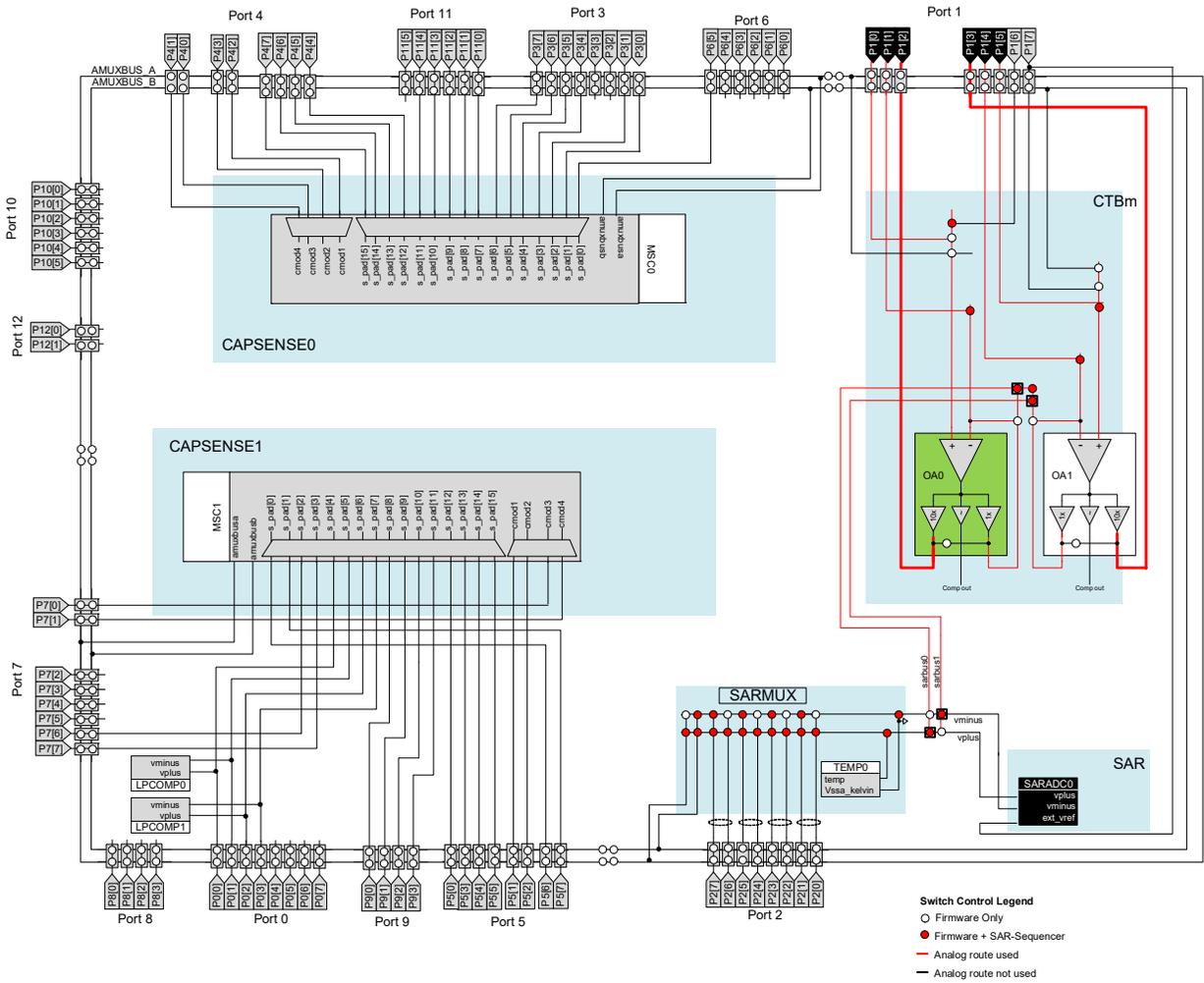


Figure 27-5. Inputs from CTBm Output via sarbus0 and sarbus1

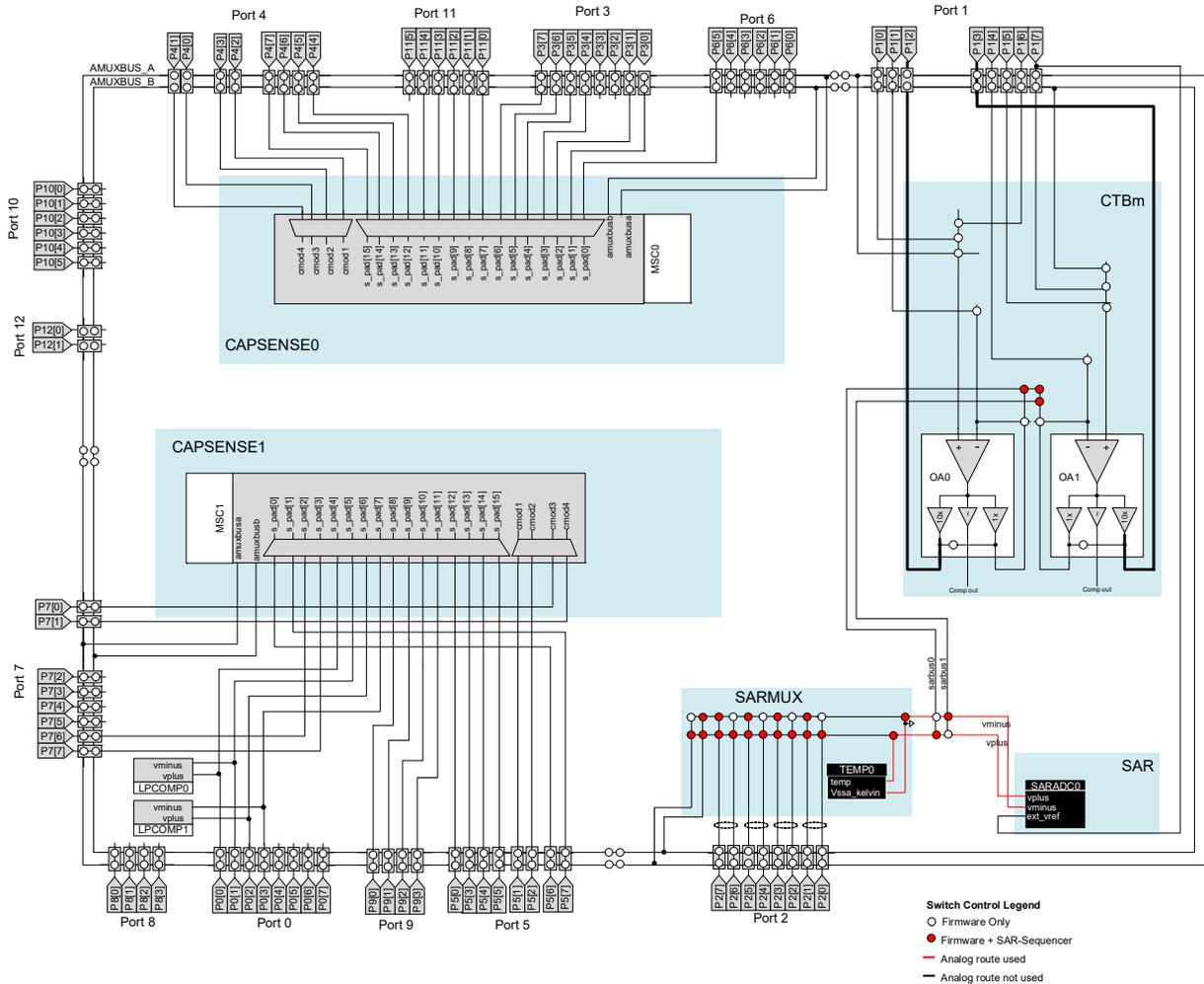


### Input from Temperature Sensor

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 27-6 shows, temperature sensor can be routed to positive input of SAR ADC via switch, which can be controlled by sequencer, firmware. 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 27-6. Inputs from Temperature Sensor



**Note:** Certain analog blocks such as CapSense and LPCOMP have additional internal routing, which is not shown in the above Figure 27-1 to Figure 27-6. See the respective chapters of these blocks in this document for details.

Analog routing in the PSoC 4 MCU consists of several on-chip analog buses and analog switches. [Table 27-1](#) gives a summary of connections available between analog blocks.

