PSoC 5 TRM

CY8C55, CY8C54, CY8C53, CY8C52

PSoC® 5 Architecture TRM

(Technical Reference Manual)

Document No. 001-69820 Rev. *D

September 26, 2012

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

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

Copyrights

Copyright © 2011-2012 Cypress Semiconductor Corporation. All rights reserved.

PSoC and CapSense are registered trademarks of Cypress Semiconductor Corporation. PSoC Designer is a trademark of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Purchase of I^2C components from Cypress or one of its sublicensed Associated Companies conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips. As from October 1st, 2006 Philips Semiconductors has a new trade name, NXP Semiconductors.

The information in this document is subject to change without notice and should not be construed as a commitment by Cypress. While reasonable precautions have been taken, Cypress assumes no responsibility for any errors that may appear in this document. No part of this document may be copied, or reproduced for commercial use, in any form or by any means without the prior written consent of Cypress. Made in the U.S.A.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Flash Code Protection

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

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

Contents Overview

Section A: Overview **19**

1. Introduction 21
2. Getting Started 27
3. Document Construction 29

Section B: CPU System **33**

4. Cortex™-M3 Microcontroller 35
5. PSoC 5 Cache Controller 49
6. PHUB and DMAC 51
7. Interrupt Controller 67

Section C: Memory **77**

10. Nonvolatile Latch 79
11. SRAM 81
12. Flash Program Memory 83
13. EEPROM 85
14. Memory Map 87

Section D: System Wide Resources **89**

16. Clocking System 91
17. Power Supply and Monitoring 105
18. Low Power Modes 111
19. Watchdog Timer 117
20. Reset 119
21. I/O System 125
22. Flash Protection 143

Section E: Digital System **147**

23. Universal Digital Blocks (UDBs) 149
24. UDB Array and Digital System Interconnect 191
25. USB 199
26. Timer, Counter, and PWM 211
27. I^2C 227
28. Digital Filter Block (DFB) 241

Section F: Analog System **257**

30. Switched Capacitor/Continuous Time 259
31. Analog Routing 273
32. Comparators 289
33. Opamp 293
34. LCD Direct Drive 297
35. CapSense® 305
36. Digital-to-Analog Converter 311
37. Precision Reference 315
38. Delta Sigma Converter 319
39. Successive Approximation Register ADC 339

Section G: Program and Debug **343**

41. Test Controller 345
42. Cortex-M3 Debug and Trace 351
43. Nonvolatile Memory Programming 357

Glossary **363**

Index **379**

Contents

Section A: Overview 19

Document Revision History 19

1. Introduction 21

1.1 Top Level Architecture 21
1.2 Features 23
1.3 CPU System 23
1.3.1 Processor 23
1.3.2 Interrupt Controller 23
1.3.3 DMA Controller 23
1.3.4 Cache Controller 23
1.4 Memory 23
1.5 System Wide Resources 24
1.5.1 I/O Interfaces 24
1.5.2 Internal Clock Generators 24
1.5.3 Power Supply 24
1.5.3.1 Sleep Modes 24
1.6 Digital System 24
1.7 Analog System 25
1.7.1 Delta Sigma ADC 25
1.7.2 Successive Approximation Register ADC 25
1.7.3 Digital Filter Block 25
1.7.4 Digital-to-Analog Converters 25
1.7.5 Additional Analog Subsystem Components 25
1.8 Program and Debug 25

2. Getting Started 27

2.1 Support 27
2.2 Product Upgrades 27
2.3 Development Kits 27

3. Document Construction 29

3.1 Major Sections 29
3.2 Documentation Conventions 29
3.2.1 Register Conventions 29
3.2.2 Numeric Naming 30
3.2.3 Units of Measure 30
3.2.4 Acronyms 30

Section B: CPU System 33

Top Level Architecture 33

4. Cortex™-M3 Microcontroller 35

4.1 Features 35

4.2 How it Works 36
4.2.1 Registers 36
4.2.1.1 Special Registers 38
4.2.2 Operating Modes 39
4.2.3 Pipelining 40
4.2.4 Thumb-2 Instruction Set 40
4.2.4.1 Data Processing Operations 40
4.2.4.2 Load Store Operations 40
4.2.4.3 Branch Operations 41
4.2.4.4 Instruction Barrier and Memory Barrier Instructions 41
4.2.4.5 Saturation Operations 41
4.2.5 SysTick Timer 41
4.2.6 Debug and Trace: 41
4.3 Memory Map 42
4.3.1 Bus Interface to SRAM Memory 42
4.4 Exceptions 44
4.4.1 Priority Definitions 44
4.4.2 Fault Exceptions 44
4.4.3 System Call Exceptions 45
4.5 Nested Vector Interrupt Controller (NVIC) 45
4.5.1 Basic Interrupt Configuration 46
4.5.1.1 Example Procedure to Set Up an Interrupt 46
4.5.2 Nested Interrupts 46
4.5.3 Tail-Chaining Interrupts 46
4.5.4 Late Arrivals 47
4.5.5 Interrupt Latency 47
4.5.6 Faults Related to Interrupts 47

5. PSoC 5 Cache Controller 49

5.1 Features 49
5.2 Block Diagram 49
5.3 Cache Enable and Disable 50
5.4 Invalidate Cache Line 50
5.4.1 Measuring Cache Hits or Misses 50
5.5 Wait States when Reading from Flash 50
5.6 Sleep Mode Behavior 50
5.7 Cache Limitations 50

6. PHUB and DMAC 51

6.1 PHUB 51
6.1.1 Features 51
6.1.2 Block Diagram 51
6.1.3 How It Works 52
6.1.4 Arbiter 53
6.2 DMA Controller 53
6.2.1 Local Memory 53
6.2.2 How the DMAC Works 54
6.2.2.1 Interspoke Transfers 54
6.2.2.2 Intraspoke Transfer 55
6.2.2.3 Handling Multiple DMA Channels 56
6.2.2.4 DMA Channel Priority 56
6.2.2.5 DMA Latency in case of Nonideal Conditions 58
6.2.2.6 Request per Burst Bit 64

6.2.2.7 Work Sep Bit 64
6.3 Low-Power Mode Behavior 64
6.4 DMAC Access 64
6.4.1 SRAM Access 64
6.4.2 SIO Port Access 64
6.4.3 Port Configuration register Access 64
6.4.4 USB End Point Access 64
6.5 DMA Transaction Modes 64
6.5.1 Simple DMA 64
6.5.2 Auto Repeat DMA 65
6.5.3 Ping Pong DMA 65
6.5.4 Circular DMA 65
6.5.5 Indexed DMA 65
6.5.6 Scatter Gather DMA 65
6.5.7 Packet Queuing DMA 65
6.5.8 Nested DMA 66
6.6 Register List 66

7. Interrupt Controller 67

7.1 Features 67
7.2 Block Diagram 67
7.3 How It Works 68
7.3.1 Enabling Interrupts 69
7.3.2 Pending Interrupts 69
7.3.3 Interrupt Priority 70
7.3.4 Level versus Pulse Interrupt 70
7.3.5 Interrupt Execution 70
7.4 PSoC 5 Features 72
7.4.1 Active Interrupts 72
7.4.2 Interrupt Nesting 72
7.4.3 Interrupt Vector Addresses 74
7.4.4 Tail Chaining 74
7.4.5 Late Arrival Interrupts 74
7.4.6 Exceptions 75
7.4.7 Interrupt Masking 75
7.5 Interrupt Controller and Power Modes 76

Section C: Memory 77

Top Level Architecture 77

10. Nonvolatile Latch 79

10.1 Features 79
10.2 Write Once NV Latch 79
10.3 Programming NV Latch 80
10.4 Sleep Mode Behavior 80

11. SRAM 81

11.1 Features 81
11.2 Block Diagram 81
11.3 How It Works 82

12. Flash Program Memory 83

12.1 Features 83
12.2 Block Diagram 83

12.3 How It Works 84
12.4 Flash Memory Access Arbitration 84

13. EEPROM 85

13.1 Features 85
13.2 Block Diagram 85
13.3 How It Works 86

14. Memory Map 87

14.1 Features 87
14.2 Block Diagram 87
14.3 How It Works 87
14.3.1 PSoC 5 Memory Map 88

Section D: System Wide Resources 89

Top Level Architecture 89

16. Clocking System 91

16.1 Features 91
16.2 Block Diagram 92
16.3 Clock Sources 92
16.3.1 Internal Oscillators 92
16.3.1.1 Internal Main Oscillator 92
16.3.1.2 Internal Low Speed Oscillator 93
16.3.2 External Oscillators 94
16.3.2.1 MHz Crystal Oscillator 94
16.3.2.2 32.768 kHz Crystal Oscillator 94
16.3.3 Oscillator Summary 95
16.3.4 DSI Clocks 95
16.3.5 Phase-Locked Loop 95
16.4 Clock Distribution 97
16.4.1 Master Clock Mux 98
16.4.2 USB Clock 99
16.4.3 Clock Dividers 100
16.4.3.1 Single Cycle Pulse Mode 100
16.4.3.2 50% Duty Cycle Mode 100
16.4.3.3 Early Phase Option 101
16.4.4 Clock Synchronization 101
16.4.5 Phase Selection and Control 102
16.4.6 Divider Update 103
16.4.7 Power Gating of Clock Outputs 103
16.4.8 System Clock 103
16.4.9 Asynchronous Clocks 103
16.5 Low Power Mode Operation 103
16.6 Clock Naming Summary 104

17. Power Supply and Monitoring 105

17.1 Features 105
17.2 Block Diagram 106
17.3 How It Works 107
17.3.1 Power Supply Sequencing and Dependencies 107
17.3.2 Regulator Summary 107
17.3.2.1 Internal Regulators 107
17.3.2.2 Hibernate Regulator 107

17.3.3 Voltage Monitoring 107
17.3.3.1 Low Voltage Interrupt 108
17.3.3.2 High Voltage Interrupt 108
17.3.3.3 Processing a Low/High Voltage Detect Interrupt 108
17.4 Register Summary 109

18. Low Power Modes 111

18.1 Features 111
18.2 Active Mode 112
18.2.1 Entering Active Mode 112
18.2.2 Exiting Active Mode 112
18.3 Alternative Active Mode 112
18.3.1 Entering Alternative Active Mode 113
18.3.2 Exiting Alternative Active Mode 113
18.4 Sleep Mode 113
18.4.1 Entering Sleep Mode 113
18.4.2 Exiting Sleep Mode 113
18.5 Hibernate Mode 113
18.5.1 Entering Hibernate Mode 114
18.5.2 Exiting Hibernate Mode 114
18.6 Timewheel 114
18.6.1 Central Timewheel (CTW) 114
18.6.2 Fast Timewheel (FTW) 114
18.7 Register List 114

19. Watchdog Timer 117

19.1 Features 117
19.2 Block Diagram 117
19.3 How It Works 118
19.3.1 Enabling and Disabling the WDT 118
19.3.2 Setting the WDT Time Period and Clearing the WDT 118
19.4 Register List 118

20. Reset 119

20.1 Reset Sources 119
20.1.1 Initial Power-on Reset 119
20.1.2 Watchdog Reset 119
20.1.3 Software Initiated Reset 119
20.1.4 External Reset 119
20.1.5 Identifying Reset Sources 119
20.2 Reset Diagram 120
20.3 Boot Process and Timing 121
20.3.1 Manufacturing Configuration NV Latch 123
20.3.1.1 Device Configuration NV Latch 123
20.3.2 Boot Phase 123
20.3.3 User Mode 123
20.4 Register List 123

21. I/O System 125

21.1 Features 125
21.2 Block Diagrams 126
21.3 How It Works 128
21.3.1 Usage Modes and Configuration 128
21.3.2 I/O Drive Modes 128

21.3.2.1 Drive Mode on Reset 130
21.3.2.2 High Impedance Analog 130
21.3.2.3 High Impedance Digital 130
21.3.2.4 Resistive Pull Up or Resistive Pull Down 130
21.3.2.5 Open Drain, Drives High and Drives Low 130
21.3.2.6 Strong Drive 130
21.3.2.7 Resistive Pull Up and Pull Down 130
21.3.3 Slew Rate Control 130
21.3.4 Digital I/O Controlled by Port Register 130
21.3.4.1 Port Configuration Registers 131
21.3.4.2 Pin Wise Configuration Register Alias 131
21.3.4.3 Port Wide Configuration Register Alias 132
21.3.5 Digital I/O Controlled Through DSI 132
21.3.5.1 DSI Output 132
21.3.5.2 DSI Input 133
21.3.5.3 DSI for Output Enable Control 133
21.3.6 Analog I/O 134
21.3.7 LCD Drive 134
21.3.8 CapSense 135
21.3.9 SIO Functions and Features 135
21.3.9.1 Regulated Output Level 135
21.3.9.2 Adjustable Input Level 135
21.3.9.3 Hot Swap 136
21.3.10 Special Functionality 136
21.3.11 I/O Port Reconfiguration 138
21.3.12 Power Up I/O Configuration 138
21.3.13 Overvoltage Tolerance 138
21.3.14 I/O Power Supply 138
21.3.15 Sleep Mode Behavior 139
21.3.16 Low Power Behavior 139
21.4 Port Interrupt Controller Unit 139
21.4.1 Features 139
21.4.2 Interrupt Controller Block Diagram 139
21.4.3 Function and Configuration 140
21.5 Register Summary 141

22. Flash Protection 143

22.1 Flash Protection 143
22.2 Device Security 144
22.3 Frequently Asked Questions About Best Practices for Flash Protection and Device Security ... 145

Section E: Digital System 147

Top Level Architecture 147

23. Universal Digital Blocks (UDBs) 149

23.1 Features 149
23.2 Block Diagram 149
23.3 How It Works 150
23.3.1 PLDs 150
23.3.1.1 PLD Macrocells 151
23.3.1.2 PLD Carry Chain 153
23.3.1.3 PLD Configuration 153

23.3.2 Datapath 153
23.3.2.1 Overview 155
23.3.2.2 Datapath FIFOs 156
23.3.2.3 FIFO Status 163
23.3.2.4 Datapath ALU 163
23.3.2.5 Datapath Inputs and Multiplexing 165
23.3.2.6 CRC/PRS Support 166
23.3.2.7 Datapath Outputs and Multiplexing 168
23.3.2.8 Datapath Parallel Inputs and Outputs 170
23.3.2.9 Datapath Chaining 170
23.3.2.10 Dynamic Configuration RAM 171
23.3.3 Status and Control Module 172
23.3.3.1 Status and Control Mode 173
23.3.3.2 Control Register Operation 175
23.3.3.3 Parallel Input/Output Mode 176
23.3.3.4 Counter Mode 177
23.3.3.5 Sync Mode 178
23.3.3.6 Status and Control Clocking 178
23.3.3.7 Auxiliary Control Register 178
23.3.3.8 Status and Control Register Summary 179
23.3.4 Reset and Clock Control Module 179
23.3.4.1 Clock Control 180
23.3.4.2 Reset Control 182
23.3.4.3 UDB POR Initialization 184
23.3.5 UDB Addressing 185
23.3.5.1 Working Register Address Space 185
23.3.5.2 Configuration Register Address Space 187
23.3.5.3 UDB Configuration Address Space 187
23.3.5.4 Routing Configuration Address Space 187
23.3.6 System Bus Access Coherency 188
23.3.6.1 Simultaneous System Bus Access 188
23.3.6.2 Coherent Accumulator Access (Atomic Reads and Writes) 188
23.4 UDB Working Register Reference 189

24. UDB Array and Digital System Interconnect 191

24.1 Features 191
24.2 Block Diagram 191
24.3 How It Works 192
24.4 UDB Array System Interface 194
24.4.1 UDB Array POR Initialization 194
24.4.2 UDB POR Configuration Sequence 195
24.4.2.1 Quadrant Route Disable 196
24.4.3 UDB Sleep and Power Control 196
24.4.4 UDB Register References and Address Mapping 196

25. USB 199

25.1 Features 199
25.2 Block Diagram 200
25.2.1 Serial Interface Engine (SIE) 200
25.2.2 Arbiter 201
25.2.2.1 SIE Interface Module 202
25.2.2.2 CPU Interface Block 202
25.2.2.3 Memory Interface 202

25.2.2.4 DMA Interface 202
25.2.2.5 Arbiter Logic 202
25.2.2.6 Synchronization Block 202
25.3 How it Works 202
25.3.1 Operating Frequency 202
25.3.2 Operating Voltage 203
25.3.3 Transceiver 203
25.3.4 Endpoints 203
25.3.5 Transfer Types 203
25.3.6 Interrupts 204
25.4 Logical Transfer Modes 204
25.4.1 No DMA Access 206
25.4.2 Manual DMA Access 207
25.4.3 Control Endpoint Logical Transfer 208
25.5 PS/2 and CMOS I/O Modes 209
25.6 Register List 209

26. Timer, Counter, and PWM 211

26.1 Features 211
26.2 Block Diagram 211
26.3 How It Works 212
26.3.1 Clock Selection 212
26.3.2 Enabling and Disabling Block 213
26.3.3 Input Signal Characteristics 213
26.3.3.1 Enable Signal 214
26.3.3.2 Capture Signal 214
26.3.3.3 Timer Reset Signal 215
26.3.3.4 Kill Signal 215
26.3.4 Operating Modes 215
26.3.4.1 Timer Mode – Free Run Mode 215
26.3.4.2 Gated Timer Mode 217
26.3.4.3 Pulse-width Modulator Mode 221
26.3.4.4 One Shot Mode 224
26.3.5 Interrupt Enabling 224
26.3.6 TC Interrupt Behavior during Reset: 225
26.3.7 Sleep Mode Behavior 225
26.4 Register Listing 225

27. I^2C 227

27.1 Features 227
27.2 Block Diagram 227
27.3 Background Information 228
27.3.1 I^2C Bus Description 228
27.3.2 Typical I^2C Data Transfer 228
27.4 How It Works 229
27.4.1 Bus Stalling (Clock Stretching) 229
27.4.2 System Management Bus 229
27.4.3 Pin Connections 229
27.4.4 I2C Interrupts 229
27.4.5 Control by Registers 230
27.4.6 Operating the I^2C Interface 230
27.4.6.1 Slave Mode 231
27.4.6.2 Master Mode 232

27.4.6.3 Multi-Master Mode....233
27.5 Slave Mode Transfer Examples....233
27.5.1 Slave Receive....234
27.5.2 Slave Transmit....235
27.6 Master Mode Transfer Examples....236
27.6.1 Single Master Receive....236
27.6.2 Single Master Transmit....237
27.7 Multi-Master Mode Transfer Examples....238
27.7.1 Multi-Master, Slave Not Enabled....238
27.7.2 Multi-Master, Slave Enabled....239

28. Digital Filter Block (DFB) 241

28.1 Features....241
28.2 Block Diagram....241
28.3 How It Works....242
28.3.1 Controller....242
28.3.1.1 FSM RAM....243
28.3.1.2 Program Counter....244
28.3.1.3 Control Store....244
28.3.1.4 Next State Decoder....244
28.3.2 Datapath....245
28.3.2.1 MAC....246
28.3.2.2 ALU....246
28.3.2.3 Shifter and Rounder....246
28.3.3 Address Calculation Unit....247
28.3.4 Bus Interface and Register Descriptions....247
28.3.4.1 Streaming Mode....247
28.3.4.2 Block Transfer Modes....248
28.3.4.3 Result Handling....249
28.3.4.4 Data Alignment....251
28.3.4.5 DMA and Semaphores....251
28.3.4.6 DSI Routed Inputs and Outputs....251
28.4 DFB Instruction Set....252
28.5 Usage Model....255

Section F: Analog System 257

Top Level Architecture....257

30. Switched Capacitor/Continuous Time 259

30.1 Features....259
30.2 Block Diagram....259
30.3 How it Works....261
30.3.1 Operational Mode of Block is Set....261
30.4 Naked Opamp....261
30.4.1 Bandwidth/Stability Control....261
30.4.1.1 BIAS_CONTROL....261
30.4.1.2 SC_COMP[1:0]....262
30.4.1.3 SC_REDC[1:0]....262
30.5 Continuous Time Unity Gain Buffer....263
30.6 Continuous Time Programmable Gain Amplifier....263
30.7 Continuous Time Transimpedance Amplifier....264
30.8 Continuous Time Mixer....266
30.9 Sampled Mixer....267

30.10 Delta Sigma Modulator 269
30.10.1 First-Order Modulator, Incremental Mode 270
30.11 Track and Hold Amplifier 271

31. Analog Routing 273

31.1 Features 273
31.2 Block Diagram 273
31.3 How it Works 276
31.3.1 Analog Globals (AGs) 276
31.3.2 Analog Mux Bus (AMUXBUS) 276
31.3.3 Liquid Crystal Display Bias Bus (LCDBUS) 276
31.3.4 Analog Local Bus (abus) 278
31.3.5 Switches and Multiplexers 278
31.3.5.1 Control of Analog Switches 278
31.4 Analog Resource Blocks – Routing and Interface 280
31.4.1 Digital-to-Analog Converter (DAC) 281
31.4.2 Comparator 282
31.4.3 Delta Sigma Modulator (DSM) 283
31.4.4 Switched Capacitor 284
31.4.5 Opamp 285
31.4.6 Low Pass Filter (LPF) 285
31.5 Track Jumping 286
31.6 Mitigating Analog Routes with Degraded Low Power Signal Integrity 286
31.7 Analog Routing Register Summary 287

32. Comparators 289

32.1 Features 289
32.2 Block Diagram 289
32.3 How it Works 290
32.3.1 Input Configuration 290
32.3.2 Power Configuration 290
32.3.3 Output Configuration 290
32.3.4 Hysteresis 291
32.3.5 Comparator Clock 291
32.3.6 Offset Trim 291
32.3.7 Register Summary 292

33. Opamp 293

33.1 Features 293
33.2 Block Diagram 294
33.3 How it Works 294
33.3.1 Input and Output Configuration 294
33.3.2 Power Configuration 294
33.3.3 Buffer Configuration 295
33.3.4 Register Summary 295

34. LCD Direct Drive 297

34.1 Features 297
34.2 LCD Drive System 297
34.2.1 Functional Description 298
34.2.1.1 LCD DAC 298
34.2.1.2 Configurations Options for DAC 299
34.2.1.3 LCD Driver Block 300
34.2.1.4 UDB 302

34.2.1.5 DMA 303
34.3 Low Power Mode Behavior (Sleep/Hibernate) 303
34.4 LCD Register Summary 304

35. CapSense® 305

35.1 Features 305
35.2 Block Diagram 305
35.3 How It Works 306
35.3.1 Reference Driver 306
35.3.2 Low Pass Filter 306
35.3.3 Analog Mux Bus 306
35.3.4 GPIO Configuration for CapSense 306
35.3.5 Other Resources 307
35.4 CapSense Delta Sigma Algorithm 308

36. Digital-to-Analog Converter 311

36.1 Features 311
36.2 Block Diagram 311
36.3 How It Works 312
36.3.1 Current DAC 312
36.3.2 Voltage DAC 312
36.3.3 Output Routing Options 313
36.3.4 Making a Higher Resolution DAC 313
36.4 Register List 314

37. Precision Reference 315

37.1 Block Diagram 315
37.2 How It Works 315
37.2.1 Precision Voltage and Current Reference 317
37.2.2 Reference Generators 317
37.2.3 Powerdown Logic 318
37.2.4 Low Power Mode Behavior 318
37.3 Registers 318

38. Delta Sigma Converter 319

38.1 Features 319
38.2 Block Diagram 319
38.3 How It Works 320
38.3.1 Input Buffer 320
38.3.2 Delta Sigma Modulator 321
38.3.2.1 Clock Selection 322
38.3.2.2 Capacitance Configuration 322
38.3.2.3 Gain Configuration 323
38.3.2.4 Power Configuration 324
38.3.2.5 Other Configuration Options 328
38.3.2.6 Quantizer 328
38.3.2.7 Reference Options 328
38.3.2.8 Reference for DSM: Usage Guidelines 331
38.3.3 Analog Interface 332
38.3.3.1 Conversion of Thermometric Code to Two's Complement 333
38.3.3.2 Modulation Input 333
38.3.3.3 Clock Selection and Synchronization 333
38.3.4 Decimator 333

38.3.4.1 Start and End of Conversion 333
38.3.4.2 Shifters 333
38.3.4.3 CIC Filter 334
38.3.4.4 Post Processing Filter 334
38.3.5 Coherency Protection 335
38.3.5.1 Protecting Writes (Gain/Offset) with Coherency Checking 336
38.3.5.2 Protecting Reads (Output Sample) with Coherency Checking 336
38.3.6 Modes of Operation 336

39. Successive Approximation Register ADC 339

39.1 Features 339
39.2 How It Works 340
39.2.1 Input Selection 340
39.2.2 Clock Selection 340
39.2.3 Input Sampling 340
39.2.4 Power Mode 340
39.2.5 Reference Selection 340
39.2.6 Operational Modes 340
39.2.7 SAR ADC Output 341

Section G: Program and Debug 343

Top Level Architecture 343

41. Test Controller 345

41.1 Features 345
41.2 Block Diagram 345
41.3 Background Information 345
41.3.1 Serial Wire Debug Interface 345
41.4 Serial Wire Debug (SWD) Interface 345
41.5 How It Works 347
41.5.1 Clocking 347
41.5.2 DP/AP Access Register 348
41.5.3 Debug Port and Access Port Registers (PSoC 5) 348
41.6 Test Controller Acquisition 349
41.6.1 Functional Test 349
41.6.2 Programming Flash/EEPROM 349
41.6.3 Program Debug/Trace 349

42. Cortex-M3 Debug and Trace 351

42.1 Features 351
42.2 How It Works 352
42.2.1 Test Controller (TC) 352
42.2.1.1 Debug Port and Access Port Registers 353
42.2.2 SWV Interface 353
42.3 Core Debug 353
42.3.1 Enabling the Debug 353
42.3.2 Halting 353
42.3.3 Stepping 353
42.3.4 Accessing PSoC Memory and Registers 353
42.4 System Debug 354
42.4.1 Flash Patch and Breakpoint (FPB) Unit 354
42.4.2 Data Watchpoint and Trace (DWT) 354
42.4.3 Instrumentation Trace Macrocell (ITM) 355
42.4.4 Single Wire Viewer (SWV) 355

42.4.4.1 Enabling SWV 355
42.4.4.2 Communicating with SWV 356
42.4.5 Using Multiple Interfaces Simultaneously 356

43. Nonvolatile Memory Programming 357

43.1 Features 357
43.2 Block Diagram 357
43.3 How It Works 358
43.3.1 Commands 358
43.3.1.1 Command Code Descriptions 359
43.3.1.2 Command Failure Codes 360
43.3.2 Register Summary 360
43.3.3 Flash Protection Settings 361

Glossary 363

Index 379

Section A: Overview

This document encompasses the PSoC® 5 CY8C55, CY8C54, CY8C53, and CY8C52 families. In conjunction with the device datasheet, it contains complete and detailed information about how to use and design with the IP blocks that construct a PSoC 5 device. This document describes the analog and digital architecture to give the designer a better understanding of features and limitations of PSoC 5. The routing of both digital and analog signals should be left to the tool (PSoC Creator™). Hand routing, analog or digital, by use of registers, may conflict with the routing performed by PSoC Creator and produce unexpected results.