Table 27-1. Available Connections between PSoC 4 MCU Analog Blocks

|                             | To GPIOs                                      | To SAR ADC  | To CTBm (OA0, OA1)                            | To LPCOMP                                       |
|-----------------------------|---|---|---|---|
| <b>From GPIOs</b>           | AMUXBUS A<br>AMUXBUS B                        | Dedicated SARSEQ port<br>Through CTBm<br>AMUXBUS A<br>AMUXBUS B | Dedicated CTBm pins<br>AMUXBUS A<br>AMUXBUS B | Dedicated LPCOMP pins<br>AMUXBUS A<br>AMUXBUS B |
| <b>From SAR ADC</b>         | N/A <sup>a</sup>                              | N/A <sup>b</sup>  | N/A <sup>a</sup>                              | N/A <sup>a</sup>                                |
| <b>From CTBm (OA0, OA1)</b> | Dedicated CTBm pins<br>AMUXBUS A<br>AMUXBUS B | Internal bus<br>AMUXBUS A<br>AMUXBUS B                          | N/A <sup>b</sup>                              | AMUXBUS A<br>AMUXBUS B                          |
| <b>From LPCOMP</b>          | N/A <sup>c</sup>                              | N/A <sup>c</sup>  | N/A <sup>c</sup>                              | N/A <sup>b</sup>                                |

- a. SAR ADC does not have analog outputs.  
 b. Same block.  
 c. LPCOMP does not have analog outputs.

#### Notes:

- For detailed routing and switch-control information of a block, click on the respective block name.
- CapSense block connections used for touch sensing are not shown in [Table 27-1](#) because CapSense is not connected to any other analog blocks during touch sensing.
- If the routing uses switches belonging to a particular analog block, for example, SAR ADC, then SAR ADC block needs to be enabled.
- SAR ADCs are designed with 16 channels, but each ADC can scan thirteen unique inputs, which includes eight GPIOs from dedicated SARMUX port, two CTBm opamp outputs, two AMUX buses, and a temperature sensor. It is possible to scan GPIOs from other ports using AMUX buses.

Most analog blocks have dedicated connections to certain pins, which, when used, provide the best possible performance. However, these blocks can be also connected to other GPIOs and each other using AMUXBUS in case the dedicated pin is not available or is used by another resource.

### 27.2.2 AMUXBUS Splitting

AMUXBUS A and AMUXBUS B are system-wide analog buses that can connect to almost all analog blocks and some GPIOs in the PSoC 4 MCU. However, the PSoC 4 MCU has only two of these buses. Each AMUXBUS can be split into multiple segments using three switches controlled by writing into the HSIOM\_AMUX\_SPLIT\_CTL registers. For more information on AMUXBUS connections, see the [I/O System chapter on page 66](#).

## 27.3 Register List

Table 27-2. Register Summary

| Name                | Description   |
|---------------------|---|
| HSIOM_AMUX_SPLIT_CT | This register controls the breaking of AMUX buses A and B into multiple segments. |

# Section F: Program and Debug

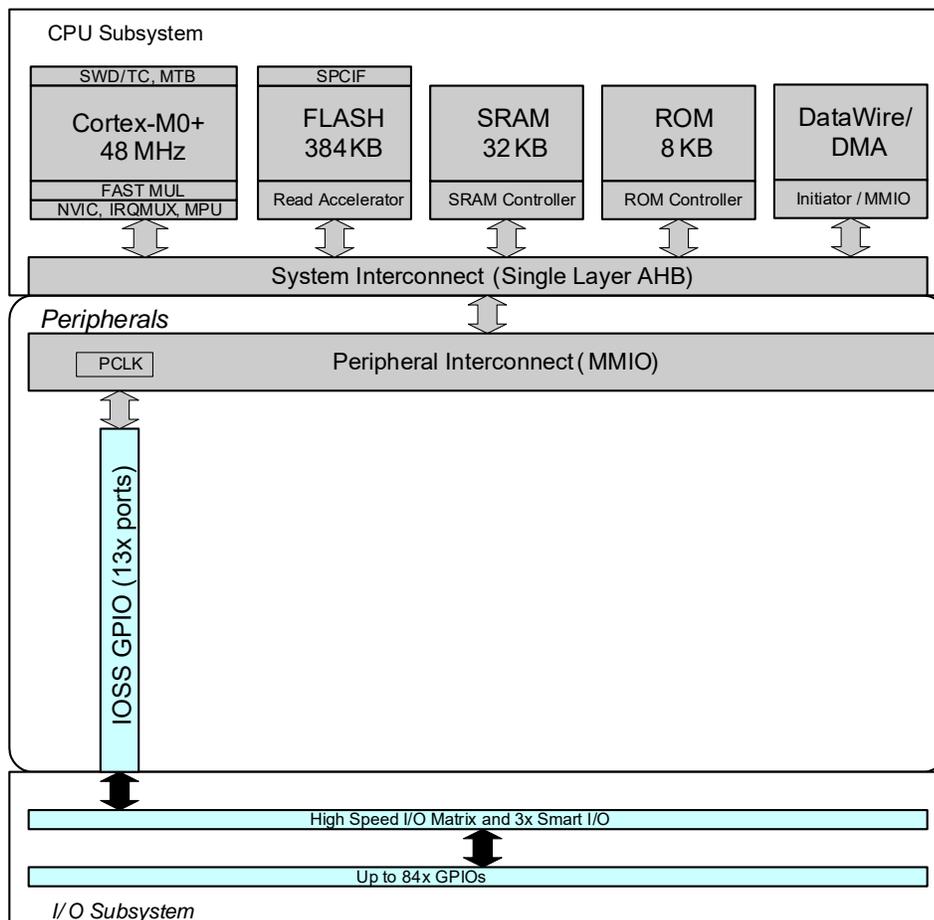


This section encompasses the following chapters:

- [Program and Debug Interface](#) chapter on page 366
- [Nonvolatile Memory Programming](#) chapter on page 374

## Top Level Architecture

Program and Debug Block Diagram





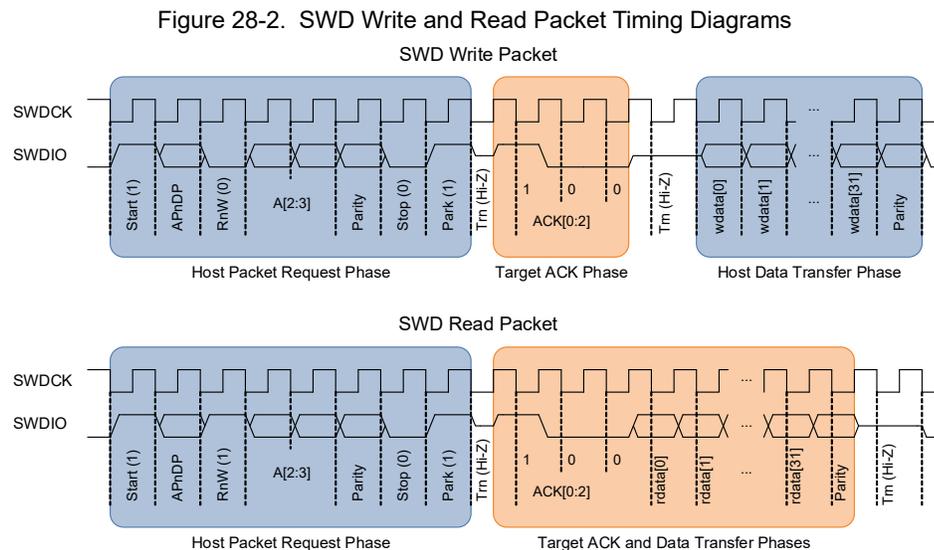
## 28.3 Serial Wire Debug (SWD) Interface

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 ( $T_{rn}$ ) 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 28-2 shows the timing diagrams of read and write SWD packets.



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 28-3](#) and [Table 28-4](#) 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 28-1](#) 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.

### 28.3.1 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 28-1 and Figure 28-2 illustrate the timing of SWDIO bit writes and reads.

Table 28-1. 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 |            |              |

### 28.3.2 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 28-2 shows the ACK bit-field decoding details.

Table 28-2. 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.

### 28.3.3 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 28-2](#). 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.