This section encompasses the following chapters:

- Introduction chapter on page 21
- Getting Started chapter on page 27
- Document Construction chapter on page 29

See the *CY8C55, CY8C54, CY8C53, CY8C52 PSoC® 5 Registers TRM (Technical Reference Manual)* for complete register sets.

Document Revision History

Table 1-1. CY8C55, CY8C54, CY8C53, CY8C52 PSoC® 5 Architecture TRM (Technical Reference Manual) Revision History

Revision	Issue Date	Origin of Change	Description of Change
**	09/08/2011	KANT	Initial release
*A	10/28/2011	MKEA	Added note on PLL P&Q modification restriction to 16.3.5 Phase-Locked Loop on page 95 Added LVI glitching details to Processing a Low/High Voltage Detect Interrupt chapter on page 108. Removed 17.3.2.4 Reset on a Voltage Monitoring Interrupt Added VDAC double-write information to 36.3 How It Works on page 312 Updated Low Power mode information throughout document Updated 27.4.6.1 Slave Mode on page 231, 27.4.6.2 Master Mode on page 232, and 27.4.6.3 Multi-Master Mode on page 233 Updated 21.4.3 Function and Configuration on page 140 Updated output count information in 38.3.3.1 Conversion of Thermometric Code to Two's Complement on page 333 and Table 38-12 on page 334
*B	02/13/2012	XKJ/JPM	Removed information on Fault Recovery from 16.3.2.1 MHz Crystal Oscillator on page 94. Corrected supported IMO frequencies Added note to constrain XRES on powerup during the boot phase Removed Temperature Sensor chapter Added recommendation for sleep mode Removed reference to SSOP package

Table 1-1. CY8C55, CY8C54, CY8C53, CY8C52 PSoC® 5 Architecture TRM (Technical Reference Manual) Revision History

Revision	Issue Date	Origin of Change	Description of Change
*C	06/05/2012	VVSK	Fixed logo in the title page Added text on comparator output to 32.3.2 Power Configuration2 Updated 20.1.5 Identifying Reset Sources Added information on PLL charge pump to 16.3.5 Phase-Locked Loop Updated description of debug feature in 4. Cortex™-M3 Microcontroller Added information on SAR ADC taking more cycles to complete conversion in triggered mode Fixed multiple errors in 26. Timer, Counter, and PWM Added 12.4 Flash Memory Access Arbitration Multiple updates to 16. Clocking System Added information to the end of 18.5 Hibernate Mode Added information on reference instability when enabling or disabling analog peripherals in 37.2 How It Works Aded clarification on relation between flash protection and debug enabled to the end of 22.1 Flash Protection, 42.3.1 Enabling the Debug. Updated section 20.1.4 External Reset and 20.1.5 Identifying Reset Sources to distinguish between XRES and POR In 30.3.1 Operational Mode of Block is Set updated register from SC[0..3]_CR1 to SC[0..3]_CR0 Changed vcc_io, vio to Vddio Updated Figure 31-9 and Table 38-10
*D	09/26/2012	SHEA	Updated the link to the Cortex-M3 Technical Reference Manual in 4.2.4 Thumb-2 Instruction Set

1. Introduction

With a unique array of configurable digital and analog blocks, the Programmable System-on-Chip (PSoC®) is a true system-level solution, offering a modern method of signal acquisition, processing, and control with exceptional accuracy, high bandwidth, and superior flexibility. Its analog capability spans the range from thermocouples (DC voltages) to ultrasonic signals.

PSoC 5 (CY8C55xxx, CY8C54xxx, CY8C53xxx, CY8C52xxx) family has 32-bit PSoC platform devices with the following characteristics:

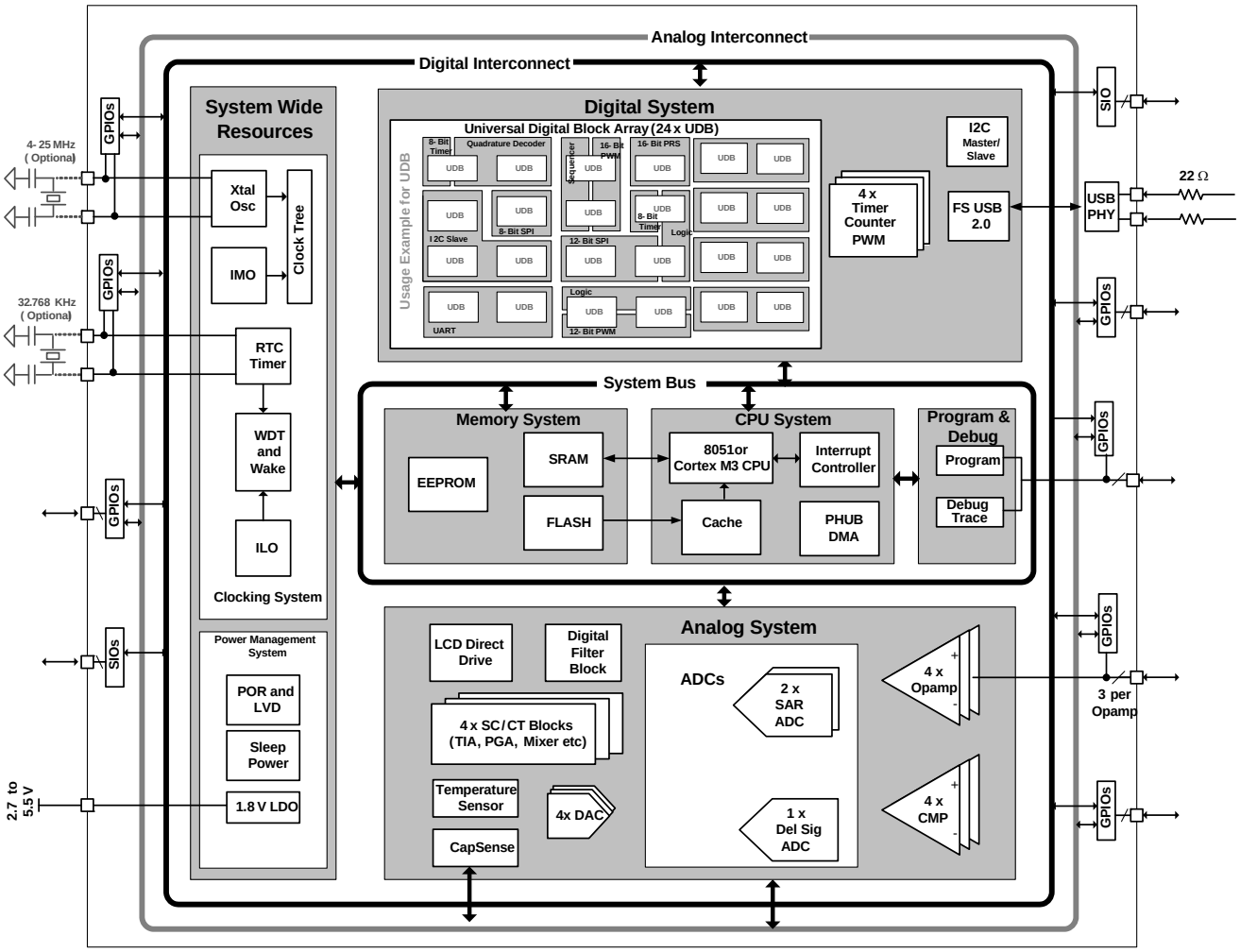
- Fully pin, peripheral compatible
- PSoC Creator, the integrated development environment software
- High performance, configurable digital system that supports a wide range of communication interfaces, such as USB, I2C, and SPI
- High precision, high performance analog system with up to 20-bit ADC, DACs, comparators, opamps, and programmable blocks to create PGAs, TIAs, mixers, etc.
- Easily configurable logic array
- Flexible routing to all pins
- High performance 32-bit ARM Cortex-M3 core

This document describes PSoC 5 devices in detail. Using this information, designers can easily create system-level designs, using a rich library of prebuilt components, or custom verilog, and a schematic entry tool that uses the standard design blocks. PSoC 5 devices provide unparalleled opportunities for analog and digital bill of materials (BOM) integration, while easily accommodating last-minute design changes.

1.1 Top Level Architecture

Figure 1-1 on page 22 shows the major components of PSoC 5 devices. PSoC 5 device uses the 32-bit Cortex M3 core.

Figure 1-1. Top Level Architecture for PSoC 5 Devices



1.2 Features

PSoC 5 devices have these major components. See Figure 1-1 on page 22.

- Cortex-M3 Central Processing Unit (CPU) with a nested vectored interrupt controller and a high performance DMA controller
- Several types of memory elements including SRAM, flash, and EEPROM
- System integration features, such as clocking, a feature-rich power system, and versatile programmable inputs and outputs
- Digital system that includes configurable Universal Digital Blocks (UDBs) and specific function peripherals, such as USB
- Analog subsystem that includes configurable switched capacitor (SC) and continuous time (CT) blocks, up to 20-bit Delta Sigma converters, 8-bit DACs that can be configured for 12-bit operation, more than one SAR ADC, comparators, PGAs, and more
- Programming and debug system through Serial Wire Debug (SWD), and Single Wire Viewer (SWV)

1.3 CPU System

1.3.1 Processor

The PSoC 5 CPU subsystem is built around a 32-bit three stage pipelined ARM Cortex-M3 processor running up to 67 MHz. The PSoC 5 instruction set is the same as the Thumb-2 instruction set available on standard Cortex- M3 devices.

1.3.2 Interrupt Controller

The CPU subsystem includes a programmable Nested Vectored Interrupt Controller (NVIC), DMA (Direct Memory Access) controller, Flash cache, and RAM. The NVIC of PSoC 5 devices provides low latency by allowing the CPU to vector directly to the first address of the

interrupt service routine, bypassing the jump instruction required by other architectures.

The PSoC 5 interrupt controller also offers a few advanced interrupt management capabilities, such as interrupt tail chaining to improve stack management with multiple pending interrupts providing lower latency.

1.3.3 DMA Controller

The DMA controller allows peripherals to exchange data without CPU involvement. This allows the CPU to run slower, save power, or use its cycles to improve the performance of firmware algorithms.

1.3.4 Cache Controller

In PSoC 5 devices, the flash cache also reduces system power consumption by reducing the frequency with which flash is accessed. The processor speed itself is configurable allowing for active power consumption tuned for specific applications.

1.4 Memory

The PSoC nonvolatile subsystem consists of Flash and byte-writable EEPROM.

The CPU can reprogram individual blocks of Flash, enabling boot loaders.

A powerful and flexible protection model allows the user to selectively lock blocks of memory for read and write protection, securing sensitive information. The byte-writable EEPROM is available on-chip for the storage of application data.

1.5 System Wide Resources

The individual elements of system wide resources are discussed in these sections.

1.5.1 I/O Interfaces

PSoC 5 devices have three I/O types:

- **General Purpose Input/Output (GPIO)** – Every GPIO has analog I/O, digital I/O, LCD drive, CapSense®, flexible interrupt, and slew rate control capability. PSoC 5 devices also provide up to four individual I/O voltage domains through the V_{DDIO} pins.
- **Special Input/Output (SIO)** – The SIOs on PSoC 5 devices allow the user to set VOH independently of VDDIO when used as outputs. When SIOs are in input mode, they are high impedance, even when the device is not powered or when the pin voltage goes above the supply voltage. This makes the SIO ideally suited for use on an I^2C bus where the PSoC 5 devices are not powered, even though other devices on the bus are powered. The SIO pins also have high current sink capability for applications such as LED drive.
- **USB Input/Output (USBIO)** – For devices with Full Speed USB, the USB physical interface is also provided (USBIO). When not using USB, these pins can be used for limited digital functionality and device programming. The functionality of the D+ and D– pins is limited by the reduced available drive modes and limited voltage range (refer to the USBIO DC Specifications in the PSoC 5 device data sheet).

1.5.2 Internal Clock Generators

PSoC devices incorporate flexible internal clock generators, designed for high stability and factory-trimmed for absolute accuracy. The Internal Main Oscillator (IMO) is the master clock base for the system with 5% absolute accuracy at 3 MHz. The IMO can be configured to run from 3 MHz up to 48 MHz. Multiple clock derivatives are generated from the main clock frequency to meet application needs.

PSoC 5 devices provide a PLL to generate system clock frequencies up to the maximum operating frequency of the device (67 MHz). The PLL can be driven from the IMO, an external crystal, or an external reference clock. The devices also contain a separate, very low power Internal Low Speed Oscillator (ILO) for the sleep and watchdog timers. The ILO provides two primary outputs, 1 kHz and 100 kHz. A 32.768 kHz external watch crystal is also supported for use in Real Time Clock (RTC) applications. The clocks, together with programmable clock dividers, provide the flexibility to integrate most timing requirements

1.5.3 Power Supply

PSoC 5 devices support extensive supply operating ranges from 2.7 V to 5.5 V, allowing operation from regulated supplies such as 3.3 V ± 10%, 5.0 V ± 10%, or directly from a wide range of battery types.

1.5.3.1 Sleep Modes

The PSoC platform supports four low power sleep modes, from the lowest current RAM retention mode (hibernation) to the full function active mode. Power to all major functional blocks, including the programmable digital and analog peripherals, is controlled independently by firmware.

This function allows low power background processing when some peripherals are not in use.

1.6 Digital System

The digital subsystems of PSoC 5 devices provide these devices their first half of unique configurability.

The subsystem connects a digital signal from any peripheral to any pin through the Digital System Interconnect (DSI). It also provides functional flexibility through an array of small, fast, low-power Universal Digital Blocks (UDBs).

Each UDB contains Programmable Array Logic (PAL) and Programmable Logic Device (PLD) functionality, together with a small state machine engine to support a wide variety of peripherals.

In addition to the flexibility of the UDB array, PSoC devices provide configurable digital blocks targeted at specific functions.

These blocks include 16-bit timer/counter/PWM blocks, I^2C slave/master/multi-master, and Full-Speed USB. See the device data sheet for a list of available specific function digital blocks.

1.7 Analog System

The PSoC analog subsystem provides the device the second half of its unique configurability. All analog performance is based on a highly accurate absolute voltage reference.

The configurable analog subsystem includes:

- Analog muxes
- Comparators
- Voltage references
- Opamps
- Mixers
- Trans Impedance Amplifiers (TIA)
- Analog-to-Digital Converters (ADC)
- Digital-to-Analog Converters (DAC)
- Digital Filter Block (DFB)

All GPIO pins can route analog signals into and out of the device, using the internal analog bus. This feature allows the device to interface up to 60 discrete analog signals.

1.7.1 Delta Sigma ADC

The heart of the analog subsystem is a fast, accurate, configurable Delta Sigma ADC. With less than 100 µV offset, a gain error of ±0.1%, Integral Non-Linearity (INL) less than 1 LSB, Differential Non-Linearity (DNL) less than 0.5 LSB, and signal-to-noise ratio (SNR) better than 90 dB (Delta Sigma) in 16-bit mode, this converter addresses a wide variety of precision analog applications, including some of the most demanding sensors.

1.7.2 Successive Approximation Register ADC

Another type of ADC seen on PSoC 5 devices is the Successive Approximation Register (SAR) ADC. Featuring 12-bit conversions at up to 1 Msps, it offers low nonlinearity, low offset errors, and an SNR better than 70 dB; it is well suited for a variety of higher-speed analog applications. Some PSoC devices offer both types of ADC and can have multiple instances of each. See the device datasheet for specific details.

1.7.3 Digital Filter Block

The output of the ADC can optionally feed the programmable Digital Filter Block (DFB) via DMA without CPU intervention. The DFB can be configured to perform IIR and FIR digital filters and a variety of user defined custom functions. The DFB can implement filters with up to 64 taps.

1.7.4 Digital-to-Analog Converters

Four high speed voltage or current DACs support 8-bit output signals at waveform frequencies up to 8 MHz and can be routed out of any GPIO pin. These DACs can be combined together to create a higher resolution 12-bit DAC.

Higher resolution voltage DAC outputs are created using the UDB array to create a pulse width modulated (PWM) DAC of up to 10 bits, at up to 48 kHz. The digital DACs in each UDB support PWM, PRS, or Delta Sigma algorithms with programmable widths.

1.7.5 Additional Analog Subsystem Components

In addition to the ADCs, DACs, and the DFB, the analog subsystem provides components such as multiple comparators, uncommitted opamps, and configurable Switched Capacitor/Continuous Time (SC/CT) blocks supporting trans impedance amplifiers, programmable gain amplifiers, and mixers.

1.8 Program and Debug

Serial Wire Debugger (SWD) (2-wire) interface is used for programming and debug. The 1-wire Single Wire Viewer (SWV) can also be used for “printf” style debugging. By combining SWD and SWV, the designer can implement a full debugging interface with just three pins.

Using these standard interfaces enables the designer to debug or program the PSoC device with a variety of hardware solutions from Cypress or third party vendors.

PSoC 5 devices support on-chip break points, and an instruction and data trace memory for debug. The PSoC 5 device offers many more advanced debugging features, such as Flash patch breakpoint capability to update instructions without reprogramming, fast “printf” style debugging using the Trace Port Interface Unit (TPIU) module, clock cycle counting capability, and various other features with Data Watchpoint and Trace (DWT) modules.

2. Getting Started

The quickest path to understanding any PSoC® device is to read the device datasheet and use PSoC Designer™ or PSoC Creator™ Integrated Development Environments (IDEs) software. This technical reference manual helps to understand the details of the PSoC 5 integrated circuit and its implementation.

For the most up-to-date ordering, packaging, or electrical specification information, refer to the individual PSoC device's datasheet or go to http://www.cypress.com/psoc.

2.1 Support

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

Applications Assistance can be reached at http://www.cypress.com/support/ or by phone at:
1-800-541-4736.

2.2 Product Upgrades

Cypress provides scheduled upgrades and version enhancements for PSoC Creator free of charge. Upgrades are available from your distributor on CD-ROM or download them directly from http://www.cypress.com under the Software tab. Also provided are critical updates to system documentation under the Documentation tab.

2.3 Development Kits

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

3. Document Construction

The content sections of this technical reference manual start after this section – Section A: Overview on page 19. The following sections include these topics:

- Section B: CPU System on page 33
- Section C: Memory on page 77
- Section D: System Wide Resources on page 89
- Section E: Digital System on page 147
- Section F: Analog System on page 257
- Section G: Program and Debug on page 343

3.1 Major Sections

The major sections of the technical reference manual are:

- Sections – Presents the top-level architecture, how to get started and conventions and overview information about any particular area that help inform the reader about the construction and organization of the product.
- Chapter – Presents the chapters specific to some individual aspect of the Section topic. These are the detailed implementation and use information for some aspect of the integrated circuit.
- Glossary – Defines the specialized terminology used in this technical reference manual. Glossary terms are presented in bold, italic font throughout.
- *PSoC® 5 Registers TRM (Technical Reference Manual)* – Supply all device register details summarized in the technical reference manual. These are additional documents.

For ease of use, information is organized into sections and chapters that are divided according to device functionality. Each section begins with some interpretation detail and contains a top level architectural explanation. This is followed by chapters that contain detailed explanation required for the implementation and use of the individual functions described. The *PSoC® 5 Registers TRM (Technical Reference Manual)* is contained in a separate .pdf file.

3.2 Documentation Conventions

There are only four distinguishing font types used in this document, 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® 5 Registers TRM (Technical Reference Manual)*.

3.2.2 Numeric Naming

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

3.2.3 Units of Measure

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

Table 3-1. Units of Measure

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

3.2.4 Acronyms

This table lists the acronyms that are used in this document

Table 3-2. Acronyms

Symbol	Unit of Measure
ABUS	analog output bus
AC	alternating current
ADC	analog-to-digital converter
API	application programming interface
APOR	analog power-on reset
BC	broadcast clock
BIFC	bit implemented functioning connection
BINC	bit implemented no connection
BOM	bill of materials
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CBUS	comparator bus
CI	carry in
CMP	compare
CMRR	common mode rejection ratio
CO	carry out
CPU	central processing unit
CRC	cyclic redundancy check
CT	continuous time
DAC	digital-to-analog converter
DAP	debug access port on ARM Cortex™-M3 of PSoC 5
DC	direct current
DFB	digital filter block
DoC	debug on-chip mode in PSoC 5
DI	digital or data input
DMA	direct memory access
DMAC	direct memory access controller
DNL	differential nonlinearity
DO	digital or data output
DSI	digital signal interface
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
FB	feedback
FSR	full scale range
GIE	global interrupt enable
GPIO	general purpose I/O
I^2C	inter-integrated circuit
ICE	In-circuit emulator
IDE	integrated development environment
ILO	internal low-speed oscillator
IMO	internal main oscillator
INL	integral nonlinearity

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
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
ISSP	In-system serial programming
IVR	interrupt vector read
LFSR	linear feedback shift register
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
MISO	master-in-slave-out
MOSI	master-out-slave-in
MSb	most significant bit
MSB	most significant byte
NVIC	nested vectored interrupt controller on Cortex-M3 of PSoC 5
PC	program counter
PCH	program counter high
PCL	program counter low
PD	power down
PGA	programmable gain amplifier
PHUB	peripheral hub
PICU	port interrupt control unit
PM	power management
PMA	PSoC memory arbiter
POR	power-on reset
PPOR	precision power-on reset
PRS	pseudo random sequence
PSoC®	Programmable System-on-Chip
PSRAM	pseudo SRAM
PSRR	power supply rejection ratio
PSSDC	power system sleep duty cycle
PVT	process voltage temperature
PWM	pulse-width modulator
RAM	random-access memory
RAS	row address strobe
RETI	return from interrupt
RO	relaxation oscillator
ROM	read only memory
RW	read/write
SAR	successive approximation register
SC	switched capacitor

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SWD	single wire debug
SWV	single wire viewer
TC	terminal count
TD	transaction descriptors
TIA	transimpedance amplifier
UDB	universal digital block
USB	universal serial bus
USBIO	USB I/O
VCO	voltage controlled oscillator
WDT	watchdog timer
WDR	watchdog reset
XRES_N	external reset, active low

Section B: CPU System

The PSoC 5 CPU subsystem is built around a 32-bit three stage pipelined ARM Cortex-M3 processor running up to 80 MHz.

This section encompasses the following chapters:

- Cortex™-M3 Microcontroller chapter on page 35
- PSoC 5 Cache Controller chapter on page 49
- PHUB and DMAC chapter on page 51
- Interrupt Controller chapter on page 67

Top Level Architecture

4. Cortex™-M3 Microcontroller

The PSoC 5 ARM Cortex-M3 core is a high performance, low power 32-bit central processing unit (CPU). It has an efficient Harvard 3-stage pipeline core, a fixed 4 GB memory map, and supports the 16- and 32-bit Thumb-2 instruction set. The Cortex-M3 also features hardware divide instructions and low-latency Interrupt Service Routine (ISR) entry and exit.

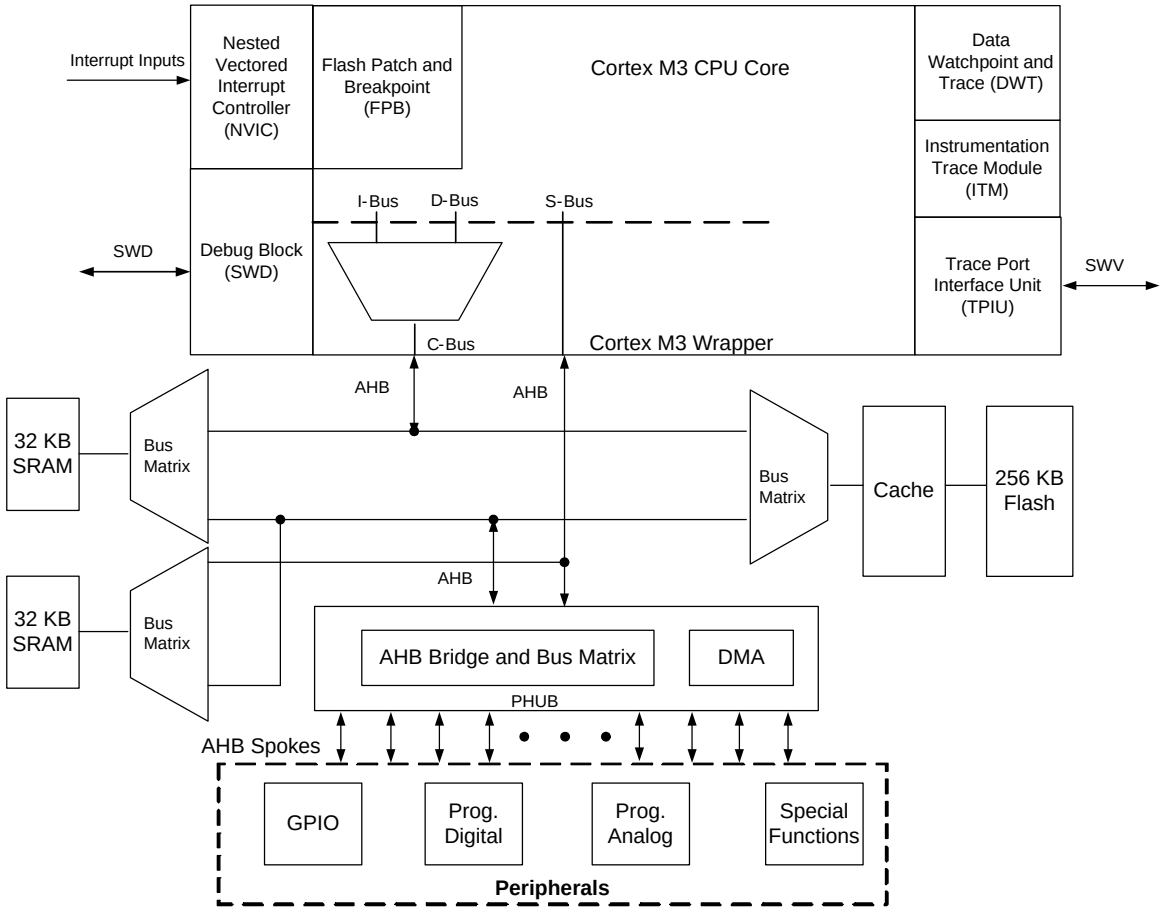
The Cortex-M3 processor includes a number of other components that are tightly linked to the CPU core. These include a Nested Vectored Interrupt Controller (NVIC), a SYSTICK timer, and numerous debug and trace blocks.