## 28.4 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](#).

### 28.4.1 Debug Port (DP) Registers

[Table 28-3](#) 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 28-3. 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 0x0BC11477.  |
| 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.   |

## 28.4.2 Access Port (AP) Registers

Table 28-4 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 28-4. 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                                 |

## 28.5 Programming the PSoC 4 Device

PSoC 4 is programmed using the following sequence. Refer to see 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.

### 28.5.1 SWD Port Acquisition

#### 28.5.1.1 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 < ms); //retry connection until OK ACK or timeout

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

if (ID != CM0P_ID) return FAIL; //confirm SWD ID of Cortex-M0+ CPU. (0x0BC11477)
```

In this pseudo code, SWD\_LineReset() is the standard Arm command to reset the 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 ( 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.

### 28.5.2 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](#). The minimum required clock frequency for the Port Acquire step and this step to succeed is 1.5 MHz.

### 28.5.3 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 374](#). The exact sequence of calling the programming routines is given in the [CY8C4xxx, CYBLxxxx Programming Specifications](#).

## 28.6 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.

### 28.6.1 Debug Control and Configuration Registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the [Armv6-M Architecture Reference Manual](#) for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0P\_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0P\_DFSR) – This register describes the reason a debug event has occurred and includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0P\_DCRSR) – This register is used to select the general-purpose register in the Cortex-M0+ CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0P\_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0P\_DCRSR register.
- Debug Exception and Monitor Control Register (CM0P\_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

### 28.6.2 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 (CM0P\_BP\_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in the PSoC 4).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0P\_BP\_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

### 28.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows:

- The watchpoint compare (CM0P\_DWT\_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT\_COMPx register.
- The watchpoint mask (CM0P\_DWT\_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0P\_DWT\_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be 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 (CM0P\_DWT\_PCSR) stores the current value of the program counter. This register is used for coarse, non-invasive profiling of the program counter register.

## 28.6.4 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 correct CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the [I/O System chapter on page 66](#) 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 63](#). Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

## 28.7 Registers

Table 28-5. List of Registers

| Register Name      | Description                                  |
|--------------------|--|
| CM0P_DHCSR         | Debug Halting Control and Status Register    |
| CM0P_DFSR          | Debug Fault Status Register                  |
| CM0P_DCRSR         | Debug Core Register Selector Register        |
| CM0P_DCRDR         | Debug Core Register Data Register            |
| CM0P_DEMCR         | Debug Exception and Monitor Control Register |
| CM0P_BP_CTRL       | Breakpoint control register                  |
| CM0P_BP_COMPx      | Breakpoint Compare Register                  |
| CM0P_DWT_COMPx     | Watchpoint Compare Register                  |
| CM0P_DWT_MASKx     | Watchpoint Mask Register                     |
| CM0P_DWT_FUNCTIONx | Watchpoint Function Register                 |
| CM0P_DWT_PCSR      | Watchpoint Comparator PC Sample Register     |

# 29. 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 bootloader operations where the CPU will update a portion of the flash memory.

## 29.1 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

## 29.2 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 by [CY8C4xxx, CYBLxxx 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 bootloader 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 117](#)) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

**Note** PSoC 4 implements a User Supervisory Flash (SFlash), which can be used to store application-specific information. These rows are not part of the hex file; their programming is optional.

## 29.3 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.

## 29.4 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.

## 29.4.1 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\_SYSREQ 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\_SYSREQ 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 29-1](#) 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\_SYSREQ and CPUSS\_SYSARG registers to fetch the values returned.

## 29.5 System Calls

Table 29-1 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 63](#) 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 29-1. List of System Calls

| System Call              | Description  | DAP Access |           |      | CPU Access |
|--------------------------|--|------------|-----------|------|------------|
|                          |  | Open       | Protected | Kill |            |