This section gives an overview of the Cortex-M3 processor. For further details, see the *ARM Cortex-M3 Technical Reference Manual* available at http://www.arm.com. Figure 4-1 shows a diagram of the Cortex-M3 and its interface to different blocks on the device.

4.1 Features

- Three stage pipelining operating at 1.25 DMIPS/MHz. This helps to increase execution speed or reduce power.
- Supports Thumb-2 instruction set:
  - Thumb-2 instruction set supports complex operations with both 16- and 32-bit instructions
  - Atomic bit level read and write instructions
  - Support for unaligned memory access
- Improved code density, ensuring efficient use of memory.
- Easy to use, ease of programmability and debugging:
  - Ensures easier migration from 8- and 16-bit processors
- NVIC unit to support interrupts and exceptions:
  - Helps to achieve rapid interrupt response
- Extensive debug support including:
  - Serial Wire Debug Port (SWD)
  - Break points
  - Flash patch
  - Code tracing

Figure 4-1. PSoC 5 Cortex-M3 Block Diagram



The bus interfaces in the Cortex-M3 are based on AHB-Lite (Advanced High Performance Bus-Lite) and the APB (Advanced Peripheral Bus) protocols.

The bus interfaces available in the Cortex-M3 are:

- I-Code Bus for instruction fetches
- D-Code Bus for data fetches
- System Bus for instruction and data fetches in memory regions 0x20000000 to 0xDFFFFFFF and 0xE0100000 to 0xFFFFFFFF
- External Private Peripheral Bus to debug components
- Debug Access Port to connect the debug interface blocks

4.2 How it Works

The Cortex-M3 is a 32-bit processor with a 32-bit data path, 32-bit register, and a 32-bit memory interface. It supports both 16-bit and 32-bit instructions in the Thumb-2 instruction set. Because the Cortex-M3 does not support the ARM instruction set, it is not backward compatible with the ARM7 processor.

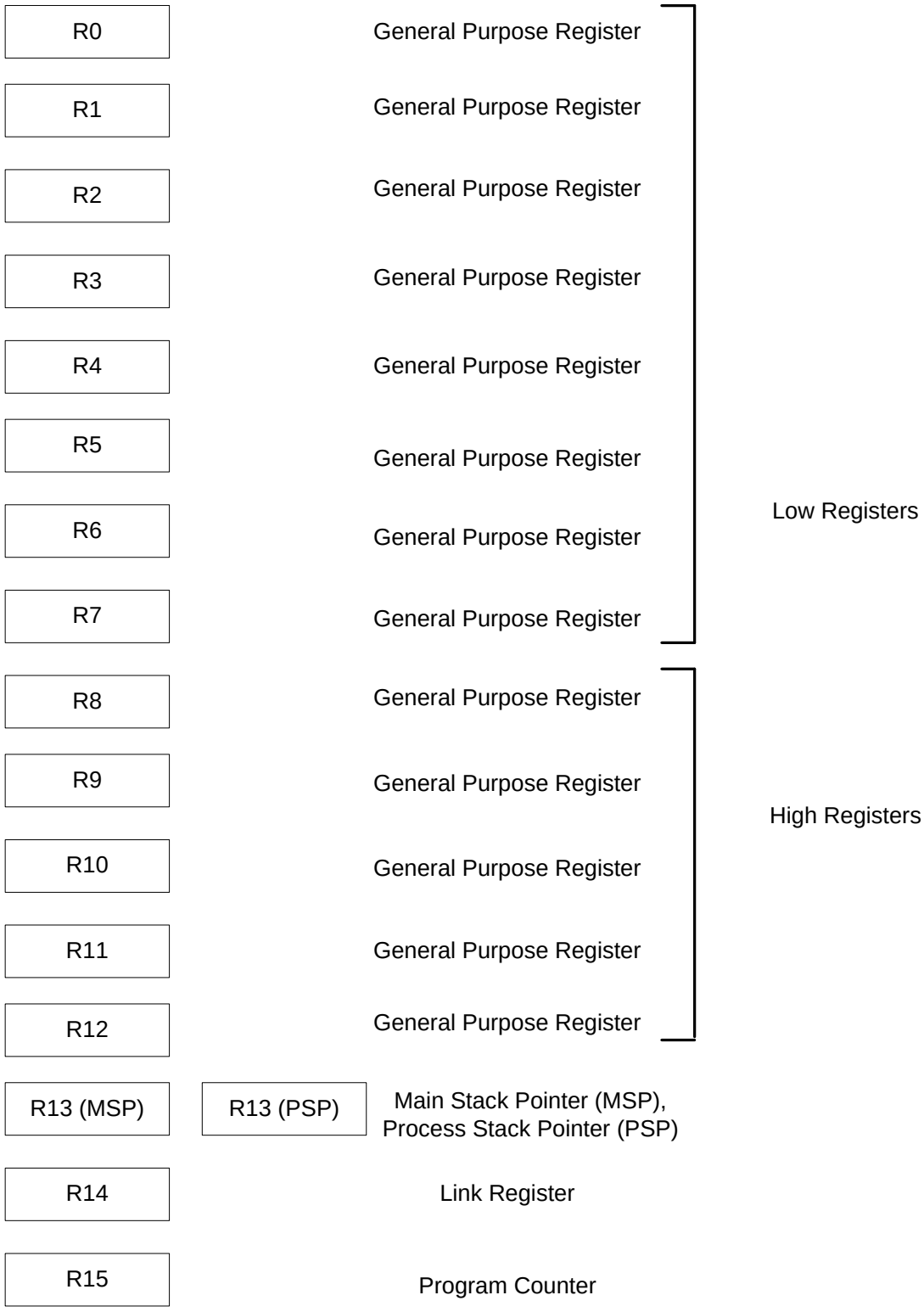
The processor supports two operating modes, and has a single cycle 32-bit multiplication instruction and hardware divide instructions.

4.2.1 Registers

The Cortex-M3 has 16 32-bit registers (Figure 4-2):

- R0 to R12 - general purpose registers
  - R0 to R7 – can be accessed by all instructions
  - R8 to R12 – can be accessed by all 32-bit and some 16-bit instructions
- R13 – Stack Pointer (SP). There are two stack pointers, with only one available at a time. The SP is always 32-bit word aligned; bits [1:0] are always ignored and considered to be ‘0’.
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

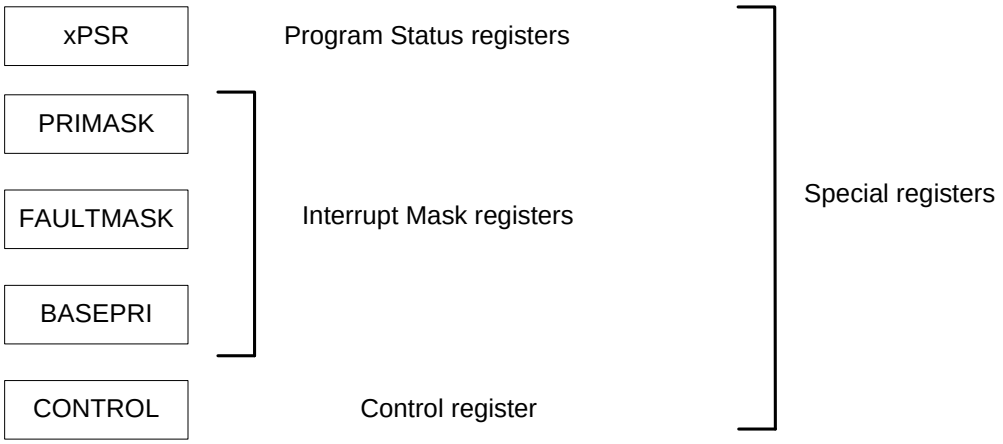
Figure 4-2. Cortex-M3 Registers



4.2.1.1 Special Registers

The special registers can only be accessed using special instructions and cannot be used for normal data processing. Cortex-M3 supports three sets of special registers:

Figure 4-3. Cortex-M3 Special Registers



Program Status Registers

These registers consist of:

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR)

These registers provide ALU flags (zero, carry), execution status, and current executing interrupt number. The three PSRs can be accessed separately or collectively, using the special instructions MSR and MRS. They can be collectively addressed as xPSR.

Figure 4-4. Cortex-M3 Program Status Registers

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8 : 0
xPSR	N	Z	C	V	Q	ICI/IT	T	--	--	ICI/IT	-	Exception Number

Where:

- N – Negative Flag
- Z – Zero Flag
- C – Carry/Borrow Flag
- V – Overflow Flag
- Q – Sticky Saturation Flag
- ICI / IT – Interrupt-Continual Instruction (ICI) bits / IF-THEN instruction status bit
- T – Thumb-2 Instruction. Always set to 1. Clearing this results in an exception
- Exception Number – Indicates which exception the processor is currently handling

Interrupt Mask Registers

- PRIMASK – Used to disable all interrupts except the Nonmaskable Interrupt (NMI) and HardFault
- FAULTMASK – Used to disable all interrupts except NMI
- BASEPRI – Used to disable interrupts of specified or lower priority levels.

These registers are used by the NVIC to mask an interrupt or exception.

Control Register

This register controls the stack pointer selection and the privilege level of the processor. It has only two bits:

CONTROL[0]

‘0’ Privileged in Thread Mode

‘1’ User state in Thread mode

CONTROL[1]

‘0’ Default stack is used

‘1’ Alternate stack is used

4.2.2 Operating Modes

The Cortex-M3 supports two privilege levels:

- Privileged – Code has no limit to resources
- User – Code has some limits to the resources

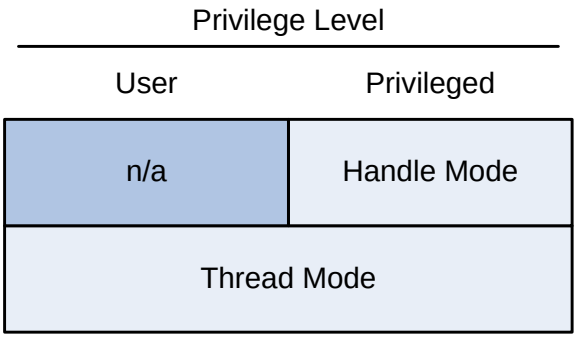
Privilege level can be controlled using the control register.

When the code is in user level, it cannot access the debug resources and certain important registers.

In addition to the privilege levels, the processor supports two types of operating modes:

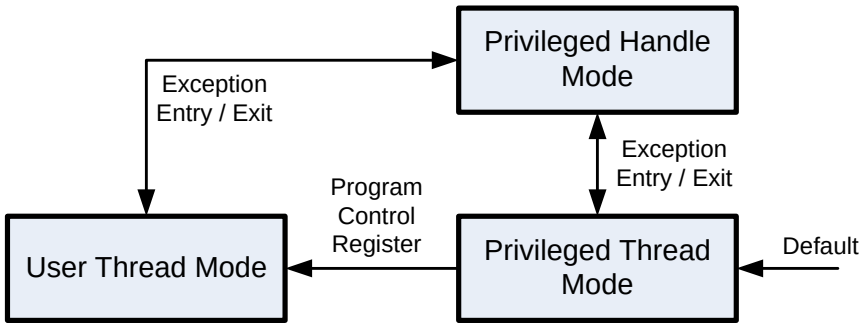
- **Thread Mode** – Thread mode is used by all normal applications. During the thread mode the Process Stack Pointer (PSP) is used. The thread mode can exist in both privileged level and user level. Switching from privileged level to user level can be done by just writing to the control register but the reverse cannot be done. When an exception occurs, the system is automatically taken to privileged level and at the exit of the exception it comes back to the user level. Restoring to the privileged level can be done only by going through an exception handler that programs the control register for the privileged mode.
- **Handle Mode** – Handle mode is used by OS kernel and exception handlers. During this mode, the main stack pointer (MSP) is used. The handle mode can exist only in the privileged level.

Figure 4-5. Operating Modes



Handle Mode: running an interrupt service routine
Thread Mode: running background code

Figure 4-6. Operating Mode Transitions



4.2.3 Pipelining

The three stage pipelining includes:

- Fetch – The instruction is fetched from memory
- Decode – Generating the addresses and branch prediction
- Execute – Instruction execution based on the address and branches

The branch prediction unit has been enhanced so that it gives nearly no ALU usage penalty.

Pipelining can give zero to two wait states when executing an instruction.

4.2.4 Thumb-2 Instruction Set

The Cortex-M3 supports a wide range of 16- and 32-bit instructions. It does not support all ARM instructions, including:

- Branch with link and exchange state
- Switch endian
- Certain coprocessor instructions
- Hint instructions
- DSP instructions
- Change process instructions

The instruction includes these data processing operations:

- Multiply and divide
- Bit
- Shift
- Load store
- Branch
- Barrier
- Exception generating
- System
- Saturation
- Miscellaneous

Cortex-M3 supports unique instructions. The following table is a summary of the important instructions:

Table 4-1. Cortex-M3 Unique Instructions

Instruction	Functionality
MSR, MRS	To access special registers
IT	IF-THEN instruction supporting up to 4 succeeding instructions
CBZ, CBNZ	Compare and then branch
SDIV, UDIV	Signed and Unsigned Divide
REV, REVH, REVSH	Reverse the byte order in data word, upper half word, lower half word, respectively

Table 4-1. Cortex-M3 Unique Instructions

Instruction	Functionality
RBIT	Reverses bit order in a data word
SXTB, SXTH, UXTB, UXTH	Extend a byte or half word into a word
BFC, BFI	BFC - Clears any number of adjacent bits in any position BFI – Copies any number of bits from any register to another register to any mentioned location
UBFX, SBFX	Unsigned and signed bit field extract instructions
LDRD, STRD	Transfer 2 words of data from or into 2 registers
TBB, TBH	Table Branch Byte and Table Branch Halfword for branch tables

The following sections detail some of the instruction types. For the entire summary of the instruction set, refer to the *Cortex-M3 Technical Reference Manual* available at http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf.

4.2.4.1 Data Processing Operations

The Cortex-M3 provides many different instructions for data processing. A few basics are introduced here. Many data operation instructions can have multiple instruction formats.

The Cortex-M3 supports arithmetic functions ADD, SUB (subtract), MUL (multiply), and UDIV/SDIV (unsigned and signed divide).

The Cortex-M3 supports 32-bit multiply instructions and multiply accumulate instructions that give 64-bit results. These instructions support signed or unsigned values.

Another group of data processing instructions are logical operations such as AND, ORR (or), EOR (exclusive OR), and rotate and shift functions. In some cases the rotate operation can be combined with other operations.

Another group of data processing instructions is used for reversing data bytes in a register. These instructions are usually used for conversion between little endian and big endian data.

The last group of data processing instructions is for bit field processing. Instructions such as BFC, BFI, SBFX, and UBFX are used to clear, set, and copy bits with sign extension or zero extension.

4.2.4.2 Load Store Operations

One of the most basic functions in a processor is transfer of data. In the Cortex-M3, data transfers can be one of the following types:

- Moving data between register and register
- Moving data between memory and register
- Moving data between special register and register

- Moving an immediate data value into a register

The command to move data between registers is MOV (move). For example, moving data from register R3 to register R8 looks similar to:

```
MOV R8, R3
```

Another instruction can generate the negative value of the original data; it is called MVN (move negative).

The basic instructions for accessing memory are Load and Store. Load (LDR) transfers data from memory to registers, and Store transfers data from registers to memory. The transfers can be in different data sizes (byte, half word, word, and double word).

Multiple Load and Store operations can be combined into single instructions called LDM (Load Multiple) and STM (Store Multiple).

ARM processors also support memory accesses with pre-indexing and post-indexing. Two other types of memory operation are stack PUSH and stack POP.

The Cortex-M3 has a number of special registers. To access these registers, use the instructions MRS and MSR.

4.2.4.3 Branch Operations

The branch operations include:

- Call and Unconditional branch instructions
- Decision and Conditional branch instructions
- Combined Compare and Conditional Branch
- Conditional Branching using IT instructions

The IT (IF-THEN) instruction block is very useful for handling small conditional code. It avoids branch penalties because there is no change to program flow. It can provide a maximum of four conditionally executed instructions with one condition check.

4.2.4.4 Instruction Barrier and Memory Barrier Instructions

The Cortex-M3 supports a number of barrier instructions. These instructions are needed with complex memory systems. In some cases, if memory barrier instructions are not used, race conditions can occur.

There are three barrier instructions in the Cortex-M3:

- DMB (Data Memory Barrier) – Ensures that all memory accesses are completed before new memory access is committed. For example, when you do a data write followed immediately by a read on a dual port memory, if the memory write is buffered, the DMB instruction can be used to ensure the read gets the updated value.
- DSB (Data Synchronization Barrier) – Ensures that all memory accesses are completed before the next instruction is executed
- ISB (Instruction Synchronization Barrier) – Flushes the pipeline and ensures that all previous instructions are completed before executing new instructions

4.2.4.5 Saturation Operations

The Cortex-M3 supports two instructions that provide signed and unsigned saturation operations: SSAT and USAT (for signed data type and unsigned data type, respectively).

Saturation is commonly used in signal processing, for example, in signal amplification.

The saturation operation does not prevent the distortion of the signal, but the amount of distortion is greatly reduced in the signal waveform.

4.2.5 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 timer uses either the Cortex-M3 internal clock or a PSoC 5 clock (see the Clocking System chapter on page 91). There are two registers that control the clock source. The first is the Cortex-M3 register NVIC_SYSTICK_CTL, which selects whether the source is internal or external (default). The second is the PSoC 5 register PANTHER_WAITPIPE, which selects the external clock source: either the ILO 100 kHz (default) or a clock routed from the DSI.

4.2.6 Debug and Trace:

The Cortex-M3 provides a wide range of debugging components. The debug unit is tightly linked with the core.

The important features of the debug and trace are:

- Debug access to all memory and registers in the system including Cortex-M3 register bank when the core is running, halted, or held in reset.
- Serial Wire Debug Port (SW-DP) debug access.
- Flash Patch and Breakpoint (FPB) unit to implement breakpoints and code patches.
- Data Watchpoint and Trace (DWT) unit to implement watchpoints, data tracing, and system profiling.
- Support for six breakpoints and four watchpoints.
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging.

The Cortex-M3 supports separate debug and trace interfaces. The debug interface uses the APB (Access Port Bus), which supports SWD. The trace interface uses the SWV port.

For further details about the debug and trace feature, refer to the Test Controller chapter on page 345.

4.3 Memory Map

The Cortex-M3 has a linear 32-bit (4 GB) address space, as shown in Figure 4-7. See also the Memory Map chapter on page 87.

The address space includes two bit-band alias regions, one for the SRAM space and the other for the Peripherals space. Accesses to a bit-band alias region affect individual bits in the corresponding bit-band region. For example, writing a 1 to address 0x22000000 sets bit 0 of address 0x20000000, and writing a 0 to address 0x42000004 clears bit 1 of address 0x40000000. Reading address 0x22000008 returns a 1 or 0, depending on the value of bit 2 of address 0x20000000.

The processor supports unaligned accesses. Unlike aligned access where the data can be situated only at even addresses, the unaligned accesses support data operations at odd addresses also. Unaligned accesses have limitations. Some instructions cannot support unaligned accesses.

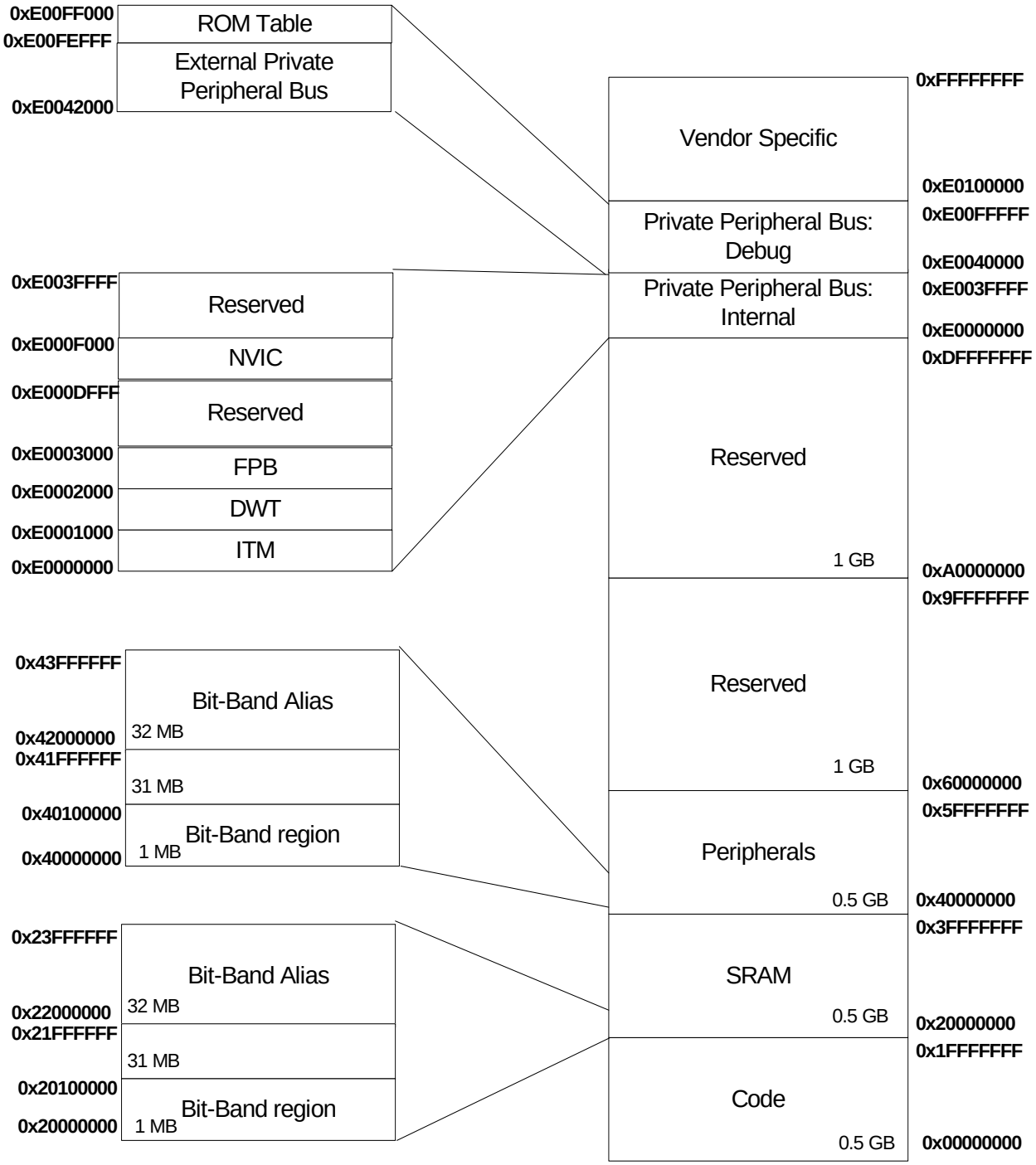
You can execute code from within the code or SRAM.

The Cortex-M3 uses little-endian format.

4.3.1 Bus Interface to SRAM Memory

The 64 KB of SRAM in PSoC 5 is split into two 32 KB of SRAM. The SRAM can be accessed by the C-Bus, S-Bus, and the PHUB's DMA. The priority decoder gives a higher priority to the C-Bus in the upper 32 KB of SRAM, whereas the PHUB DMA takes a higher priority in the lower 32 KB of SRAM. The upper and lower halves of SRAM can be accessed simultaneously but with different buses.

Figure 4-7. Cortex-M3 Memory Map



4.4 Exceptions

The Cortex-M3 provides a feature-packed exception architecture that supports a number of system exceptions and external interrupts. Exceptions are numbered 1 to 15 for system exceptions and 16 and above for external interrupt inputs. PSoC 5 architecture supports 32 external interrupts.

The exceptions are handled by the NVIC. Most of the exceptions have programmable priority, and a few have fixed priority. Table 4-2 shows the list of exceptions available in the Cortex-M3:

Table 4-2. PSoC 5 Exceptions

Interrupt Number	Exception Type	Priority	Comment
1	Reset	–3 (highest) Not programmable	Reset
2	NMI	–2 Not programmable	Non-Maskable Interrupt
3	Hard Fault	–1 Not Programmable	All fault conditions if the corresponding handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; access to illegal locations
5	Bus Fault	Programmable	Bus error occurs when AHB interface receives an error response from a bus slave (also called prefetch abort if it is an instruction fetch or data abort if it is a data access)
6	Usage Fault	Programmable	Exceptions due to program error
7	Reserved	NA	–
8	Reserved	NA	–
9	Reserved	NA	–
10	Reserved	NA	–
11	SVCall	Programmable	System Service Call
12	Debug Monitor	Programmable	Debug monitor (watchpoints, breakpoints, external debug request)
13	Reserved	NA	–
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer

The value of the current running exception is indicated by the special register IPSR or from the NVIC's Interrupt Control State Register (the VECTACTIVE field).

Interrupts are a subset of exceptions. So exceptions are handled the same way as an interrupt. The exception handler for each exception is stored in the interrupt vector table. The vector table begins with the exception handler and is followed by the interrupt service routine addresses. The vector table pointer is dynamically changeable. Also, if the vector table is in SRAM, then vectors can be dynamically changeable.

4.4.1 Priority Definitions

In the Cortex-M3, whether and when an exception can be carried out can be affected by the priority of the exception. A higher priority (smaller number in priority level) exception can preempt a lower priority (larger number in priority level) exception; this is the nested exception/interrupt scenario. From Table 4-2, you can see that some of the exceptions (reset, NMI, and hard fault) have fixed priority levels. They are negative numbers to indicate that they are higher priority than other exceptions. Other exceptions have programmable priority levels.

4.4.2 Fault Exceptions

A number of system exceptions are useful for fault handling. There are several categories of faults:

- Bus faults
- Memory Management Faults
- Usage Faults
- Hard Faults

The faults can be enabled by setting the corresponding bits in the handler control and state register. The reason for a particular fault is updated in the corresponding status register (for example, BFSR register for bus fault, MFSR for memory management fault, UFSR for Usage Fault, HFSR for Hard Fault). These registers can be read to know the exact reason for fault.

When these types of faults (except vector fetches) take place, and if the corresponding exception handler is enabled and no other exceptions with the same or higher priority are running, the fault exception handler is executed. If the

exception handler is enabled but at the same time the core receives another exception handler/interrupt with higher priority, this fault exception handler will be pending and will be executed after the high priority exception/interrupt has completed its execution.

If the fault handler is not enabled or when the fault happens in an exception handler that has the same or higher priority than the current fault handler, the hard fault handler will be executed instead.

Bus Faults

Bus faults are produced when an error response is received during a transfer on the AHB interfaces. It can happen during prefetch, data read/write, or during stacking and unstacking operations.

Memory Management Faults

Memory management faults can be caused by certain illegal accesses, including the following:

- Trying to execute code from non-executable memory regions
- Writing to read-only regions
- Access in the user state to a region defined as privileged access only

Usage Faults

Usage faults can be caused by a number of reasons, including the following:

- Undefined instructions
- Coprocessor instructions (the Cortex-M3 processor does not support a coprocessor, but it is possible to use the fault exception mechanism to run software compiled for other Cortex processors via coprocessor emulation)
- Trying to switch to the ARM state (software can use this faulting mechanism to test whether the processor on which it runs supports ARM code; because the Cortex-M3 does not support the ARM state, a usage fault takes place if there is an attempt to switch)
- Invalid interrupt return (link register contains invalid/incorrect values)
- Unaligned memory accesses using multiple load or store instructions

It is also possible, by setting up certain control bits in the NVIC, to generate usage faults for:

- Divide by zero
- Any unaligned memory accesses

Hard Faults

The hard fault handler can be caused by:

- Usage faults, bus faults, and memory management faults if their handler cannot be executed.
- Bus faults during vector fetch (reading of a vector table during exception handling).

4.4.3 System Call Exceptions

SVC (System Service Call) and PendSV (Pended System Call) are two exceptions targeted at software and operating systems.

SVC

SVC is to generate system function calls. It can be configured to generate an interrupt. This interrupt can be used for task management in a realtime system. SVC is generated using the SVC instruction.

PendSV

PendSV works with SVC in the OS. Although SVC (by SVC instruction) cannot be pended (an application calling SVC will expect the required task to be done immediately), PendSV can be pended and is useful for an OS to pend an exception so that an action can be performed after other important tasks are completed. PendSV is generated by writing ‘1’ to the NVIC PendSV pending register. A typical use of PendSV is context switching.

SysTick Timer Exception

The SysTick Timer exception takes the vector number 15. Cortex-M3 supports a 24-bit down counter. This timer is very useful to perform task management where the software can be handled inside the timer interrupt.

The SYSTICK Timer can be used to generate interrupts. It has a dedicated exception type and exception vector. It makes porting operating systems and software easier because the process is the same across different Cortex-M3 products.

The SYSTICK Timer is controlled by four registers. Of the four registers, TICKINT is used to enable or disable the timer exception.

4.5 Nested Vector Interrupt Controller (NVIC)

The Nested Vectored Interrupt Controller (NVIC), is an integral part of the Cortex-M3 processor. It is closely linked to the Cortex-M3 CPU core logic. Its control registers are accessible as memory-mapped devices. Besides control

registers and control logic for interrupt processing, the NVIC also contains control registers for the SYSTICK Timer, and debugging controls.

Following are the important features of the NVIC:

- Supports 32 interrupts and 16 exceptions.
- Configurable priority levels.
- Dynamic reprioritization of interrupts.
- Support for nested interrupts
- Programmable interrupt vector
- Supports tail-chaining and late arrival interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
- Processor state automatically saved upon interrupt entry, and restored upon interrupt exit, with no instruction overhead.

4.5.1 Basic Interrupt Configuration

Each external interrupt has several associated registers.

- Enable and Clear Enable
- Set Pending and Clear Pending
- Priority Level
- Active Status
- Exception-masking registers (PRIMASK, FAULTMASK, and BASEPRI)
- Vector Table Offset

The interrupt enable and clear enable registers are 32-bit registers. They are used to enable/disable an interrupt. An interrupt that is waiting for the CPU execution sets the pending bit in the set pending register. When the interrupt has been executed by the CPU, the interrupt is cleared automatically by setting the clear-pending register. The interrupts can take priorities 0 to 7. The priorities are configured using the 3-bit priority registers. They can be dynamically configured during run time.

The Active Status register stores the details of the interrupt currently active. A bit set in this register indicates that the corresponding interrupt is currently active. An interrupt is called active if it is currently executed by the CPU or if it is already nested and put to the stack. When the interrupt execution is complete, the active status bit of the interrupt is automatically cleared. With PSoC 5 devices, the addresses of the interrupt service routine are stored in the Interrupt vector table. The interrupt vector table can be located either in RAM or ROM. The position of the vector table is controlled using the Vector Table Offset register.

The exception masking registers, PRIMASK, FAULTMASK and BASEPRI, are special registers used to mask the interrupts and exceptions.

- PRIMASK – When set, all interrupts except NMI and Fault interrupts are masked
- FAULTMASK – When set, all interrupts except NMI are masked
- BASEPRI – Masks all interrupts at the specified priority and lower priorities

4.5.1.1 Example Procedure to Set Up an Interrupt

Here is a simple example procedure to set up an interrupt:

1. Copy the Hard Fault and NMI handlers to a new vector table location if vector table relocation is required. (In simple applications, this might not be needed.)
2. The Vector Table Offset register should also be set up to get the vector table ready (optional).
3. Set up the interrupt vector for the interrupt. Because the vector table could have been relocated, you might need to read the Vector Table Offset register; then calculate the correct memory location for your interrupt handler. This step might not be needed if the vector is hardcoded in ROM.
4. Set up the priority level for the interrupt.
5. Enable the interrupt.

4.5.2 Nested Interrupts

Nested interrupt support is built into the Cortex-M3 processor core and the NVIC. The nesting is done based on the priority of the interrupts. When the processor is handling an exception, all other exceptions with the same or lower priority will be blocked. When a high priority interrupt occurs, the low priority interrupt is nested and the high priority interrupt completes the execution. Nesting is done without risk of losing data in registers because automatic hardware stacking and unstacking is done. Cortex-M3 uses the main stack to store the nesting interrupt details; therefore, take care to ensure sufficient stack space is available.

Reentrant exceptions are not supported in the Cortex-M3.

4.5.3 Tail-Chaining Interrupts

The Cortex-M3 uses a number of methods to improve interrupt latency. Tail-chaining is one such method.

When an exception takes place but the processor is handling another exception of the same or higher priority, the exception will be pending. When the processor has finished executing the current exception handler, instead of POP, the registers go back into the stack and PUSH it back in again,

skipping the unstacking and the stacking. In this way the timing gap between the two exception handlers is greatly reduced.

4.5.4 Late Arrivals

Another feature that improves interrupt performance is late arrival exception handling. When an exception takes place and the processor has started the stacking process, and if during this delay a new exception arrives with higher preemption priority, the late arrival exception will be processed first.

For example, if Exception #1 (lower priority) takes place a few cycles before Exception #2 (higher priority), the processor behaves such that Handler #2 is executed as soon as the stacking completes. After this, Handler #1 is executed.

4.5.5 Interrupt Latency

The term interrupt latency refers to the delay from the start of the interrupt request to the start of interrupt handler execution.

- In the Cortex-M3 processor, if the memory system has zero latency, and provided that the bus system design allows vector fetch and stacking to happen at the same time, the interrupt latency can be as low as 12 cycles. This includes stacking the registers, vector fetch, and fetching instructions for the interrupt handler. However, this depends on memory access wait states and a few other factors.
- For tail-chaining interrupts, because there is no need to carry out stacking operations, the latency of switching from one exception handler to another exception handler can be as low as 6 cycles.
- When the processor is executing a multi-cycle instruction such as divide, load double, or store double, the instruction could be abandoned and restarted after the interrupt handler completes.
- To reduce exception latency, the Cortex-M3 processor allows exceptions in the middle of multiple load and store instructions (LDM/STM). If the LDM/STM instruction is executing, the current memory accesses are completed, and the next register number is saved in the stacked xPSR (ICI bits). After the exception handler completes, the multiple load/store resumes from the point at which the transfer stopped.

4.5.6 Faults Related to Interrupts

Faults (bus fault, memory fault) can happen during the following stages of interrupt execution:

- Stacking
- Unstacking
- Vector Fetches

5. PSoC 5 Cache Controller

The cache block is an Instruction cache only. It services instruction fetches from the CPU. It stores lines of code from the flash in its internal buffer for fast accesses made by the CPU at a later time.

5.1 Features

- Instruction cache
- Direct mapped
- 128 bytes total cache memory
- Registers to measure cache hit/miss ratios

5.2 Block Diagram

Figure 5-1 shows the system interaction with the cache block as well as the cache interfaces and data/instruction flow.

Figure 5-1. Cache Interfaces

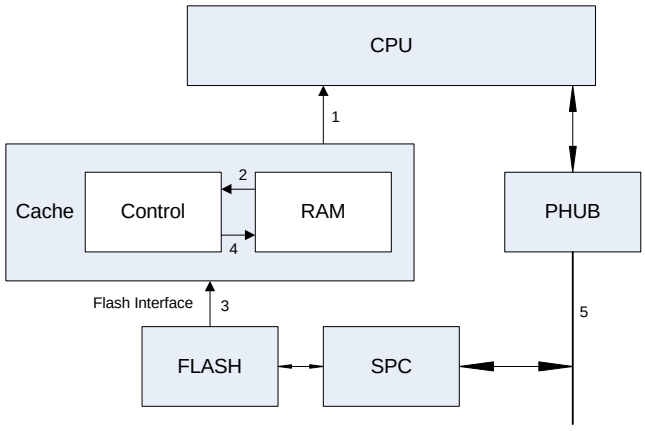


Table 5-1. Cache Operational Interfaces

Interface	Function
1	CPU sends instruction fetch request through this interface to the cache and eventually receives back the instruction
2	When the CPU instruction fetch gets a hit in the cache, it is retrieved from the cache memory (RAM) through this interface.
3	CPU instruction fetch (interface #1) that gets a miss in the cache is translated into one fetch request from the Flash. The Flash access time is much larger than the cache RAM access time, up to four CPU clock cycles.
4	Instructions returned from the Flash are cached through this interface for later CPU use.
5	Flash memory locations are accessed by the CPU using this interface.

5.3 Cache Enable and Disable

By default, the cache is enabled; to disable it, set the DISABLE bit (Bit 0) of CACHE.CC_CTL register.

5.4 Invalidate Cache Line

Software can invalidate all cached data associated with an interface by setting the Flush bit (Bit 2) of CACHE.CC_CTL register. Invalidate takes effect in one cycle and affects all lines.

5.4.1 Measuring Cache Hits or Misses

The CACHE.HITMISS register provides two 16-bit counters that count the number of cache hits and misses. To measure the cache performance, reset the HITMISS register to ‘0’ at the start of the block of code that is being measured. The the code is executed; at the end of the code under measurement, the HITMISS register is read. The cache hit ratio is computed as follows:

Cache hit ratio = the number of cache hits (HITMISS [31:16])/Number of cache misses (HITMISS[15:0])

5.5 Wait States when Reading from Flash

The wait states refer to the number of bus clocks for which cache must wait before data is available from flash. It varies with the bus clock frequency. The number of wait state clocks can be configured in FLASH_CYCLES field of CACHE.CC_CTL register. The following table gives the recommended values of wait states for different bus clock frequency range.

Table 5-2. Wait States

FLASH_CYCLES in CACHE.CC_CTL Register	Bus Clock Frequency	Wait States on Cache Miss
0x1	1 Hz to 16 MHz	1
0x2	>16 MHz to 33 MHz	2
0x3	>33 MHz to 50 MHz	3
0x0	>50 MHz to 67 MHz	4

5.6 Sleep Mode Behavior

When the device wakes up from low power modes, all cache data and tags are invalidated. However, all the cache registers (where cache settings are made) maintain their state and are not reset. The cache will be refilled as the CPU begins fetching instructions.

On system reset, cache is invalidated and begins to fill with the first request from the CPU.

5.7 Cache Limitations

Cache coherency is the software's responsibility; no hardware mechanism exists to ensure coherency. If the software modifies the Flash or memory contents, it also needs to invalidate the cache and ensure the new instruction is fetched into the cache.

6. PHUB and DMAC

PSoC® 5 devices use a high-performance bus for peripheral access and bulk data transfer. The high-performance bus and the associated central controller are known as the peripheral hub (PHUB). The PHUB is a programmable and configurable central bus backbone within a PSoC 5 device that ties the various on-chip system elements together. It consists of multiple spokes; each spoke is connected to one or more peripheral blocks. The PHUB also includes a direct memory access controller (DMAC), which is used for data transfer. The DMAC supports multiple DMA channels.

There are two bus masters (blocks that can initiate bus traffic) in PSoC 5 devices. These are the DMAC and the CPU. An arbiter in the PHUB is responsible for arbitrating requests from the CPU and the DMAC. Upon receiving a request from the microcontroller or the DMAC, the PHUB relays the request to the appropriate peripheral spoke.

6.1 PHUB

PHUB manages arbitration between the CPU and the DMAC.

6.1.1 Features

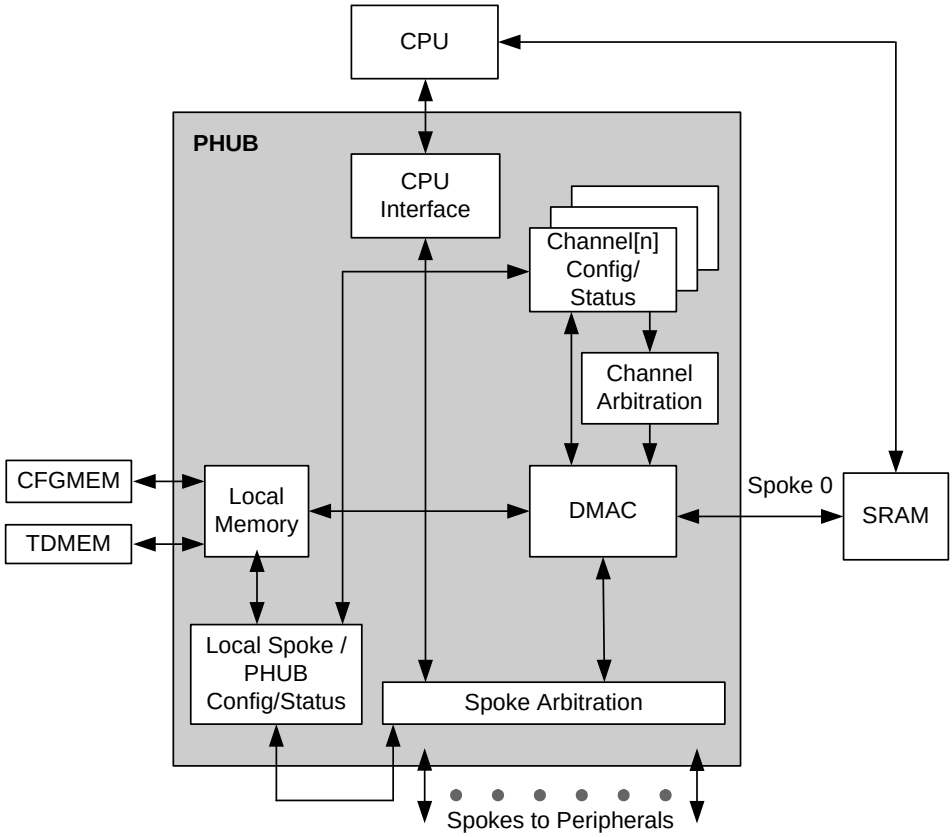
The PHUB has the following features:

- Industry-standard Advanced Microcontroller Bus Architecture High-performance Bus (AMBA -HB) lite protocol
- 8 spokes connected to various peripherals
- 8-/16-/32-bit data-width support
- Peripherals of various address widths connected to the same spoke
- Includes programmable DMAC with 24 direct memory access (DMA) channels
- Byte order and data width difference translation

6.1.2 Block Diagram

Figure 6-1 on page 52 is the block diagram of the PHUB. The DMAC is also shown.

Figure 6-1. PHUB Block Diagram



6.1.3 How It Works

The PHUB is used to connect the CPU to memory and peripherals, including SRAM, flash, EEPROM, analog subsystem, digital blocks, digital filter block, and others.

The PHUB connects to the peripherals using a spoke. There are eight spokes. Each spoke connects to one or more peripherals. Each spoke is configured for:

- **Address width** – The address width of a spoke depends on the maximum number of addresses required for the peripherals connected to the spoke.
- **Data width** – The data width of a spoke can be 16 or 32 bits. Eight-bit data transfer can be performed on 16- and 32-bit spokes.
- **Number of peripherals** – This depends on the device architecture. Each spoke is usually connected to multiple peripherals.

Table 6-1 shows the address width, data width, and peripherals connected to each spoke in the PSoC 5 device.

Table 6-1. Spoke Configuration

Spoke	Address Width (in bits)	Data Width (in bits)	Peripheral Names
0	14	32	SYSMEM(SRAM)
1	9	16	IO interface, port interrupt control unit (PICU)
2	19	32	PHUB local spoke, power management, clock, serial wire viewer (SWV), EEPROM cache
3	11	16	Delta-sigma ADC, analog interface
4	10	16	USB, fixed-function I^2C, fixed-function timers
5	11	32	Digital filter block (DFB)
6	17	16	UDB set 0 registers (including DSI, configuration, and control registers), UDB interface
7	17	16	UDB set 1 registers (including DSI, configuration, and control registers)

- The peripherals connected to each spoke can have data widths longer than the spoke. For example, a Delta-Sigma ADC can support up to 20-bit data although it is placed in the 16-bit spoke (spoke 03).

  In this case, the PHUB uses an internal FIFO to accommodate the width differences during data transfer.

- One peripheral can extend across multiple spokes. In this case, the peripheral will have different address spaces that are connected to each spoke.

  For example, Table 6-1 shows that UDB registers extend across two spokes. UDB registers can be accessed in 8-bit mode and also in 16-bit mode. In this case, the 8-bit mode access needs a different address space than the 16-bit mode access though they reside in the same spoke.
- Peripherals of different data widths can be connected to a single spoke.

  An example of this is spoke 3, which is connected to the analog interface (digital-to-analog converter) and delta-sigma ADC. The delta-sigma ADC can support up to 20-bit data, and the digital-to-analog converter register is 8-bit.
- Spoke 0 is connected to SRAM. The CPU can access the SRAM without going through the PHUB. The DMAC accesses the SRAM through PHUB. For DMA access, the SRAM region 0x1FFF_8000 to 0x1FFF_FFFF is remapped to 0x2000_8000 so that the entire SRAM contiguous within a 64 KB boundary. See Table 14-2 on page 88 for more details.

The spoke address width, data width, and peripherals are fixed in a device and cannot be changed. The spoke and the peripheral details affect the time required for data transfer. interspoke and intraspoke transfers take different amounts of time.

The effects of spoke data width, and interspoke and intraspoke transfer, on latency of data transfer are explained in 6.1.4 Arbiter.

6.1.4 Arbiter

The PHUB receives data read or write requests from either the CPU or the DMAC. The PHUB processes each request to determine which spoke and peripheral should be accessed, and then manages the data access.

When the DMAC and CPU initiate transactions in the PHUB at the same time, the arbiter decides which request has priority. The priority can be configured for every spoke except spoke 0. Spoke 0 is accessed only by the DMAC because the CPU has a separate interface to SRAM. You can configure priority using the “spk_cpu_pri” bits in the PHUB_CFG register.

When the CPU and DMAC access different spokes simultaneously, both accesses are independent and arbitration is not necessary. This enables a multiprocessing environment. The arbitration issues when the CPU and DMA want to access the same spoke simultaneously are detailed in further sections.

6.2 DMA Controller

The DMA Controller (DMAC) transfers data between memory and peripherals.

- Uses the PHUB for data transfer
- Includes 24 DMA channels
- Includes 128 transaction descriptors (TD)
- Eight levels of priority per channel
- Transactions can be triggered by any digitally routable signal, the CPU, or another DMA channel
- Transactions can be stalled or canceled
- Each transaction can be from 1 to 64 KB
- Entire transaction can be broken into 1 to 127 byte bursts
- Each channel can be configured to generate a transaction complete (termout) signal at the end of the transfer
- Supports 2 byte or 4 byte swapping, for conversion between big-endian and little-endian formats
- Handles data-width differences between spokes during data transfer

Note The TD 0xFE is reserved; do not use the same.

6.2.1 Local Memory

As shown in Figure 6-1 on page 52, the PHUB includes local memory to store configuration data. The local memories are called

- Configuration memory (CFGMEM)
- Transaction descriptor memory (TDMEM)

The PHUB also includes a 16-byte FIFO for data handling during data transfers.

The CFGMEM is used to store the DMA channel configuration data. There are two registers for each channel: CFGMEMn.CFG0 and CFGMEMn.CFG1 (n is the channel number and can be in the range 0 to 23). Each register is 32 bits, so the size of CFGMEM is 8 bytes × 24 channels = 192 bytes.

The TDMEM is used to store the TD configuration data, which includes the number of bytes to transfer, source address, destination address, next TD, and other configuration data. Each TD has two registers: TDMEMn.ORIG_TD0 and TDMEMn.ORIG_TD1. Each register is 32 bits, so the size of TDMEM is 8 bytes × 128 TDs = 1 KB of memory.

If a channel requires a separate working area for processing

TDs, then some portion of TDMEM becomes reserved for DMAC's private use (CHn_SEP_TD0/1). In that case, instead of processing the original TDs, DMAC will copy the

original TDs to this separate working area and process them. There can be one CHn_SEP_TD0/1 per channel.

The local memory is accessed through the local spoke of the PHUB (see Table 6-1 on page 52).

6.2.2 How the DMAC Works

The DMAC is one of the bus masters for PHUB. The DMAC can perform the following data transfers:

- Memory to memory
- Memory to peripheral
- Peripheral to memory
- Peripheral to peripheral

Any DMA channel goes through the following phases to perform data transfers:

- Arbitration phase
- Fetch phase
- Source engine phase
- Destination engine phase
- Write back phase

The total time required for a DMA transfer depends on the time taken for each phase. The DMA transfer can be either an intraspoke DMA transfer or interspoke DMA transfer

In an intraspoke transfer, the data transfer happens within the same spoke. This transfer makes use of the internal FIFO.

- Arbitration phase

  The DMAC selects which DMA channel to process based on the priority.
- Fetch phase

  The DMAC fetches the TD and DMA channel details from the configuration registers.
- Source engine phase

  The source engine selects the spoke to which the source peripheral is connected. When the spoke is available for data transfer, the data transfer from the source begins.
- Destination engine phase

  This phase selects the spoke on which the destination peripheral is available. When the spoke is available, the data collected in the source engine phase is transferred to the destination peripheral.
- Write back phase

  This phase is the completion phase were the TD and DMA channel configurations are updated after data transfer.

Ideal conditions for data transfer are:

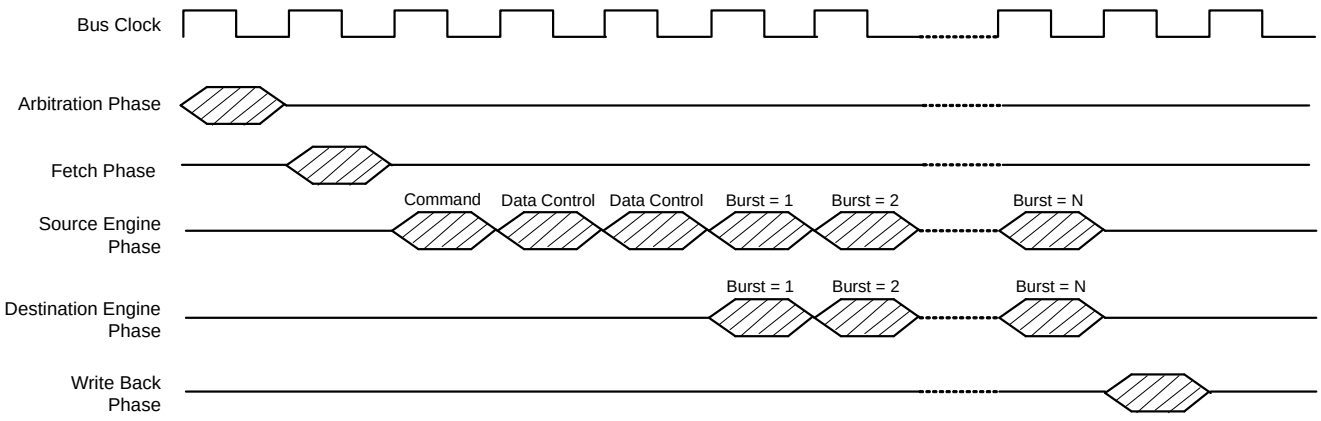
- Single requestor
- CPU does not interrupt the fetch phase
- Both source and destination spoke are readily available
- Source spoke and destination spoke are of same width
- Source and destination address start at even addressing
- Transfer count is a multiple of burst count
- Burst count matches the spoke width

The number of bursts for transfer (N) =
Transfer count ÷ Spoke width

6.2.2.1 Interspoke Transfers

The timing diagram for an interspoke transfer under ideal conditions is shown in Figure 6-2.