| Silicon ID               | Returns the device Silicon ID, Family ID, and Revision ID  | ✓          | ✓         | –    | ✓          |
| Load Flash Bytes         | Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 128 bytes  | ✓          | –         | –    | ✓          |
| Write Row                | Erases and then programs a row of flash with data in the page latch buffer   | ✓          | –         | –    | ✓          |
| Program Row              | Programs a row of flash with data in the page latch buffer   | ✓          | –         | –    | ✓          |
| Erase All                | Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area   | ✓          | –         | –    |            |
| Checksum                 | Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash  | ✓          | ✓         | –    | ✓          |
| Write Protection         | This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)   | ✓          | ✓         | –    |            |
| Non-Blocking Write Row   | Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access | –          | –         | –    | ✓          |
| Non-Blocking Program Row | Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access                 | –          | –         | –    | ✓          |
| Resume Non-Blocking      | Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access   | –          | –         | –    | ✓          |

## 29.5.1 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         |
|------------------------------|---------------------|---------------------|
| <b>CPUSS_SYSARG Register</b> |                     |                     |
| Bits [7:0]                   | 0xB6                | Key1                |
| Bits [15:8]                  | 0xD3                | Key2                |
| Bits [31:16]                 | 0x0000              | Not used            |
| CPUSS_SYSREQ register        |                     |                     |
| Bits [15:0]                  | 0x0000              | Silicon ID opcode   |
| Bits [31:16]                 | 0x8000              | Set SYSCALL_REQ bit |

### Return

| Address               | Return Value      | Description  |
|-----------------------|-------------------|--|
| CPUSS_SYSARG register |                   |  |
| Bits [7:0]            | Silicon ID Lo     |  |
| Bits [15:8]           | Silicon ID Hi     | 2E00-2EFF  |
| Bits [19:16]          | Minor Revision Id | See the <a href="#">CY8C4xxx, CYBLxxx Programming Specifications</a> for these values. |
| Bits [23:20]          | Major Revision Id |  |
| Bits [27:24]          | 0xFF              | Not used (don't care)  |
| Bits [31:28]          | 0xA               | Success status code  |
| CPUSS_SYSREQ register |                   |  |
| Bits [11:0]           | Family ID         | Family ID is 0xBE for PSoC 4100S Max   |
| Bits [15:12]          | Chip Protection   | See the <a href="#">Device Security</a> chapter on page 63.                            |
| Bits [31:16]          | 0XXXXX            | Not used   |

## 29.5.2 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 HFCLK (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.

### 29.5.3 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 128 bytes. Data is loaded into the page latch buffer starting at the location specified by the “Byte Addr” input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS\_SYSARG register. Note that the starting parameter address should be a word-aligned address.

#### Parameters

| Address   | Value to be Written      | Description  |
|---|--------------------------|--|
| SRAM Address - 32'hYY (32-bit wide, word-aligned SRAM address)    |                          |  |
| Bits [7:0]  | 0xB6                     | Key1   |
| Bits [15:8]   | 0xD7                     | Key2   |
| Bits [23:16]  | Byte Addr                | Start address of page latch buffer to write data<br>0x00 – Byte 0 of latch buffer<br>0x7F – Byte 127 of latch buffer |
| Bits [31:24]  | Flash Macro Select       | 0x00 – Flash Macro 0<br>0x01 – Flash Macro 1<br>0x02 – Flash Macro 2   |
| SRAM Address- 32'hYY + 0x04                                       |                          |  |
| Bits [7:0]  | Load Size                | Number of bytes to be written to the page latch buffer.<br>0x00 – 1 byte<br>0x7F – 128 bytes                         |
| Bits [15:8]   | 0xFF                     | Don't care parameter   |
| Bits [23:16]  | 0xFF                     | Don't care parameter   |
| Bits [31:24]  | 0xFF                     | 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) |

## 29.5.4 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 HFCLK (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 4100S Max: PSoc 4 Registers TRM* for more information.

### Parameters

| Address   | Value to be Written | Description   |
|---|---------------------|---|
| SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address) |                     |   |
| Bits [7:0]  | 0xB6                | Key1  |
| Bits [15:8]   | 0xD8                | Key2  |
| Bits [31:16]  | Row ID              | Row number to write<br>0x0000 – Row 0   |
| CPUSS_SYSARG register   |                     |   |
| Bits [31:0]   | 32'hYY              | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register   |                     |   |
| Bits [15:0]   | 0x0005              | Write Row opcode  |
| Bits [31:16]  | 0x8000              | Set SYSCALL_REQ bit   |

### Return

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

### 29.5.5 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 HFCLK (clk\_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

#### Parameters

| Address   | Value to be Written | Description   |
|---|---------------------|---|
| SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address) |                     |   |