Figure 6-2. Interspoke Transfer Cycle Timing



The total number of cycles for data transfer in the case of interspoke DMA transfers is the sum of cycles required for each phase.

Total cycle time = Arbitration phase time (1) + Fetch phase (1) + Source Engine phase (N + 3) + Destination engine phase (0, because it happens in parallel with the source engine phase) + Write back phase (1)

Total cycle time = N + 6 cycles (where N = Transfer count ÷ Spoke width)

Example

Move five samples of 16-bit ADC data to memory.

Note The ADC (decimator) is connected to spoke 3, which is a 16-bit spoke; memory is in spoke 0, which is a 32-bit spoke.

The DMA configuration includes:

- DMA channel burst count (configured in CFGMEMn.CFG0) = 2
- TD transfer count (configured in TDMEMn.ORIG_TD0) = 2 bytes × 5 samples = 10
- TD configuration includes an Increment Destination Address to copy data to an array in the memory (configured in TDMEMn.ORIG_TD0)
- N = Transfer count ÷ Spoke width = 10 ÷ 2 = 5

For more information about the DMA configuration, refer to the PHUB registers in *PSoC 5 Registers TRM*.

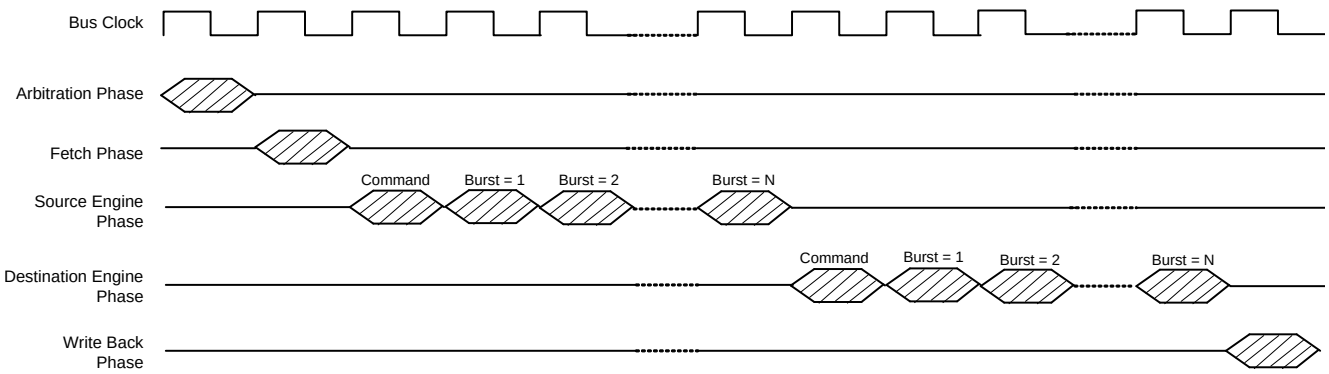
The source engine phase needs N + 3 cycles = 8 cycles.

Total cycle time required for interspoke transfer is N + 6 = 5 + 6 = 11 cycles.

6.2.2.2 Intraspoke Transfer

The timing diagram for intraspoke transfer under ideal conditions is shown in Figure 6-3.

Figure 6-3. Intraspoke Transfer Cycle Timing



The total number of cycles for data transfer in the case of intraspoke DMA transfer is the sum of the cycles required for each phase.

Total cycle time = Arbitration phase time (1) + Fetch phase (1) + Source engine phase (N + 1) + Destination engine phase (N + 1) + Write back phase (1)

Total cycle time = 2N + 5 cycles (where N = Transfer count ÷ Spoke width)

In intraspoke DMA transfers, because the source and destination reside in the same spoke, the 16-byte internal FIFO of the PHUB is used as an intermediate buffer. When the FIFO is full, the PHUB waits for the FIFO to be emptied and the destination engine to read the data, and then fills the next set of data. This is the reason why the destination engine phase cannot happen in parallel with the source engine phase.

Example

Move four 32-bit data words from one SRAM location to another SRAM location.

Note SRAM lies in spoke 0, which is a 32-bit spoke. In this case, both source and destination is SRAM.

The DMA configuration includes:

- Burst count (configured in CFGMEMn.CFG0) = 4
- Transfer count (configured in TDMEMn.ORIG_TD0) = 4 bytes × 4 words = 16
- TD configuration includes increment source address and increment destination address to copy data from one array to another (configured in TDMEMn.ORIG_TD0)
- N = Transfer count ÷ Spoke width = 16 ÷ 4 = 4

The source and destination engine phase needs
2N + 2 cycles = (2 × 4) + 2 cycles = 10 cycles

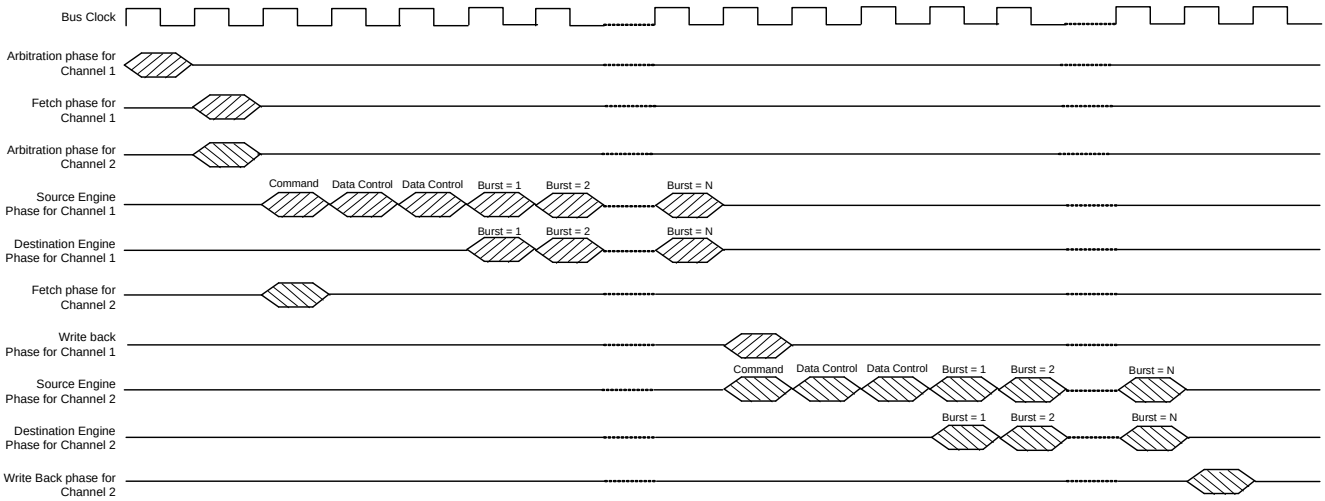
Total cycle time required for intraspoke transfer is 2N + 5 = (2 × 4 + 5) = 13 cycles

6.2.2.3 Handling Multiple DMA Channels

The DMAC can perform phases in parallel. This helps to reduce the latency for executing data transfer. When multiple channels need to execute, the channels can be pipelined.

Figure 6-4 shows processing of two DMA channels that were requested at the same time. The figure shows only the inter-spoke transfer. The same is applicable also for intraspoke transfer.

Figure 6-4. Multiple DMA Channel Processing



6.2.2.4 DMA Channel Priority

Each channel can take a priority from 0 to 7 with 0 being the highest priority.

The DMAC supports two methods to handle the priority: simple priority and grant allocation fairness algorithm.

The priority handling method can be changed by writing to register PHUB.CFG bit “simple_pri” (bit 23).

- Simple Priority: This method handles the channels similar to any normal priority algorithm where high priority channel can interrupt low priority channel.
- Grant allocation Fairness algorithm: In this method, channel 0 and 1 take highest priority and no other priority can interrupt the channels with priority 0 and 1. A DMA channel of priority 0 and priority 1 occupy the bus 100%. The remaining priorities share the bus based on the number of channels requested at that time. Because 0 has higher priority than 1, priority 0 can interrupt priority 1.

In both cases, a DMA channel of low priority can be interrupted by a high priority channel only during the source engine phase.

The arbitration phase time depends on the number of channels requesting the DMAC (non-ideal conditions). When there is only one channel requesting an idle DMAC, the arbitration phase takes one cycle.

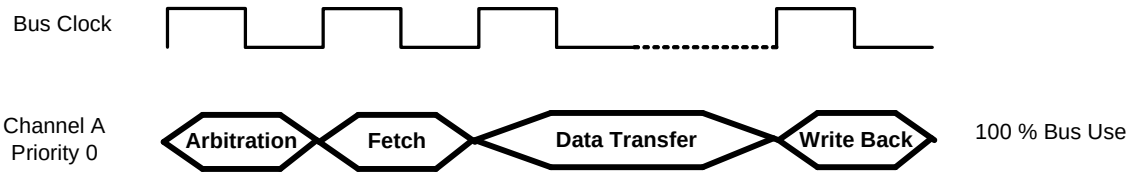
When there is more than one channel requesting a free DMAC, the arbitration phase takes 2 cycles.

Examples using the Grant allocation Fairness Algorithm

Scenario 1

DMAC is free. Channel A with Priority 0 comes

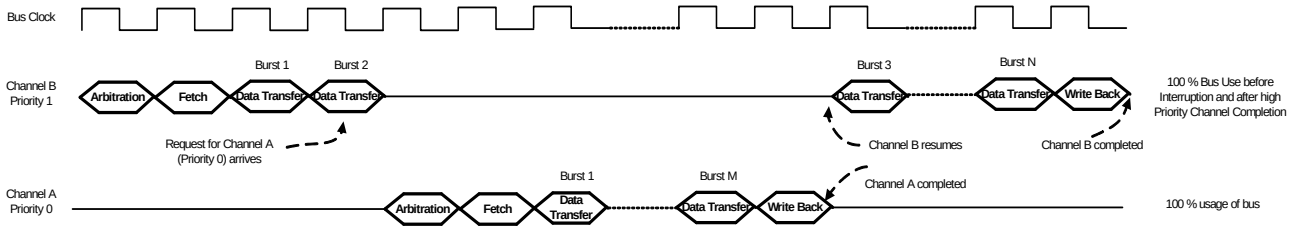
Figure 6-5. Priority 0 and Idle DMAC



Scenario 2

DMAC is free. Channel B with Priority 1 is executing. Channel A with Priority 0 comes

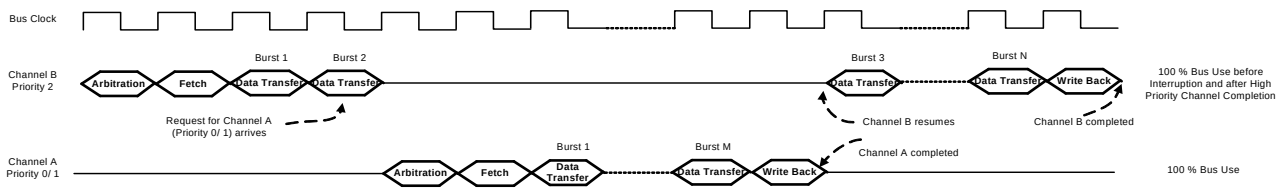
Figure 6-6. Priority 0 and Priority 1



Scenario 3

DMAC is free. Channel B with Priority 2 is executing. Channel A with Priority 0/1 comes

Figure 6-7. Priority 0/1 and Other Low Priority



Scenario 4

DMAC is free. Channel B with Priority 3 is executing. Channel A with Priority 2 comes

Figure 6-8. Lower Priority Channels with Grant Allocation

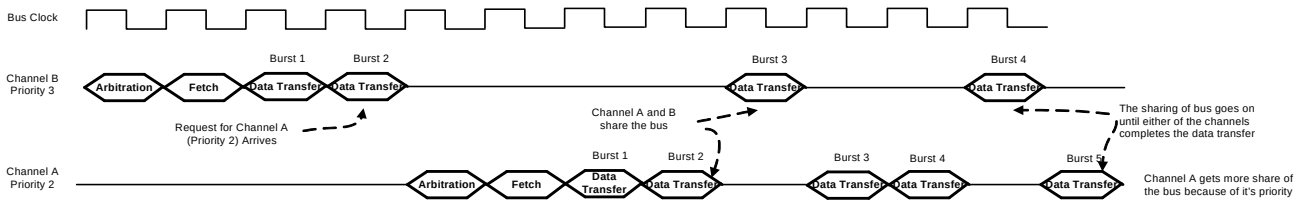


Table 6-2 shows the minimum guarantee for a DMA channel priority to get bus access

Table 6-2. Priority Levels and Bus Allocation

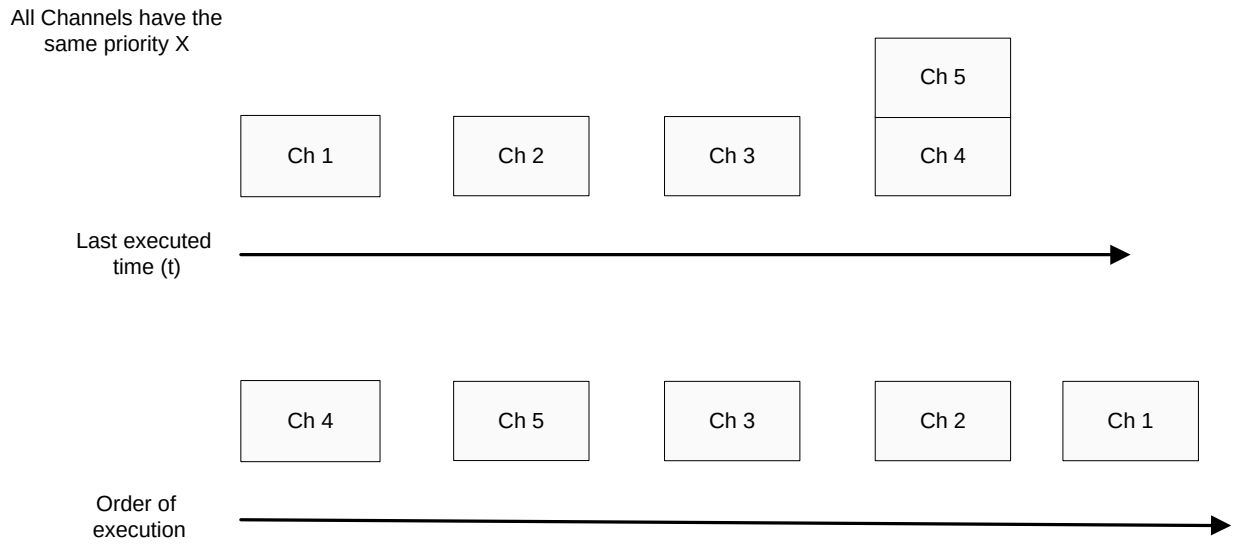
Priority Level	Bus Allocation Percentage
0	100
1	100
2	50
3	25
4	12.5
5	6.3
6	3.1
7	1.5

Because there are as many as 24 DMA channels but only 8 priority levels, there can be multiple channels taking the same priority levels.

DMAC uses the Round Robin method to handle DMA channels with same priority. In the round robin algorithm, the DMA channel that was not executed recently takes a higher priority. The execution of same priority DMA channels when round robin algorithm is enabled depends on:

- The last time when the channel was enabled
- If the last time is the same for two channels, then DMA channel with lower number takes higher priority

Figure 6-9. Round Robin Scheduling



6.2.2.5 *DMA Latency in case of Nonideal Conditions*

The previous section explained the latency in an ideal condition. But in real time, the ideal condition rarely exists. This section explains the latency calculation in non-ideal conditions. The latency calculation in nonideal conditions cannot be explained using formula as against the ideal condition.

Multiple Requestors

In real time system, the PHUB will be requested by multiple channels and also by the CPU.

If there are multiple DMA channels sending requests simultaneously, the arbitration phase will take two cycles instead of the ideal one cycle.

CPU Interrupts with Fetch Phase

The fetch phase ideally takes only one cycle for the PHUB to access the configuration registers through the PHUB local

spoke. When CPU interrupts the fetch phase, the latency depends on when the CPU releases the configuration registers. Typically CPU takes two cycles for the access of configuration registers.

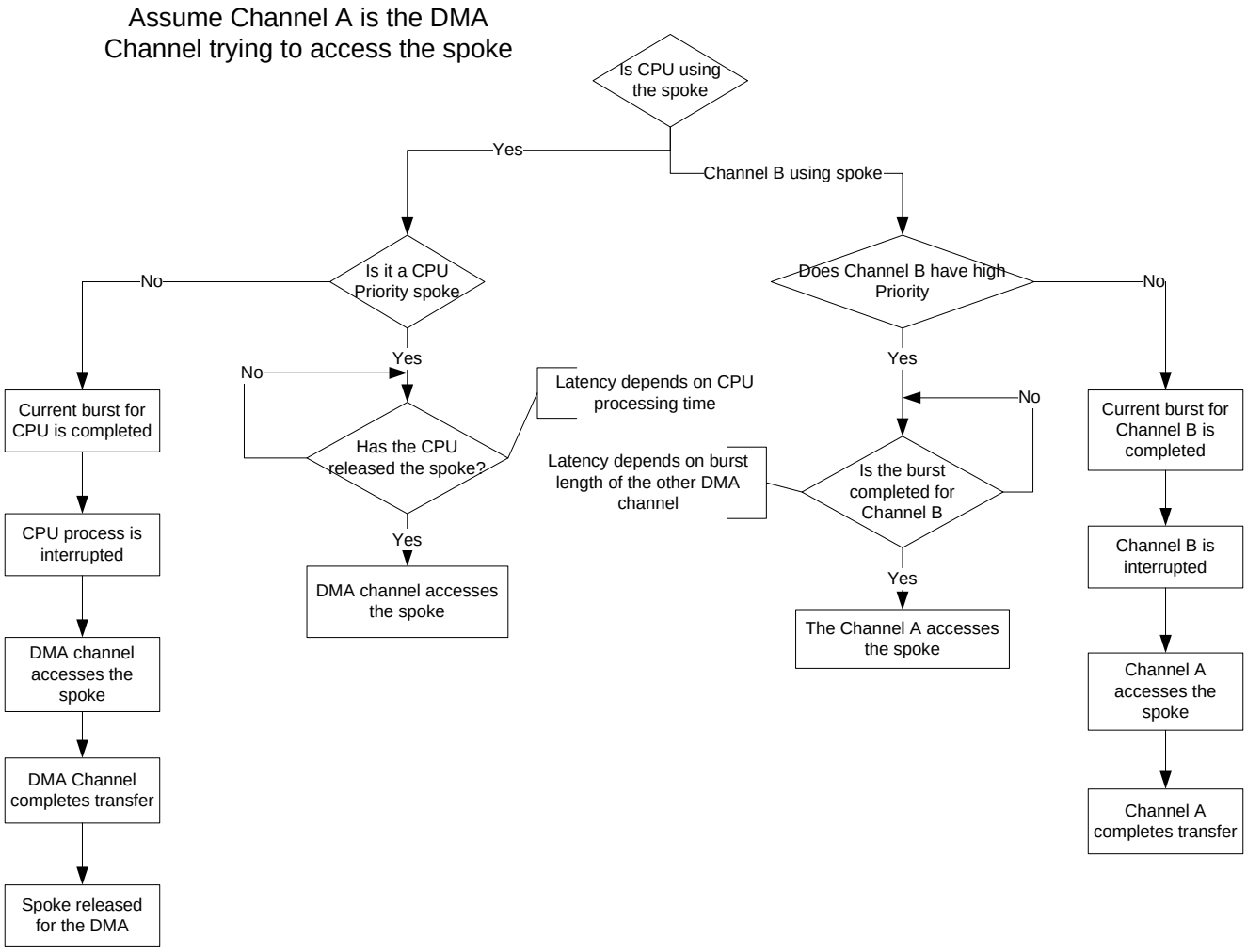
Also, there might be some high priority DMA channel in the Fetch phase. These scenarios will also add to the DMA Channel execution latency.

Source and Destination Spokes in Use

The source and destination for a particular DMA channel should be free for the channel to use it. In real time, a source or destination spoke may be already used by CPU or another DMA channel.

When source and destination spoke is already in use, the PHUB does the arbitration. The following flowchart shows the arbitration mechanism.

Figure 6-10. DMA Channel Arbitration



This latency is not measurable and depends on the real time situation where the same spoke can be accessed by multiple resources.

Source and destination peripherals are not ready

When the source or destination peripheral is not ready to send or receive data, then the DMA channel must wait until it is ready. If the source peripheral is not ready, the DMA channel will wait for the source peripheral to become ready. If the destination peripheral is not ready, the DMA channel will use the 16 byte FIFO of the PHUB. It reads the data from the source and enters it in the FIFO until the destination peripheral is ready. Thus the internal 16 byte FIFO is used during intra-spoke transfer and when the source and destination peripherals are not ready.

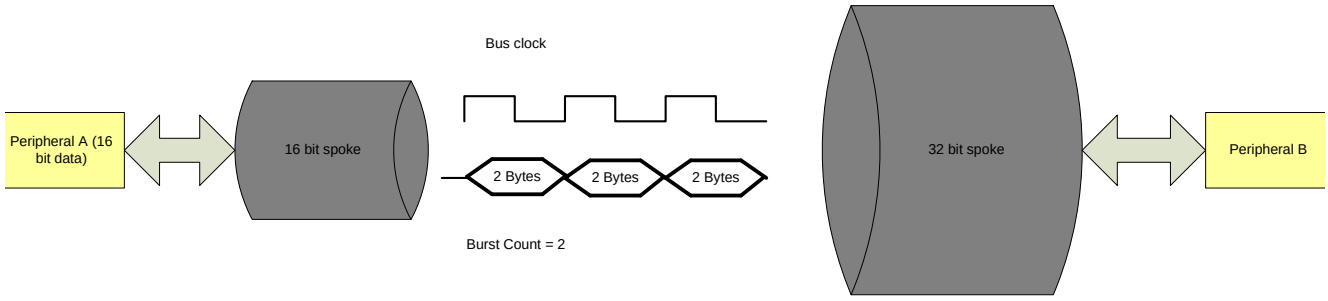
Source and destination spoke are of different width

The spoke widths play an important role in latency. There are chances that the source spoke might be smaller than the destination spoke and vice versa. In this case, the burst count also plays an important role. Here are some examples for this condition:

Scenario 1 (Inter spoke: 16 bit spoke to 32 bit spoke; burst of 2)

- Source: 16 bit spoke (ADC)
- Destination: 32 bit spoke (DFB)
- Burst count: 2 (for 16 bit ADC data)

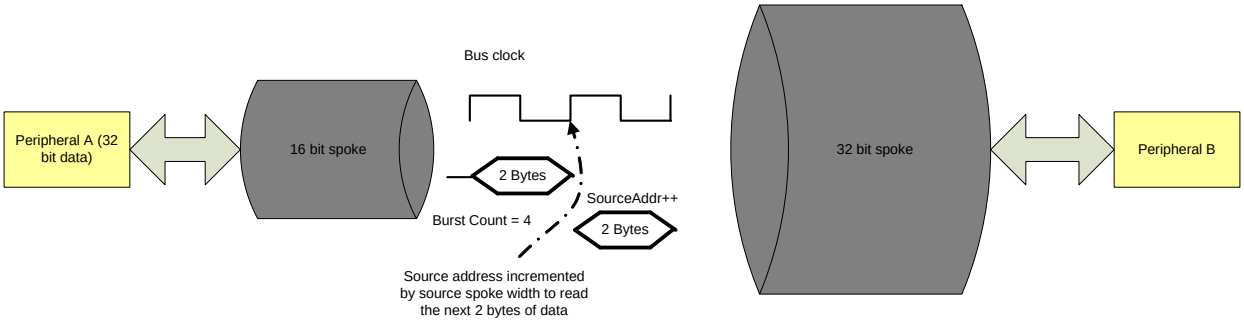
Figure 6-11. Data Transfer Between 16-Bit and 32-Bit Spoke



Scenario 2 (Inter spoke: 16 bit spoke to 32 bit spoke; burst of 4)

- Source: 16 bit spoke (ADC)
- Destination: 32 bit spoke (DFB)
- Burst count: 4 (for 20 bit ADC data)

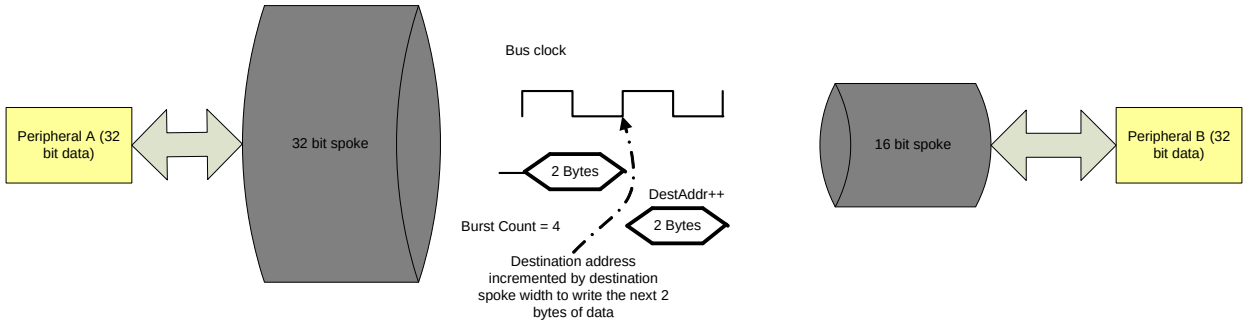
Figure 6-12. Data Transfer Between 16-Bit and 32-Bit Spoke



Scenario 3 (Inter spoke: 32 bit spoke to 16 bit spoke; burst of 4)

- Source: 32 bit spoke (Memory)
- Destination: 16 bit spoke (UDB peripheral)
- Burst count: 4

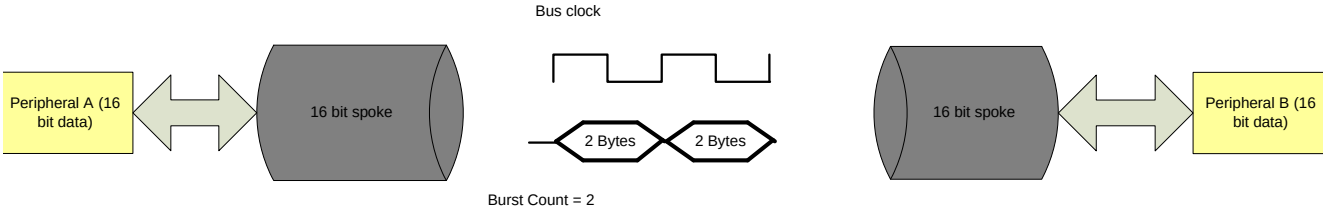
Figure 6-13. Data Transfer Between 16-Bit and 32-Bit Spoke



Scenario 4 (Inter spoke: 16 bit spoke to 16 bit spoke; burst of 2)

- Source: 16 bit spoke
- Destination: 16 bit spoke
- Burst count: 2

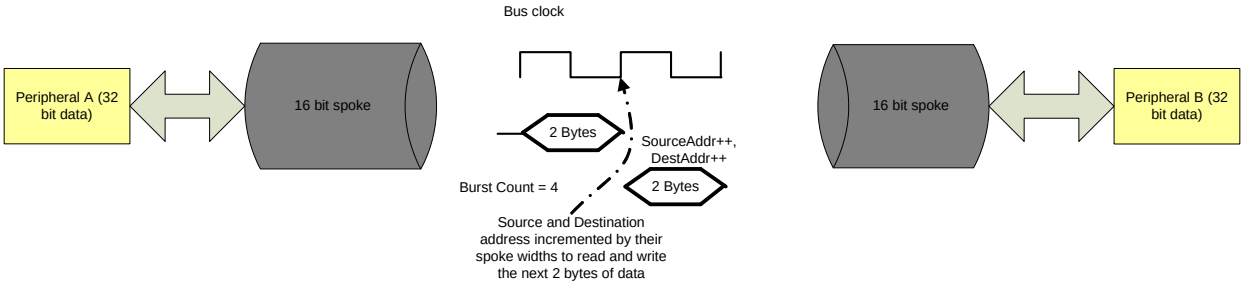
Figure 6-14. Data Transfer Between Two 16-Bit Spoke



Scenario 5 (Inter spoke: 16 bit spoke to 16 bit spoke; burst of 4)

- Source: 16 bit spoke
- Destination: 16 bit spoke
- Burst count: 4

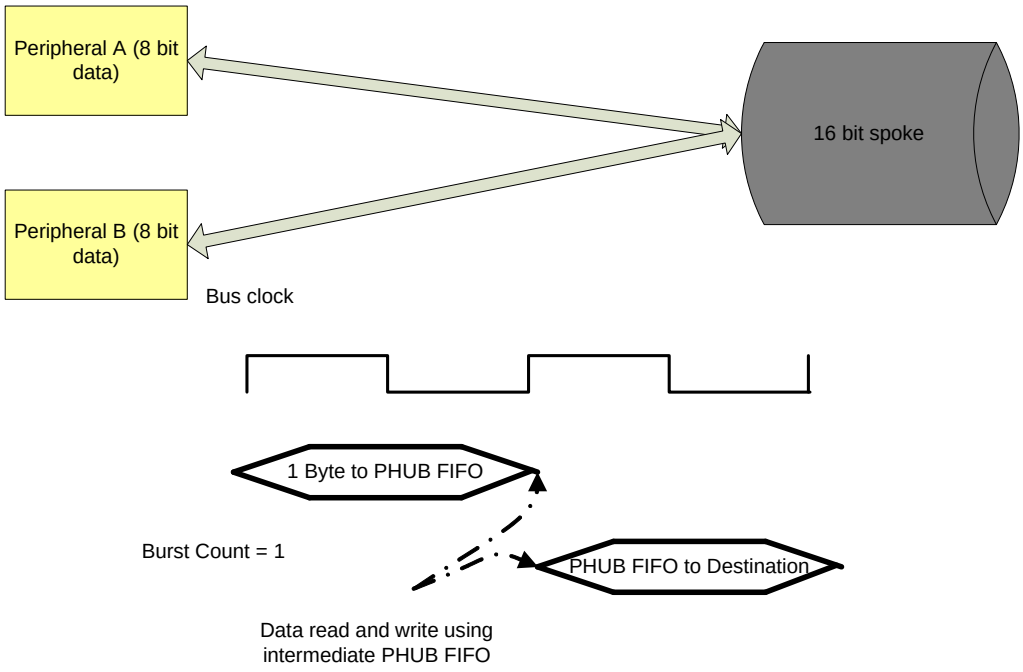
Figure 6-15. Data Transfer Between Two 16-Bit Spoke



Scenario 6 (Intra spoke: 16 bit spoke; burst of 1)

- Source and destination: Same spoke (16 bit)
- Burst count: 1

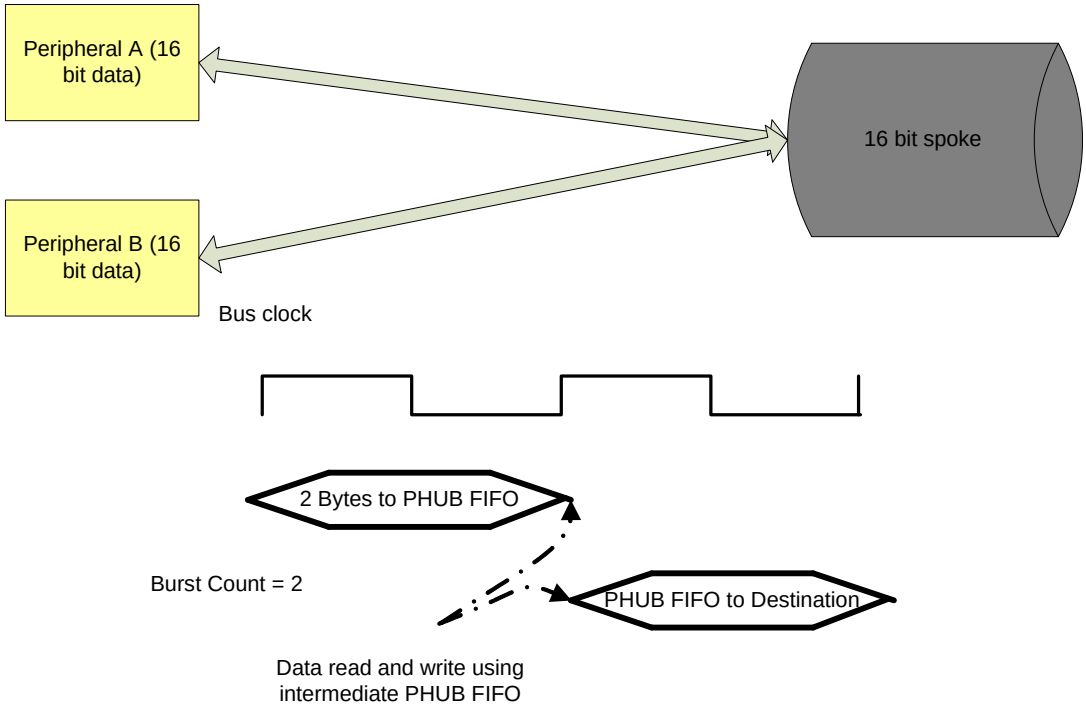
Figure 6-16. Intra Spoke Data Transfer



Scenario 6 (Intra spoke: 16 bit spoke; burst of 2)

- Source and destination: Same spoke (16 bit)
- Burst count: 2

Figure 6-17. Intra Spoke Data Transfer



Source and destination address do not have even addressing

The address of the source and destination play a very important role in deciding the latency. The AHB protocol supports reading from even addresses. Use this notation for a 32 bit spoke.

Figure 6-18. Addressing in 32 bit Spoke

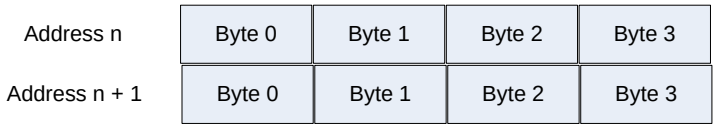
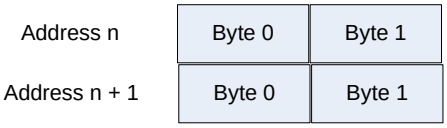
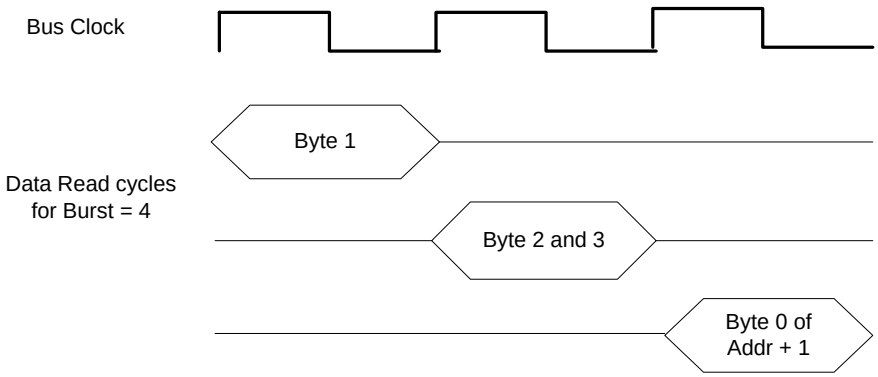


Figure 6-19. Addressing in 16 bit Spoke



Scenario 1: 32 bit spoke, Burst count of 4, Address begins at Byte 1

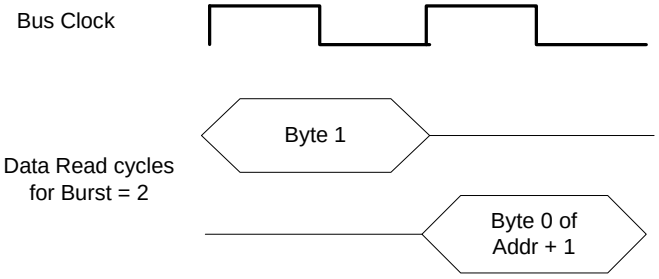
Figure 6-20. Odd Addressing in 32-Bit Spoke



As seen from Figure 6-20, when the even addressing is not met, the bus cycle increases. In an ideal condition where the address begins at Byte 0, a single cycle is sufficient to read all the 4 bytes.

Scenario 2: 16 bit spoke, Burst count of 2, Address begins at Byte 1

Figure 6-21. Odd Addressing In 16 bit Spoke



6.2.2.6 Request per Burst Bit

The data to be transferred can be split into multiple burst - each of same size. This feature is useful under the following situations:

- When the user does not want to control the bus with a single channel which has huge data to transfer
- When the user needs to control the transfer times

The “Request per bit” is bit 7 in CFGMEMn.CFG0 register. This bit is available for individual channel. When this bit is set, the DMA needs a request to transfer the next burst of data. When this bit is set, the DMA channel should go through the whole process from Arbitration phase until Write back phase for every burst. Thus the “Request per bit” parameter will significantly increase the transfer time

6.2.2.7 Work Sep Bit

The “work_sep” bit is bit 5 of the CHn.BASIC_CFG register. This bit is available for individual channel. When this bit is cleared, a TD mapped to that particular DMA channel cannot restore its initial configuration after the data transfer. The TD will retain its last source address, destination address, and transfer count details at the end of transfer.

When this bit is set, a TD mapped to that particular DMA channel restores its initial configuration after the data transfer. This is useful when the TD should be repeated. When the “work_sep” bit is set, DMA uses a separate processing area to store the TD configuration details.

6.3 Low-Power Mode Behavior

Before entering a low power mode, it is recommended to ensure that the DMA is in idle state. PHUB configuration and TDMEM is retained during low power modes. Upon wakeup, DMA activity can be resumed and it will pick up where it left off.

6.4 DMAC Access

6.4.1 SRAM Access

In PSoC 5, the DMAC cannot access SRAM from 0x1FFF8000 to 0x1FFFFFFF, but it can access the same memory at 0x20008000 to 0x2000FFFF.

The CPU accesses:

0x1FFF8000 - 0x1FFFFFFF C-BUS 32KB

0x20000000 - 0x20007FFF S-BUS 32KB

The DMAC accesses:

0x20000000 - 0x20007FFF S-BUS 32KB

0x20008000 - 0x2000FFFF C-BUS 32KB

Note that when the DMA engine increments an address, it only increments the lower 16 bits. Therefore, the next address following 0x2000FFFF will be 0x20000000, which results in the memory space still functioning as a contiguous block of memory.

6.4.2 SIO Port Access

All access to SIO port registers must be done in byte mode either using DMA or CPU. Therefore, the BURSTCNT (see PHUB.CFGMEM[0..23].CFG0 register) for DMA channel must be configured as 1 while accessing SIO port data registers.

6.4.3 Port Configuration register Access

DMA cannot be used to write to pin configuration registers. Always use CPU for writing to pin configuration registers.

6.4.4 USB End Point Access

DMA cannot be used to read/write data to/from one USB endpoint while CPU is accessing a different endpoint or the same endpoint. The only exception is endpoint 0, where DMA can access other endpoints while CPU is accessing endpoint 0 and vice versa.

6.5 DMA Transaction Modes

The DMA channels can be chained to perform complex operation. Similarly, TDs can be nested or chained to perform complex operations. Chaining of TDs is done using the bit “next_td_ptr” in TDMEMn.ORID_TD0 register. This flexibility of the DMA channel and TD helps to create both simple and complex cases

General use cases might include the following types

6.5.1 Simple DMA

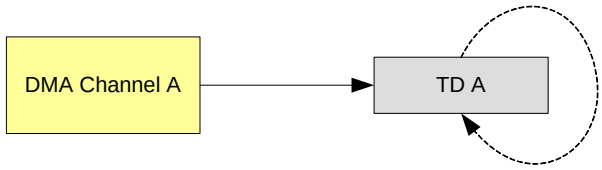
A single TD is used to transfer data between two peripherals or memory locations.

Figure 6-22. Simple DMA Transfer

6.5.2 Auto Repeat DMA

A static pattern is repetitively read from system memory and written to a peripheral. This is done with a single TD that chain to itself.

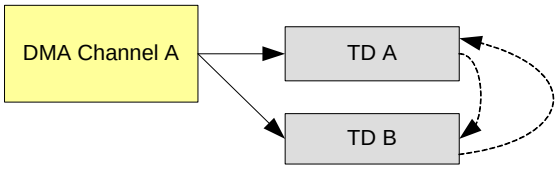
Figure 6-23. Auto Repeat DMA



6.5.3 Ping Pong DMA

Double buffering is used to allow one buffer to be filled by one client, while another client is consuming the data previously received in the other buffer. In its simplest form, this is done by chaining two TDs together where each TD calls the opposite TD when complete.

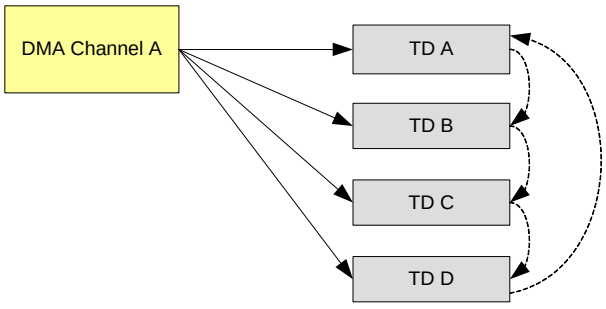
Figure 6-24. Ping Pong DMA



6.5.4 Circular DMA

This is similar to ping pong DMA except that it contains more than two buffers. In this case, there are multiple TDs where after the last TD is complete it chains back to the first TD.

Figure 6-25. Circular DMA

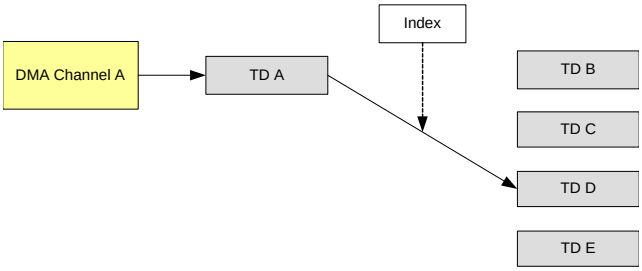


6.5.5 Indexed DMA

An external master requires access to locations on the system bus as if those locations were shared memory.

Example: If a peripheral is configured as an SPI or I2C slave where an address is received by the external master, that address becomes an index or offset into the internal system bus memory space. This is accomplished with an initial “address fetch” TD that reads the target address location from the peripheral and writes that value into a subsequent TD in the chain. This causes the TD chain to be modified during the process. When the “address fetch” TD completes, it can move onto the next TD, which has the new address information embedded in it. This TD carries out the data transfer with the address location requested by the external master.

Figure 6-26. Indexed DMA



6.5.6 Scatter Gather DMA

Multiple noncontiguous sources or destinations are required to effectively carry out an overall DMA transaction.

Example: A packet can be required to be transmitted off of the device and the packet elements, including the header, payload, and trailer exist in various non-continuous locations in memory. Scatter-gather DMA allows the segments to concatenate together by using multiple TDs in a chain that gathers data from multiple locations.

A similar concept applies for the reception of data onto the device. Certain parts of the received data may need to be scattered to various locations in memory for software- processing convenience. Each TD in the chain specifies the location for each discrete element in the chain.

6.5.7 Packet Queuing DMA

This is similar to scatter gather DMA, but it specifically connotes packet protocols whereby there can be separate configuration, data, and status phases associated with sending or receiving a packet.

Example: To transmit a packet, a memory mapped configuration register can be written inside a peripheral specifying the overall length of the ensuing data phase. This configuration information can be set up by the CPU anywhere in system memory and copied with a simple TD to the peripheral. After the configuration phase, a data phase TD (or a series

of data phase TDs) can begin (potentially using scatter gather). After the data phase TDs finish, a status phase TD can be invoked that reads some memory mapped status information from the peripheral and copies it to a location in system memory specified by the CPU for later inspection. Multiple sets of configuration/data/status phase sub-chains can be strung together to create larger chains that transmit multiple packets in this way. A similar concept exists in the opposite direction for the reception of the packets.

6.5.8 Nested DMA

One TD can modify another TD, as the TD configuration space is memory mapped, just as any other peripheral.

Example: A first TD loads a second TDs configuration and then calls the second TD. The second TD moves data as required by the application. When complete, the second TD calls the first TD, which again updates the second TDs configuration. This process repeats as often as necessary.

6.6 Register List

Table 6-3. PHUB and DMA Register List

Register Name	Comments	Features
PHUB_CFG	PHUB General Configuration register	Specifies prune_clock delay, number of wait states, allocation fairness algorithm, priority, priority spoke, CPU_CLOCK_EN setting
PHUB_ERR	PHUB Error Detection register	PHUB detects the following errors: 1. Bus Timeout 2. Unpopulated address access 3. Peripheral AHB ERROR response If the error was detected as a result of a CPU access then PHUB will send an AHB ERROR response to the CPU. If the error was detected as a result of either a CPU or DMA access then PHUB will set the corresponding bit in the following ERR register.
PHUB_ERR_ADDR	PHUB Error Address register	Contains the address that caused an error to trigger
PHUB_CH[0..23]_BASIC_CFG	Channel Basic Configuration register	Sets basic channel configurations in gates inside PHUB
PHUB_CH[0..23]_ACTION	Channel Action register	Sets action for each channel
PHUB_CH[0..23]_BASIC_STATUS	Channel Basic Status register	Provides status information in gates inside PHUB
PHUB_CFGMEM[0..23]_CFG0	PHUB Channel Configuration register 0	Each channel has some configuration information stored in RAM. This configuration information is called CHn_CFG0/1. CHn_CFG0/1 are stored in CFGMEM at {CH_NUM[5:0], 000}.
PHUB_CFGMEM[0..23]_CFG1	PHUB Channel Configuration register 1	
PHUB_TDMEM[0..127]_ORIG_TD0	PHUB Original Transaction Descriptor 0	Each channel has a TD chain (as short as one TD in length) that provides instructions to the DMAC for carrying out a DMA sequence for the channel. The TD chain is comprised of one or more CHn_ORIG_TD0/1 TDs. DMAC accesses the CHn_ORIG_TD0/1 chain from TDMEM and the address in TDMEM of the current TD in the chain is {TD_PTR[7:0], 000}.
PHUB_TDMEM[0..127]_ORIG_TD1	PHUB Original Transaction Descriptor 1	

7. Interrupt Controller

The Interrupt Controller provides the mechanism for hardware resources to change the program address to a new location independent of the current execution in the main code. The interrupt controller also handles continuation of the interrupted code being executed after the completion of the interrupt service routine.

7.1 Features

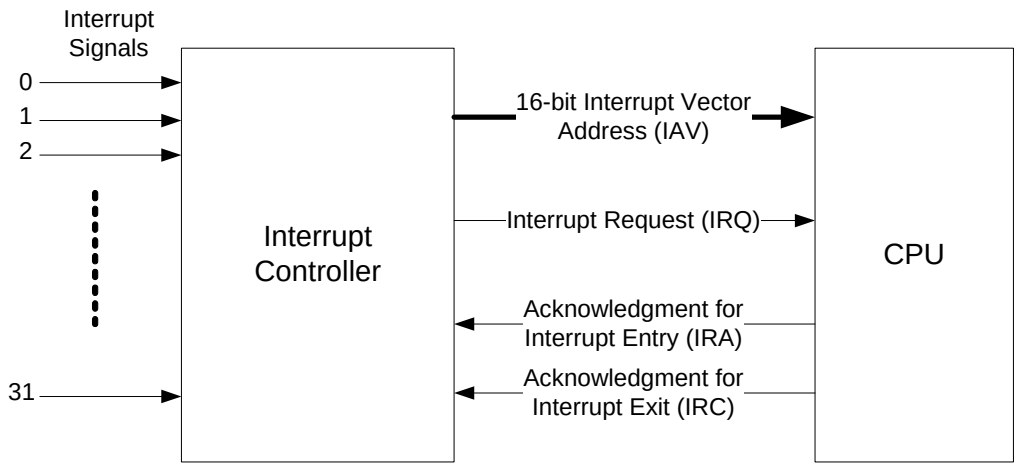
The following are features of the interrupt controller:

- Supports 32 interrupt lines
- Programmable interrupt vector
- Configurable priority levels from 0 to 7
- Support for dynamic change of priority levels
- Support for individual enable/ disable of each interrupt
- Nesting of interrupts
- Multiple sources for each interrupt line (can be either fixed function, UDB, or from DMA)
- Supports both level trigger and pulse trigger
- Tail chaining, late arrivals and exceptions are supported in PSoC® 5 devices

7.2 Block Diagram

Figure 7-1 is a block diagram of the interrupt controller.

Figure 7-1. Interrupt Controller Block Diagram



7.3 How It Works

The interrupt controller supports 32 interrupt signals. The interrupt signal can come from one of the three sources (see Figure 7-2):

- Fixed function block
- DMA channels
- UDB blocks

The interrupt lines pass through a multiplexer. The mux selects one among the following: Fixed function IRQ (Interrupt request), UDB IRQ with level, UDB IRQ with Edge, and DMA IRQ. The IDMUX.IRQ_CTL register is used to configure the mux for the IRQ selection. See Table 7-1 for the different interrupt sources that can be selected for each interrupt vector number.