| Bits [7:0]  | 0xB6                | Key1  |
| Bits [15:8]   | 0xD9                | Key2  |
| Bits [31:16]  | Row ID              | Row number to program<br>0x0000 – Row 0   |
| 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) |

## 29.5.6 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 HFCLK (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) |

## 29.5.7 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   |
|------------------------|---------------------|---|
| CPIUSS_SYSARG register |                     |   |
| Bits [7:0]             | 0xB6                | Key1  |
| Bits [15:8]            | 0xDE                | Key2  |
| Bits [31:16]           | Row ID              | Selects the flash row number on which the checksum operation is done<br>Row number – 16 bit flash row number<br>or<br>0x8000 – Checksum is performed on entire flash memory |
| CPIUSS_SYSREQ register |                     |   |
| Bits [15:0]            | 0x000B              | Checksum opcode   |
| Bits [31:16]           | 0x8000              | Set SYSCALL_REQ bit   |

### Return

| Address                | Return Value | Description  |
|------------------------|--------------|--|
| CPIUSS_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 |

### 29.5.8 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 HFCLK (clk\_hf) are set to IMO at 48 MHz. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

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

#### Parameters

| Address               | Value to be Written    | Description  |
|-----------------------|------------------------|--|
| CPUSS_SYSARG register |                        |  |
| Bits [7:0]            | 0xB6                   | Key1   |
| Bits [15:8]           | 0xE0                   | Key2   |
| Bits [23:16]          | Device Protection Byte | Parameter applicable only for Flash Macro 0<br>0x01 – OPEN mode<br>0x02 – PROTECTED mode<br>0x04 – KILL mode |
| Bits [31:24]          | Flash Macro Select     | 0x00 – Flash Macro 0<br>0x01 – Flash Macro 1<br>0x02 – Flash Macro 2   |
| 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     |                       |

## 29.5.9 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 375](#).

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 HFCLK (clk\_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0+ CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

### Parameters

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

### Return

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

## 29.5.10 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 375. 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 HFCLK (clk\_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0+ CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

### Parameters

| Address  | Value to be Written | Description   |
|--|---------------------|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) |                     |   |
| Bits [7:0]   | 0xB6                | Key1  |
| Bits [15:8]  | 0xDB                | Key2  |
| Bits [31:16]   | Row ID              | Row number to write<br>0x0000 – Row 0   |
| CPUSS_SYSARG register  |                     |   |
| 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]           | 0xFFFFFFFF   | Not used (don't care) |

## 29.5.11 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 375](#).

### Parameters

| Address  | Value to be Written | Description   |
|--|---------------------|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) |                     |   |
| Bits [7:0]   | 0xB6                | Key1  |
| Bits [15:8]  | 0xDC                | Key2  |
| Bits [31:16]   | 0XXXXX              | Don't care. Not used by SROM  |
| CPUSS_SYSARG register  |                     |   |
| Bits [31:0]  | 32'hYY              | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register  |                     |   |
| Bits [15:0]  | 0x0009              | Resume Non-Blocking opcode  |
| Bits [31:16]   | 0x8000              | Set SYSCALL_REQ bit   |

### Return

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

## 29.6 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 0xAFFFFFFF, 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 29-2. System Call Status Codes

| Status Code<br>(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. |
| F000001h   | Invalid Chip Protection Mode – This API is not available during the current chip protection mode.   |
| F000003h   | 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.                 |
| F000004h   | Invalid Address – The row ID or byte address provided is outside of the available memory.   |
| F000005h   | Row Protected – The row ID provided is a protected row.   |
| F000007h   | Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API.   |
| F000008h   | Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other APIs may be called.                               |
| F000009h   | System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume.                               |
| F00000Ah   | Checksum Zero Failed – The calculated checksum was not zero.  |
| F00000Bh   | Invalid Opcode – The opcode is not a valid API opcode.  |
| F00000Ch   | Key Opcode Mismatch – The opcode provided does not match key1 and key2.   |
| F00000Eh   | Invalid Start Address – The start address is greater than the end address provided.   |
| F000012h   | 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.                             |

## 29.7 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_IUSER_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_IUSER_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 52](#) for details on configuring the CM0+ exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the *iStatusInt* variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.