Figure 7-2. Interrupt and DMA Processing in the IDMUX

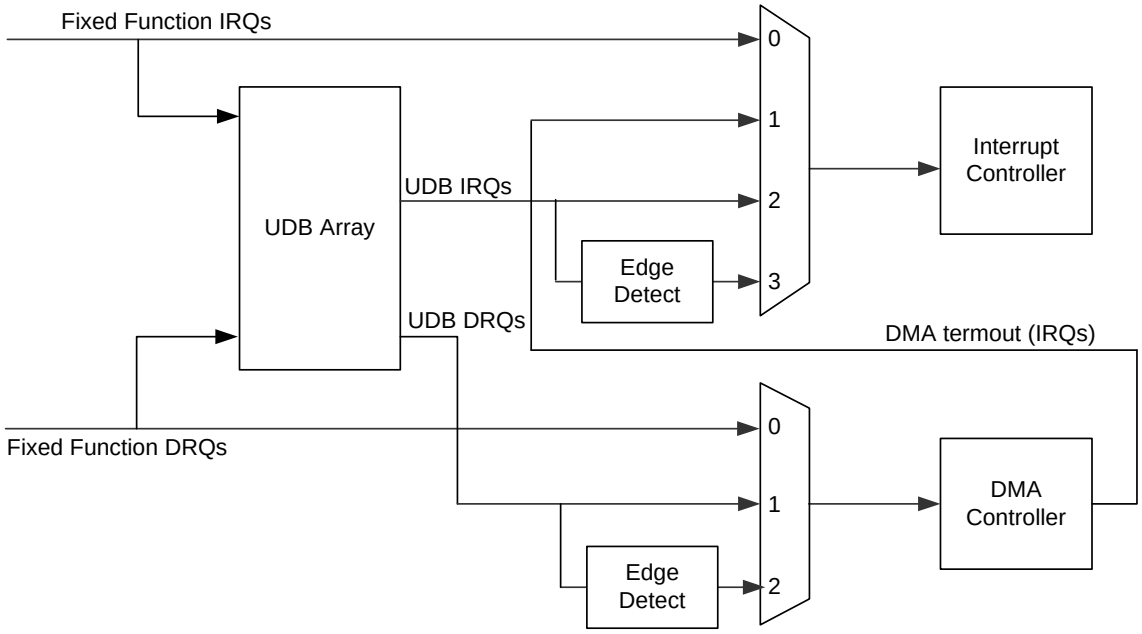


Table 7-1. Interrupt Vector Table

Interrupt #	Cortex-M3 Exception #	Fixed Function	DMA	UDB
0	16	Low voltage detect (LVD)	phub_termout0[0]	udb_intr[0]
1	17	Cache	phub_termout0[1]	udb_intr[1]
2	18	Reserved	phub_termout0[2]	udb_intr[2]
3	19	Pwr Mgr	phub_termout0[3]	udb_intr[3]
4	20	PICU[0]	phub_termout0[4]	udb_intr[4]
5	21	PICU[1]	phub_termout0[5]	udb_intr[5]
6	22	PICU[2]	phub_termout0[6]	udb_intr[6]
7	23	PICU[3]	phub_termout0[7]	udb_intr[7]
8	24	PICU[4]	phub_termout0[8]	udb_intr[8]
9	25	PICU[5]	phub_termout0[9]	udb_intr[9]
10	26	PICU[6]	phub_termout0[10]	udb_intr[10]
11	27	PICU[12]	phub_termout0[11]	udb_intr[11]
12	28	PICU[15]	phub_termout0[12]	udb_intr[12]
13	29	Comparators Combined	phub_termout0[13]	udb_intr[13]

Table 7-1. Interrupt Vector Table *(continued)*

Interrupt #	Cortex-M3 Exception #	Fixed Function	DMA	UDB
14	30	Switched Caps Combined	phub_termout0[14]	udb_intr[14]
15	31	I^2C	phub_termout0[15]	udb_intr[15]
16	32	Reserved	phub_termout1[0]	udb_intr[16]
17	33	Reserved	phub_termout1[1]	udb_intr[17]
18	34	Reserved	phub_termout1[2]	udb_intr[18]
19	35	Reserved	phub_termout1[3]	udb_intr[19]
20	36	Reserved	phub_termout1[4]	udb_intr[20]
21	37	USB SOF Int	phub_termout1[5]	udb_intr[21]
22	38	USB Arb Int	phub_termout1[6]	udb_intr[22]
23	39	USB Bus Int	phub_termout1[7]	udb_intr[23]
24	40	USB Endpoint[0]	phub_termout1[8]	udb_intr[24]
25	41	USB Endpoint Data	phub_termout1[9]	udb_intr[25]
26	42	Reserved	phub_termout1[10]	udb_intr[26]
27	43	Reserved	phub_termout1[11]	udb_intr[27]
28	44	DFB Int	phub_termout1[12]	udb_intr[28]
29	45	Decimator Int	phub_termout1[13]	udb_intr[29]
30	46	phub_err_int	phub_termout1[14]	udb_intr[30]
31	47	eeprom_fault_int	phub_termout1[15]	udb_intr[31]

The interrupt controller unit prioritizes and sends the request to the CPU for execution. The list of interrupt sources and the corresponding interrupt number is available in the device datasheet.

7.3.1 Enabling Interrupts

The interrupt controller provides features to enable and disable individual interrupt lines. The Enable register (SETEN) and the Clear Enable register (CLREN), respectively, enable and disable the interrupt lines. Each bit in the register corresponds to an interrupt line; these registers enable and disable interrupts and read the enable status of interrupts. The register that is updated latest (SETEN or CLREN register) determines the interrupt enable status. Table 7-2 shows the status of bits during read and write.

Table 7-2. Bit Status During Read and Write

Register	Operation	Bit Value	Comment
SETEN	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
CLREN	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

7.3.2 Pending Interrupts

When the interrupt controller receives the interrupt signal, it sets the pending bit.

“Set Pending register” (SETPEND) and the “Clear Pending register” (CLRPEND) also allow the pending bit to be set and cleared through software. Each bit in the register corresponds to an interrupt line. The pending bit status can be read by reading these registers. For both pulse/level interrupts, the pending bit is cleared immediately upon receiving the acknowledgement from the CPU on interrupt entry (IRA). For pulse interrupts, the pending bit can be set again by arrival of a new pulse interrupt on the same line after the IRA. But for level interrupt, the interrupt controller checks the status of the interrupt line when it receives the acknowledgement from the CPU on interrupt exit (IRC). During that time, if the interrupt line is still asserted, the pending bit is reset. If there is no assertion on the interrupt line, the pending bit remains in cleared state.

Table 7-3. Pending Bit Status

Register	Operation	Bit Value	Comment
SETPEND	Write	1	To put an interrupt to pending
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending
CLRPEND	Write	1	To clear a pending interrupt
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending

The pending register can also be written by software. When the software writes a 1 to the pending bit, it activates the interrupt. When software clears the pending bit, the interrupt does not occur. When the software request to clear a pending bit and hardware request to set the pending bit occurs simultaneously, the hardware request takes the higher priority.

Setting of the pending bit when the same bit is already set results in only one execution of the interrupt. The pending bit can be updated regardless of whether the corresponding enable bit is set. If the enable bit is not set, the interrupt line will be pended until the interrupt is enabled, unless the user clears the bit. It is advisable to check the state of the pending bit before enabling the interrupt. The choice is left to the user, of whether to set the pending bit before or after the enable bit is set, for enabling the corresponding interrupt.

7.3.3 Interrupt Priority

The interrupt controller provides a priority handling feature to help a user assign priority for each interrupt. Characteristics of this feature are as follows:

- Eight levels of interrupt priorities from 0 to 7.
- Priority level 0 is highest and level 7 is lowest.
- Priority levels set using the Interrupt Priority Registers PRI_[x].
- Support of dynamic configuration of priority levels – A change of priority level of an interrupt on the fly does not affect the current execution of the same interrupt; it takes effect for the next assertion.

Priority handling is very important in the following cases:

- **Case 1** – If an interrupt (INT B) is asserted when another interrupt (INT A) is being executed, there are three possibilities with unique handling sequences:
  - If INT A has lower priority than INT B:
    1. INT A is stopped at the point of execution.
    2. The details of INT A are pushed to the stack, and INT B begins to execute.
    3. After the execution of INT B, INT A execution is resumed from the point of its interruption.
  - If INT A has higher priority than INT B:
    1. INT B must wait until INT A is executed.
    2. After the execution of INT A, INT B can start execution.
  - If INT A and INT B have equal priority:
    1. If INT A is being executed; INT B must wait until INT A is executed. After the execution of INT A, INT B can start execution.
    2. If INT B is being executed; INT A must wait until INT B is executed. After the execution of INT B, INT A can start execution.
- **Case 2** – During the simultaneous occurrence of interrupts:
  - If INT A has lower priority than INT B, then INT B wins arbitration and begins to execute.
  - If INT A has higher priority than INT B, then INT A wins arbitration and begins to execute.
  - If INT A and INT B have equal priority, then the interrupt with the lower index number wins arbitration and begins to execute.

7.3.4 Level versus Pulse Interrupt

The interrupt controller supports both Level and Pulse interrupts. The interrupt controller includes the Pulse detection logic, which detects the rising edge on the interrupt line. The pulse detection logic pends the interrupt bit whenever it detects the rising edge. The interrupt controller detects any assertion in the interrupt signal and executes the interrupt as follows:

- **Level Interrupt** – With level interrupts, the interrupt request bit in the corresponding peripheral register must be cleared by the firmware inside the interrupt service routine. If the interrupt request bit in the peripheral register is set, it results in a level high signal on the interrupt line. At the interrupt exit, if the interrupt request bit is set in the peripheral register, the interrupt pending bit is set again and the interrupt is processed again if it is enabled.
- **Pulse Interrupt** – A pulse occurs at the interrupt line. The low to high edge of the pulse sets the pending bit and the corresponding interrupt is executed. If the pulse occurs while the pending bit is already set, the second pulse has no effect, because the pending bit is already set. The Pending bit is automatically cleared by the interrupt controller at ISR entry. However, if the pulse comes while the interrupt is currently active, the interrupt pending bit is set again, and the interrupt is executed again.

7.3.5 Interrupt Execution

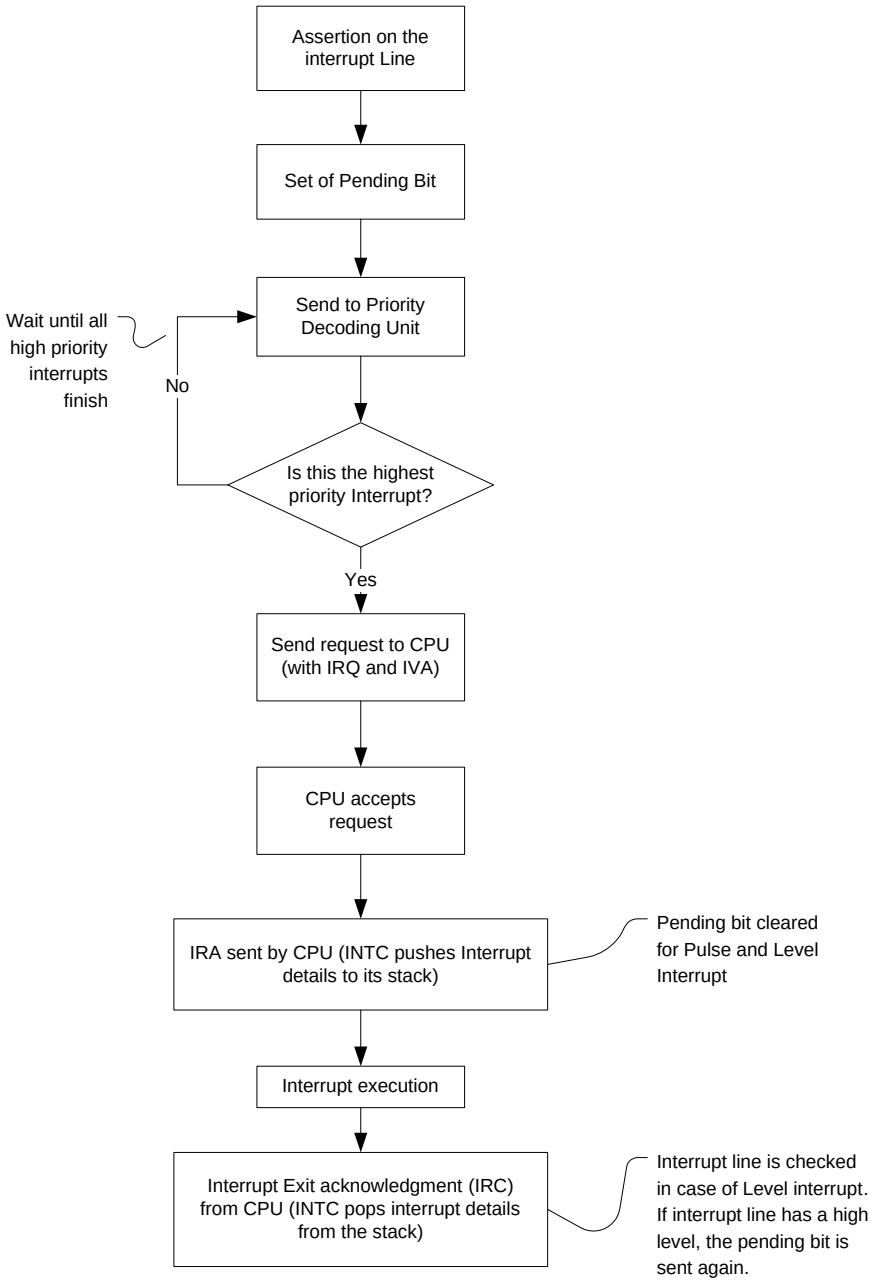
The interrupt controller controls both Level and Pulse interrupt in the following sequence:

1. Interrupt execution corresponding to the interrupt signal requires the interrupt to be enabled (assuming priority and interrupt vector address are programmed already).
2. When an assertion occurs in the interrupt signal, the pending bit corresponding to the interrupt number is set in the pending register, indicating that the interrupt is waiting for its execution.

3. The Priority Decoding unit reads the priority and determines when the interrupt can be executed.
4. The interrupt controller sends the interrupt request to the CPU, along with the interrupt vector address for execution.
5. The CPU receives the request.
6. **Interrupt Entry (IRA)** – The CPU acknowledges the interrupt entry. The next assertion in the same interrupt line can be detected only after the interrupt entry. Any assertions before that are ignored. The interrupt controller clears the pending bit upon receiving the acknowledgement.
7. The current interrupt number and its priority are pushed to the interrupt controller stack by the interrupt controller.
8. **Interrupt Exit (IRC)** – When interrupt execution is completed, the processor is free to address the next request.The CPU acknowledges the interrupt exit. At the interrupt exit, the interrupt context (i.e., interrupt number and priority) is popped from the stack.

Figure 7-3 lists the basic operations during an interrupt signal assertion and its handling.

Figure 7-3. Interrupt Signal Assertion and Handling



7.4 PSoC 5 Features

Because PSoC 5 architecture is based on the Cortex-M3 core, it has additional features supported by the Cortex-M3 core. In PSoC 5 devices, the interrupt controller is a part of the Cortex-M3 core. For more detailed information about the PSoC 5 Interrupt Controller, refer to the *ARM Cortex-M3 Technical Reference Manual* available at http://www.arm.com.

7.4.1 Active Interrupts

An active interrupt is the one being executed currently. The interrupt priority and interrupt number of the active interrupt are stored in the CPU stack. Whenever an interrupt begins to execute, the interrupt priority and number are pushed to the stack. The contents of the stack can be read to find the Active Interrupt details. With PSoC 5 devices, the CPU stack is used. There are two stacks accessed using two different stack pointers: The Process Stack Pointer (PSP) and the Main Stack Pointer (MSP).

Cortex-M3 can be configured to use two stacks. When it is configured to use both the stacks, the first interrupt uses the PSP or the MSP to store interrupt details, depending on which is currently active. The stack grows downwards. A nested interrupt uses only MSP to store the details. When it is not configured to use two stacks, only the MSP is used.

PSoC 5 devices also support an ACTIVE register to store the active status of the interrupt. Its characteristics are:

- Each bit in the register indicates the active state of the corresponding interrupt.
- When the bit is set to 1 in the ACTIVE register, the interrupt is active. When the bit is set to 0, the interrupt is currently inactive.
- When the current running interrupt is suspended due to a high priority interrupt, the state of the current running interrupt is maintained as “Active” because it continues its execution after execution of the high priority interrupt.
- The active state of the bit is cleared only after execution of the interrupt.

PSoC 5 devices also supports exceptions other than interrupts. The ACTIVE bits correspond only to interrupts and not to exceptions. The active status details of exceptions are stored in the Exception Status register. Exception Status registers are not only used to read the active status but also to enable exceptions.

7.4.2 Interrupt Nesting

Nesting of an interrupt occurs when a high priority interrupt is asserted during a low priority interrupt execution. With PSoC 5 architecture, only the CPU stack is available to store all nesting interrupt details.

- Current interrupt number, current interrupt priority
- Program counter, PSR, R0 to R3, R12 and LR
- Depending on the application, other registers from R4 to R11

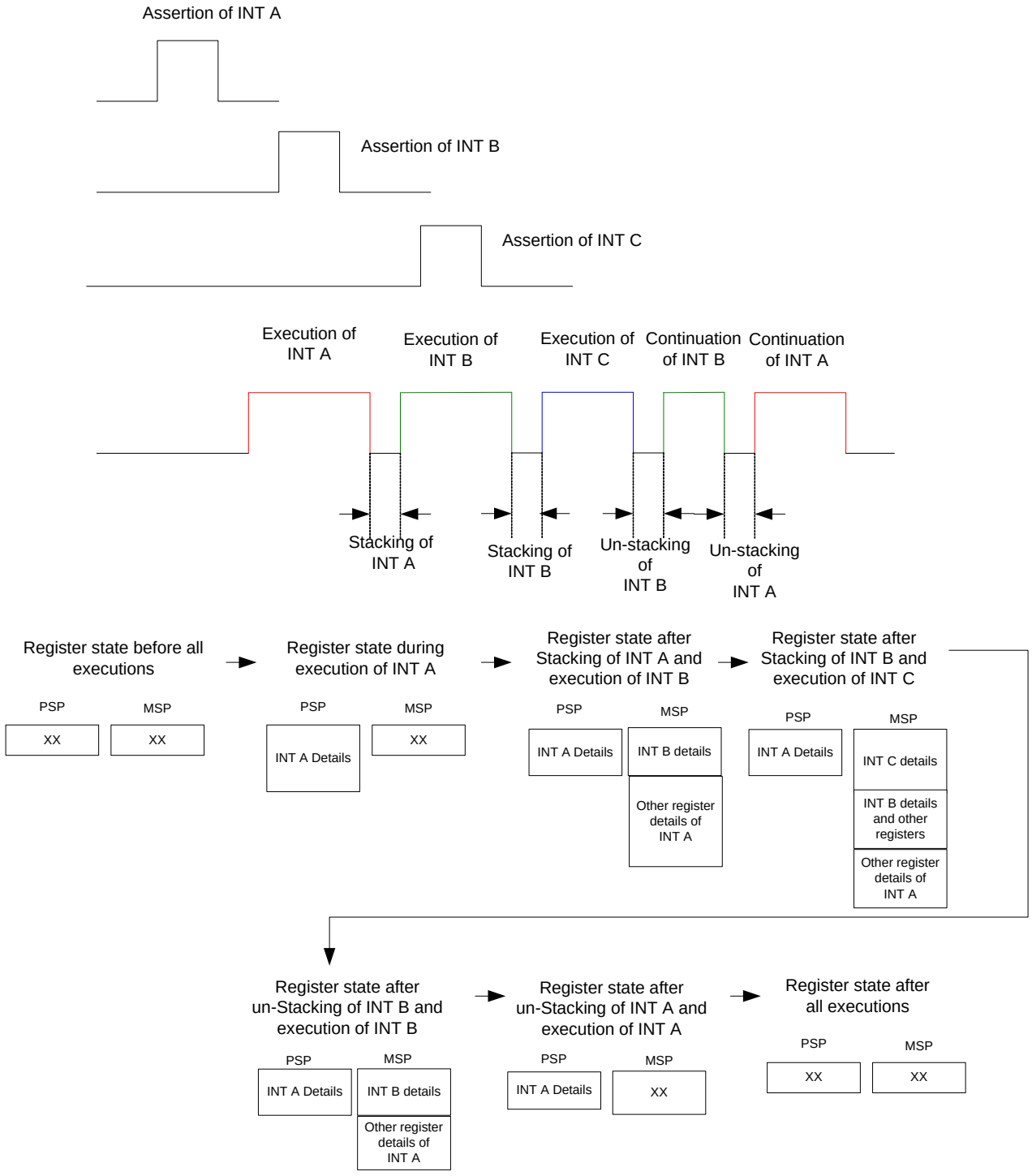
The CPU stack grows down while the CPU handles push and pop.

The configuration controls how you use PSP and MSP. If both stacks are used, the Process Stack Pointer or Main Stack Pointer, which ever is currently active, is used by the first interrupt. All other nested interrupts use only the MSP. If only one stack is configured for use, the interrupt details are stored in the MSP. The sequence is:

1. When the high priority interrupt comes during the execution of the low priority interrupt, the interrupt controller sends a request to the CPU and low priority interrupt execution is stopped by the CPU at that point.
2. The details, such as instruction pointer and other general purpose registers for the low priority interrupt, are pushed to the stack. (The stack used depends on nesting. It can be either MSP or PSP as explained previously).
3. The number of nesting supported depends on the availability of stack space. Because system stack is used, the user should ensure that sufficient stack space is available. Insufficient stack space causes undetermined results. After the stack push for the low priority is done, the details of the current active interrupt (high priority interrupt) is stored in the CPU stack. The high priority interrupt executes.
4. After the higher priority interrupt has executed, the interrupt details of the high priority interrupt are popped from the stack. Following this, the details of the low priority interrupt (PC and other register details) are popped from the stack. The low priority interrupt continues its execution from the point of suspension.
5. Because the push and pop of stack is handled by the hardware, there is minimum latency; no instruction is involved in the operation.

Figure 7-4 on page 73 shows a timing diagram of the register states during the nesting operation.

Figure 7-4. Register Timing During Nesting



In Figure 7-4, INT A is suspended, and the high priority interrupt INT B is executed. During nesting, the INT A is pushed to the stack. During execution of INT B, INT C occurs. So INT B is pushed, and INT C is executed. After INT C is executed, INT B is popped and executed. After INT B is executed, the stack is popped. When an interrupt begins to execute, interrupt information is stored in the stack; when it completes, the stack is popped. The use of both PSP and MSP is shown. It is assumed that PSP is active during the first interrupt and that the first active interrupt uses the PSP.

7.4.3 Interrupt Vector Addresses

PSoC 5 architecture has a feature that allows a user to specify the interrupt service routine for every interrupt line. The call of the interrupt service routine corresponding to an interrupt line is not a branch instruction. The address of the interrupt service routine is stored in the vector table, which results in the direct call of the routine. This method of execution prevents latency in the call of the interrupt service routine.

When interrupt assertion occurs, the following sequence occurs:

1. The address of the interrupt service routine is taken from the interrupt vector table and is executed.
2. The list of interrupt vector addresses is stored in the vector table.

   The interrupt service routine address is programmable and is stored in the vector table. The vector table is a location in the memory and has a base address; the other vector addresses are accessed as offset from the base address. By default, the vector table is at location 0x00 in the ROM.

   The base address of the vector table can be changed; the vector table can be moved, either in the ROM itself or to the RAM. Each vector address is 32 bits long; when moving the vector table, the user should ensure that there is enough space to hold the supported 4-byte addresses for the 32 interrupt lines
3. PSoC 5 devices contain the Vector Table Offset register that contains two data:

   Position of vector table in ROM/RAM.

   Offset value from the start ROM or RAM region. This offset value acts as the base address for the vector table.
4. When the vector table is moved, the boot image should contain the stack pointer value, Reset vector, NMI vector, and hard Fault vector, because these are required for the beginning of execution of code.
5. Because the vector address is 32 bits long, the LSB is filled with 0x01, and the MSB contains the corresponding 24-bit ISR address to be executed. The presence of 0x01 in the LSB indicates Thumb instructions.
6. During the interrupt signal assertion, the address of the interrupt service routine (the Interrupt Vector Address (IVA)) is retrieved from this table and given to the CPU for execution of the interrupt.
7. Because PSoC 5 devices also support exceptions, the vector table has the address corresponding to the 15 exceptions followed by the interrupt service routine addresses.

7.4.4 Tail Chaining

Tail chaining is the process used to reduce interrupt latency. When a new interrupt assertion occurs at the same time as another interrupt being executed with the same or higher priority, the following sequence occurs:

1. The new interrupt with a lower priority is pended.
2. After the current interrupt is executed, the details of the current interrupt in the stack are not popped.
3. The details of the new interrupt are pushed to the stack and the new interrupt begins its execution.
4. After the execution of the new interrupt, details of the new interrupt and the previous interrupt are popped from the stack.

Because stacking and unstacking are avoided between the two, interrupts, latency is greatly reduced. Tail chaining can save a maximum of six cycles.

7.4.5 Late Arrival Interrupts

A late arrival interrupt occurs when another interrupt is being pushed to the stack for execution. Another feature reduces interrupt latency by handling such late arrival interrupts.

The following sequence describes the process:

1. A low priority interrupt is asserted.
2. The details of the low priority interrupt are being pushed to the stack, when a high priority interrupt assertion happens.
3. After the stacking of the low priority interrupt, the high priority interrupt is stacked and executed, instead of the low priority interrupt.
4. After execution of the high priority interrupt, the low priority interrupt is executed.

7.4.6 Exceptions

PSoC 5 architecture supports 15 different exceptions, as shown in Table 7-4.

These exceptions are used to handle fault conditions that can occur in the system. Exceptions can have fixed priority or configurable priority. Exceptions are handled in the same manner as interrupts. The State register is used to enable or disable exceptions.

Table 7-4. PSoC 5 Exceptions

Interrupt Number	Exception Type	Priority	Comments
1	Reset	–3 (highest) not programmable	Reset
2	NMI	–2 not programmable	Non-Maskable Interrupt
3	Hard Fault	–1 not programmable	All fault conditions if the corresponding handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus Fault	Programmable	Bus error; occurs when AHB interface receives an error response from a bus slave (also called prefetch abort if it is an instruction fetch or data abort if it is a data access)
6	Usage Fault	Programmable	Exceptions due to program error
7	Reserved	NA	--
8	Reserved	NA	--
9	Reserved	NA	--
10	Reserved	NA	--
11	SVCall	Programmable	System Service Call
12	Debug Monitor	Programmable	Debug monitor (watchpoints, breakpoints, external debug request)
13	Reserved	NA	--
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer

7.4.7 Interrupt Masking

PSoC 5 architecture supports special methods to mask interrupts and exceptions, preventing them from execution. Any new assertions in the interrupt lines are detected and pended until the interrupts are unmasked.

Masking of interrupts is different from enabling or disabling. When masked, the interrupt is blocked for some time, even though it is enabled. This feature is useful when it is necessary to protect some critical section of code. When interrupts are masked, pending interrupts are not executed, even though the interrupts are enabled in the enable register. The interrupts are executed only when masking is cleared.

PSoC 5 devices have special registers to provide masking facilities, including:

- **PRIMASK** – When the bit in the PRIMASK register is set, all interrupts and exceptions except NMI and Hard fault are blocked.
- **FAULTMASK** – When the bit in the FAULTMASK register is set, all interrupts and exceptions except NMI are blocked.
- **BASEPRI** – When interrupts below a certain priority level must be masked, the priority number can be specified in the BASEPRI register. All interrupts with a priority number equal to or less than the priority level specified in the BASEPRI register are masked.

7.5 Interrupt Controller and Power Modes

The CPU core (Cortex-M3) can execute even when the power or clock for the Interrupt Controller is switched off. In this case, care should be taken during entry/exit into different low power modes (alternate active, sleep and hibernate).

Perform these steps after switching off the Interrupt Controller clock.

1. Clear all pending interrupts and disable all interrupts in Interrupt Controller.
2. NOP.
3. Disable the Global Interrupt bit.
4. Turn OFF the clock for Interrupt Controller in the CLOCK_EN bit in the INTC.CLOCK_EN register.

It is preferred not to operate any Interrupt related functions when the clock to the interrupt controller is not available. When an Interrupt Service routine is executed by the CPU when the clock to the interrupt controller is switched off, the CPU should make sure the clock for the Interrupt Controller is re-enabled before the exit from the ISR (to process the IRC signal). If this is not taken care, it will lead to undefined behavior.

When returning from the lower power mode or wants to continue in the alternate active mode, follow these steps:

1. Clock must be available to Interrupt Controller
2. Enable the Global interrupt bit
3. Enable the required interrupts in the Interrupt Controller

The CPU can run when the interrupt controller clock is switched off only during active and alternate active modes. When the user wants to switch from alternate active to Active mode when the Interrupt controller clock is switched off.

a. Follow the steps mentioned above to switch off the clock for the Interrupt controller
b. Now the CPU can run any code that doesn't involve the Interrupt functionality.
c. Switch to the active state whenever required
d. To switch to active mode only on wake up on interrupt, then the CPU should keep polling the PM.MODE_CSR register to find when the system should switch to active mode.
e. When switching back to active mode, follow the procedures mentioned above for switching from low power mode to active mode.

Section C: Memory

The PSoC® nonvolatile subsystem consists of Flash, byte-writable EEPROM, and nonvolatile configuration options. The CPU can reprogram individual blocks of Flash, enabling boot loaders.

A powerful and flexible protection model allows the user to selectively lock blocks of memory for read and write protection, securing sensitive information. The byte-writable EEPROM is available on-chip for the storage of application data.

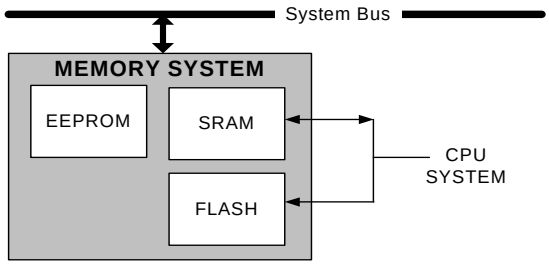
This section encompasses the following chapters:

- Nonvolatile Latch chapter on page 79
- SRAM chapter on page 81
- Flash Program Memory chapter on page 83
- EEPROM chapter on page 85
- Memory Map chapter on page 87

Top Level Architecture

(Block diagram here taken from main block diagram in Introduction.)

Memory Block Diagram



10. Nonvolatile Latch

A Nonvolatile Latch (NVL or NV latch) is an array of programmable, nonvolatile memory elements whose outputs are stable at low voltage. Each bit in the array consists of a volatile latch paired with a nonvolatile cell. On POR release, nonvolatile cell outputs are loaded to volatile latches and the volatile latch drives the output of the NVL.

10.1 Features

NV latches include a 4x8-bit Write Once NV latch for device security

10.2 Write Once NV Latch

The Write Once (WO) latch is a type of nonvolatile latch. The cell itself is an NVL with additional logic wrapped around it. Each WO latch device contains 4 bytes (32 bits) of data. The wrapper outputs a ‘1’ if a super-majority (28 of 32) of its bits match a pre-determined pattern (0x50536F43) and it outputs a ‘0’ if this majority is not reached. When the output is ‘1’, the Write Once NV latch locks the part out of Debug and Test modes; it also permanently gates off the ability to erase or alter the contents of the latch. Matching of all bits is intentionally not required, so that single (or few) bit failures do not deassert the WO latch output. The state of the NV latch bits after wafer processing is truly random with no tendency toward 1 or 0.

The WOL only locks the part when the correct 32-bit key (0x50536F43) is loaded into the NVL's volatile memory, programmed into the NVL's nonvolatile cells, and the part is reset. The output of the WOL is only sampled on reset and used to disable the access.

This precaution prevents anyone from reading, erasing, or altering the content of the internal memory.

If the device is protected with a WO latch setting, Cypress cannot perform failure analysis and, therefore, cannot accept RMAs from customers. The WO latch can be read via the SWD to electrically identify protected parts.

The user can write the key in WOL to lock out external access only if no Flash protection is set. However, after setting the values in the WOL, a user still has access to the part until it is reset. Therefore a user could write the key into the WOL, program the Flash protection data, and then reset the part to lock it. Refer to the Flash Protection chapter on page 143 for details about Flash protection.

10.3 Programming NV Latch

The volatile latch is intended to be initialized from a nonvolatile memory cell at POR release. NV Latches are configured by writing to the volatile cells of the array and then programming the volatile cell data into the nonvolatile cells (Write Nonvolatile Cell Mode). See the Nonvolatile Memory Programming chapter on page 357 for more details about NV latch programming sequence.

NVL programming is done through a simple command/status register interface. Commands and data are sent as a series of bytes to either SPC_CPU_DATA or SPC_DMA_DATA, depending on the source of the command. Response data is read via the same register to which the command was sent. The following commands are used to program NVLs:

- Command 0x00 – Load Byte

  Loads a single byte of data into the volatile cells at the given address.
- Command 0x10 – Read Byte

  Reads a single byte of data from volatile cells at the given address.
- Command 0x06 – Write User NVL

  Writes all nonvolatile cells in a User NVL with the corresponding values in its volatile latches.
- Command 0x03 – Read User NVL

  Reads a single byte of data from nonvolatile cells at the given address. Note that when this command is executed, all of the bytes are transferred from nonvolatile cells to the volatile cells of the array.

To program the write once (WO) nonvolatile latch, the Vddd supply voltage should be less than or equal to 3.3 V, and the programming temperature should be between 10 °C and 40 °C (Vddd = 3.3 V and TJ = 25 °C ±15 °C). Note that the constraints are only on the Vddd power supply pin and not on Vdda or Vddio power supply pins. The WO NVL programming is an optional step and is required only for applications that require the Device Security feature to be enabled.

10.4 Sleep Mode Behavior

NV latches remain powered up during sleep, but they stay in an idle state, not allowing any direct reads or writes. During sleep, the outputs of the NVLs remain stable.

11. SRAM

PSoC® 5 devices include on-chip SRAM. These families offer devices that range from 2 to 64 kilobytes.

11.1 Features

The PSoC 5 SRAM has these features:

- Organized as up to 16 blocks of 4 KB each, for CY8C55 family.
- Code can be executed out of portions of SRAM, for CY8C55 family.
- 8-, 16-, or 32-bit accesses.
- Zero wait state accesses.
- Arbitration of SRAM accesses by the CPU and the DMA controller.
- Different blocks can be accessed simultaneously by the CPU and the DMA controller.

11.2 Block Diagram

The block diagram for the CY8C55 family is as indicated. Figure 11-1 shows CY8C55 family SRAM accesses.

Figure 11-1. CY8C55 Family SRAM Accesses

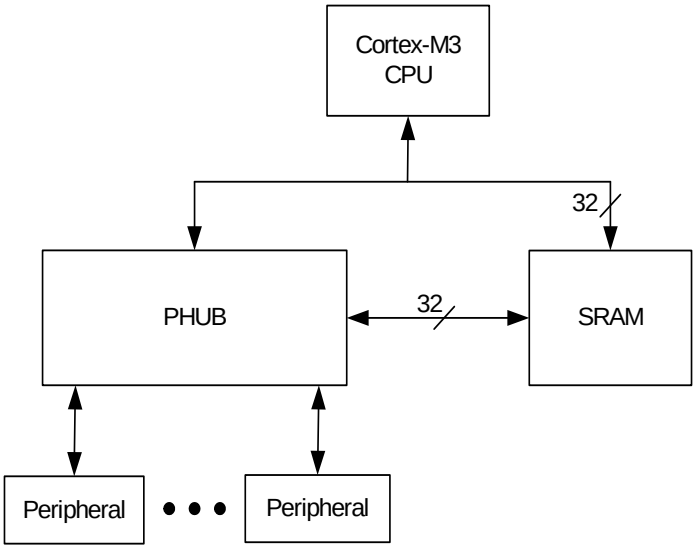
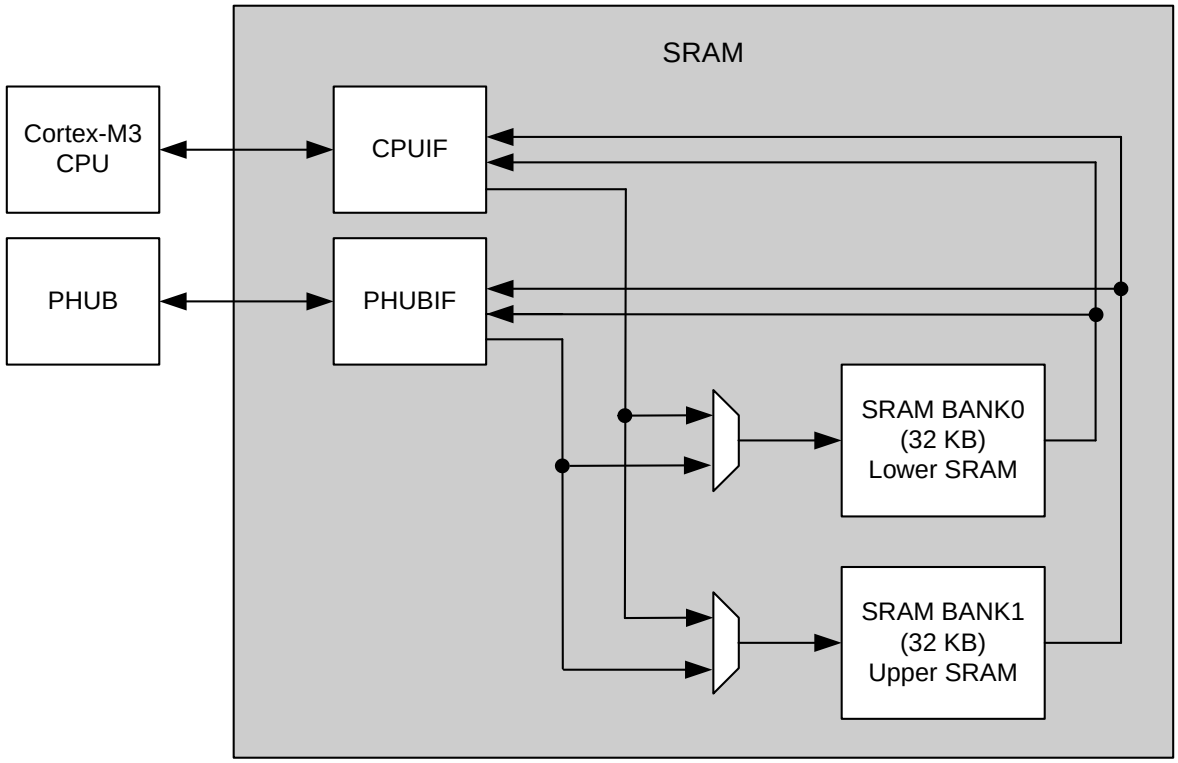


Figure 11-2 shows internal SRAM organization for the CY8C55 family.

Figure 11-2. CY8C55 Family SRAM Organization



11.3 How It Works

The CY8C55 family has up to 64 KB SRAM implemented as sixteen 4 KB blocks. All 64 KB are accessible by the Cortex-M3 CPU and by the PHUB DMA controller in normal operation. The SRAM is further organized as two 32 KB memory banks, centered at address 0x20000000. This allows access to both SRAM banks with either the c-Bus (Cortex-M3 I and D buses) or the s-Bus (Cortex-M3 system bus). Code can be executed from all SRAM below address 0x20000000. Note that when accessed by the DMA, the SRAM that is in the region of 0x1FFF8000 to 0x1FFFFFFF is remapped to 0x20008000 to 0x2000FFFF. The net result is that for DMA, the SRAM is all on a single 64KB page. Also, the memory stays contiguous because it will wrap from 0x2000FFFF to 0x20000000.

The PHUB can use SRAM as a DMA source or target.

All data paths to SRAM are 32 bits wide.

The CPU has a direct connection to SRAM without going through the PHUB. In addition to faster SRAM access by the CPU, this allows for simultaneous accesses to SRAM by both the CPU and the PHUB DMA controller, because SRAM is physically implemented as multiple separate blocks. If the CPU and the PHUB are accessing separate blocks, they both have simultaneous unimpeded access.

In case of contention, the following applies:

- In most cases, the Cortex-M3 CPU has priority over the PHUB for all SRAM.

The SRAM responds to CPU, PHUB, and CY8C DoC accesses with zero wait states for both reads and writes as long as the access does not lose priority arbitration. Arbitration is done on a cycle-by-cycle basis at the time of SRAM access. The losing master is held off until the winning master has finished accessing the SRAM block; the losing master gains access on the cycle immediately after.

SRAM data is maintained during all low power and sleep modes. At reset, the SRAM contents are not initialized; they power up as unknown values.

12. Flash Program Memory

PSoC® 5 includes on-chip Flash memory. This family offers devices that range from 16 to 256 kilobytes. Up to an additional 32 KB of flash space is available for storing device configuration data and bulk user data.

12.1 Features

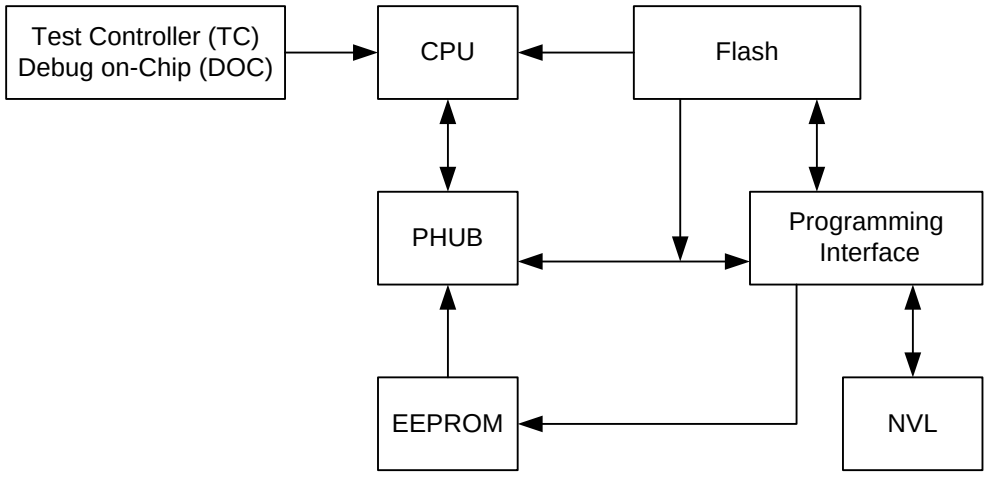
PSoC 5 Flash memory have the following features:

- Organized in rows, where each row contains 256 data bytes plus 32 bytes for device configuration data and bulk user data.
- Organized as either one block of 128 or 256 rows, or as multiple blocks of 256 rows each.
- Stores CPU program and bulk or nonvolatile data
- For PSoC 5 architecture: CY8C55 Family, 8-, 16-, or 32-bit read accesses.
- Programmable with a simple command / status register interface (see Nonvolatile Memory Programming chapter on page 357).
- Four levels of protection (see Nonvolatile Memory Programming chapter on page 357 and Flash Protection chapter on page 143).

12.2 Block Diagram

Figure 12-1 is a block diagram of the Flash programming system.

Figure 12-1. Flash Block Diagram



12.3 How It Works

Flash memory provides nonvolatile storage for firmware, device configuration data, bulk data storage, factory configuration data, and protection information.

Flash memory contains two regions – a main region, and a much smaller, extended region. All user data is stored in the main region, including device configuration data. Factory configuration and user-defined protection data are stored in the extended region, also known as the hidden rows of Flash.

For each row, protection bits control whether the Flash can be read or written by external debug devices and whether it can be reprogrammed by a boot loader. For more information see the Nonvolatile Memory Programming chapter on page 357 and Flash Protection chapter on page 143.

The CPU or DMA controller read both user code and bulk data located in flash through the cache controller. This provides higher CPU performance. Flash programming is performed through a special interface and preempts code execution out of flash. Code execution out of cache may continue during flash programming as long as that code is contained inside the cache.

Flash is erased in 64-row sectors or in its entirety, and it is programmed in rows. Erase and programming operations are done by a programming system, using a simple command/status register interface. For more information, see the Nonvolatile Memory Programming chapter on page 357.

12.4 Flash Memory Access Arbitration

Flash memory can be accessed either by the cache controller or the nonvolatile memory programming interface (system performance controller (SPC)). Cache controller can perform only flash read operations while the SPC can perform both read and write operations on the flash memory. There is an internal arbitration mechanism to facilitate flash memory access by both the cache and the SPC. Flash memory is organized as flash arrays. PSoC 5 can have up to four flash arrays, where each flash array size can be up to 64 KB. Both the SPC and the cache controller can simultaneously access the flash memory locations that are present in different flash arrays. On the other hand, if cache controller tries to access the same flash array already being accessed by the SPC, then it must wait until the SPC completes its flash access operation. The CPU, which accesses the flash memory through the cache controller, is also halted until the cache is filled with the code to be executed from the flash memory. Similarly, if SPC tries to access the flash array already being accessed by the cache controller, then it must wait until the cache controller completes its access operation.

13. EEPROM

PSoC® 5 devices have on-chip EEPROM memory to store user data. This family offers devices that range from 512 bytes to 2 kilobytes of EEPROM memory.

13.1 Features

The PSoC 5 EEPROM memory has the following features:

- Organized in rows, where each row contains 16 bytes
- Organized as one block of 32, 64, or 128 rows, depending on the device
- Stores nonvolatile data
- Allows to erase and write at the byte and row levels using SPC commands
- Supports erase operation at the sector level, where a sector is 64 rows
- Byte read access by CPU or DMA using the PHUB
- Programmable with a simple command/status register interface (see Nonvolatile Memory Programming chapter on page 357)

13.2 Block Diagram

There is no block diagram associated with EEPROM.

13.3 How It Works

EEPROM memory provides nonvolatile storage for user data. EEPROM allows erase and write operation at the byte and row levels using SPC commands. It may be read by both the CPU and the DMA controller, using the PHUB. All read accesses are 8-bit. CPU code execution can continue from flash during EEPROM writes.

If a PHUB access is attempted while the SPC is in control of EEPROM, a System Fault Interrupt is generated to the interrupt controller and the bit EEPROM_error is set in SPC_EE_ERR[0]. When set, this bit remains set until it is read from the PHUB. Before a PHUB access of EEPROM is done, set the firmware EEPROM request bit AHB_EE_REQ in SPC_EE_SCR[0], then poll for the EEPROM acknowledge bit EE_AHB_ACK in SPC_EE_SCR[1] to be set. Before a PHUB access of EEPROM is done, firmware should set the EEPROM request bit AHB_EE_REQ in SPC_EE_SCR[0], then poll for the EEPROM acknowledge bit EE_AHB_ACK in SPC_EE_SCR[1] to be set.

EEPROM is erased in 64-row sectors, or in its entirety, and is programmed in rows. Erase, programming and read operations are done by a programming system using a simple command/status register interface. EEPROM can be written at the byte level by using the “Load byte” SPC command to load the byte at specific location in row, and then calling the “Write row” command to modify only that specific byte location of row. For more information, see Nonvolatile Memory Programming chapter on page 357.

14. Memory Map

All PSoC® 5 memory (Flash, EEPROM, Nonvolatile Latch, and SRAM) and all registers are accessible by the CPU, DMA controller, and in most cases by the debug systems. This chapter contains an overall map of the addresses of the memories and registers.

14.1 Features

The PSoC 5 memory map has the following features:

- ARM Cortex-M3 32-bit linear address space, with regions for code, SRAM, peripherals, and CPU internal registers.
- Flash is mapped to the Cortex-M3 code region.
- Half of SRAM is mapped to the code region, the other half to the SRAM bitband region.
- SRAM mapped to the code region is also accessible by DMA in the SRAM bitband region.
- All other memories, and all registers, are accessed in the Cortex-M3 peripheral bitband region.

14.2 Block Diagram

There is no block diagram associated with the memory map.

14.3 How It Works

The PSoC 5 memory map is detailed in the following sections. For additional information refer to the *PSoC® 5 Registers TRM (Technical Reference Manual)*.

14.3.1 PSoC 5 Memory Map

The ARM Cortex-M3 has a fixed address map allowing access to peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in Table 14-1. Note that code can be executed from the code and SRAM regions.

Table 14-1. Cortex-M3 Address Map

Address Range	Size	Use
0x00000000 – 0x1FFFFFFF	0.5 GB	Program code. Includes the exception vector table at power up, which starts at address 0
0x20000000 – 0x3FFFFFFF	0.5 GB	SRAM. This includes a 1 MByte bit-band region starting at 0x20000000, and a 32 Mbyte bit-band alias region starting at 0x22000000.
0x40000000 – 0x5FFFFFFF	0.5 GB	Peripherals. This includes a 1 MByte bit-band region starting at 0x40000000, and a 32 Mbyte bit-band alias region starting at 0x42000000.
0xE0000000 – 0xFFFFFFFF	0.5 GB	Internal peripherals, including the NVIC and debug and trace modules

The PSoC 5 address map is shown in Table 14-2. For more information refer to the Cortex-M3 chapter.

Table 14-2. PSoC 5 Address Map

Address Range	Purpose
0x0000 0000 – 0x0003 FFFF	Up to 256 KB Flash
0x1FFF 8000 – 0x1FFF FFFF	Up to 32 KB SRAM in code region
0x2000 0000 – 0x2000 7FFF	Up to 32 KB SRAM in SRAM region
0x2000 8000 – 0x2000 FFFF	Alias of address range 0x1FFF 8000 – 0x1FFF FFFF, accessible by DMA
0x4000 4000 – 0x4000 42FF	Clocking, PLLs, and oscillators
0x4000 4300 – 0x4000 43FF	Power management
0x4000 4500 – 0x4000 45FF	Ports interrupt control
0x4000 4700 – 0x4000 47FF	Flash programming interface
0x4000 4900 – 0x4000 49FF	I^2C controller
0x4000 4E00 – 0x4000 4EFF	Decimator
0x4000 4F00 – 0x4000 4FFF	Fixed timer/counter/PWMs
0x4000 5000 – 0x4000 51FF	General purpose I/Os
0x4000 5300 – 0x4000 530F	Output port select register
0x4000 5800 – 0x4000 5FFF	Analog subsystem interface
0x4000 6000 – 0x4000 60FF	USB controller
0x4000 6400 – 0x4000 6FFF	UDB configuration
0x4000 7000 – 0x4000 7FFF	PHUB configuration
0x4000 8000 – 0x4000 87FF	EEPROM
0x4000 C000 – 0x4000 C800	Digital filter block
0x4001 0000 – 0x4001 FFFF	Digital interconnect configuration
0x4800 0000 – 0x4800 7FFF	Flash configuration region
0xE000 0000 – 0xE00F FFFF	Cortex-M3 PPB registers, including NVIC, debug, and trace

Section D: System Wide Resources

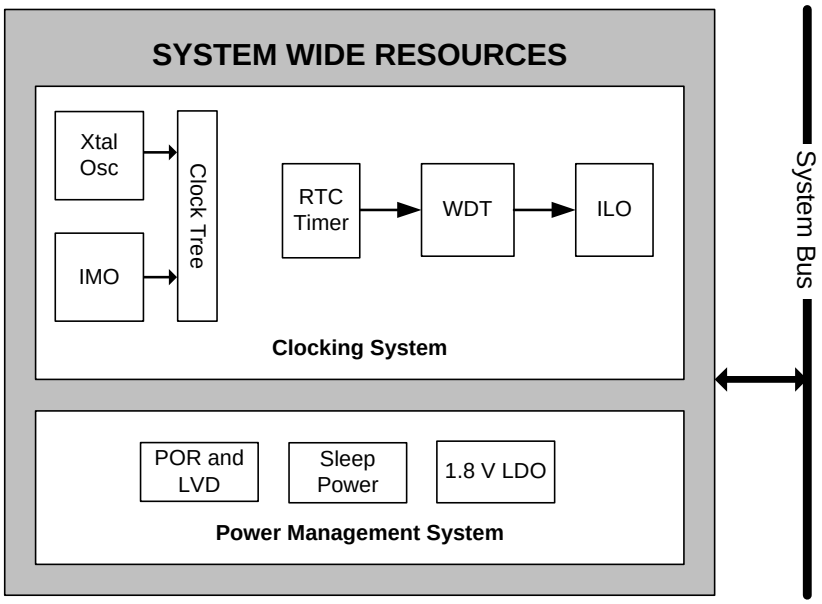
The System Wide Resources section details three types of I/O, internal clock generators, power supply, and sleep modes.

This section contains these chapters:

- Clocking System chapter on page 91
- Power Supply and Monitoring chapter on page 105
- Low Power Modes chapter on page 111
- Watchdog Timer chapter on page 117
- Reset chapter on page 119
- I/O System chapter on page 125
- Flash Protection chapter on page 143

Top Level Architecture

System Wide Resources Block Diagram



16. Clocking System

The clocking system provides clocks for the entire device. It allows the user to trade off current, frequency, and accuracy. A wide range of frequencies can be generated, using multiple sources of clock inputs combined with the ability to set divide values.

16.1 Features

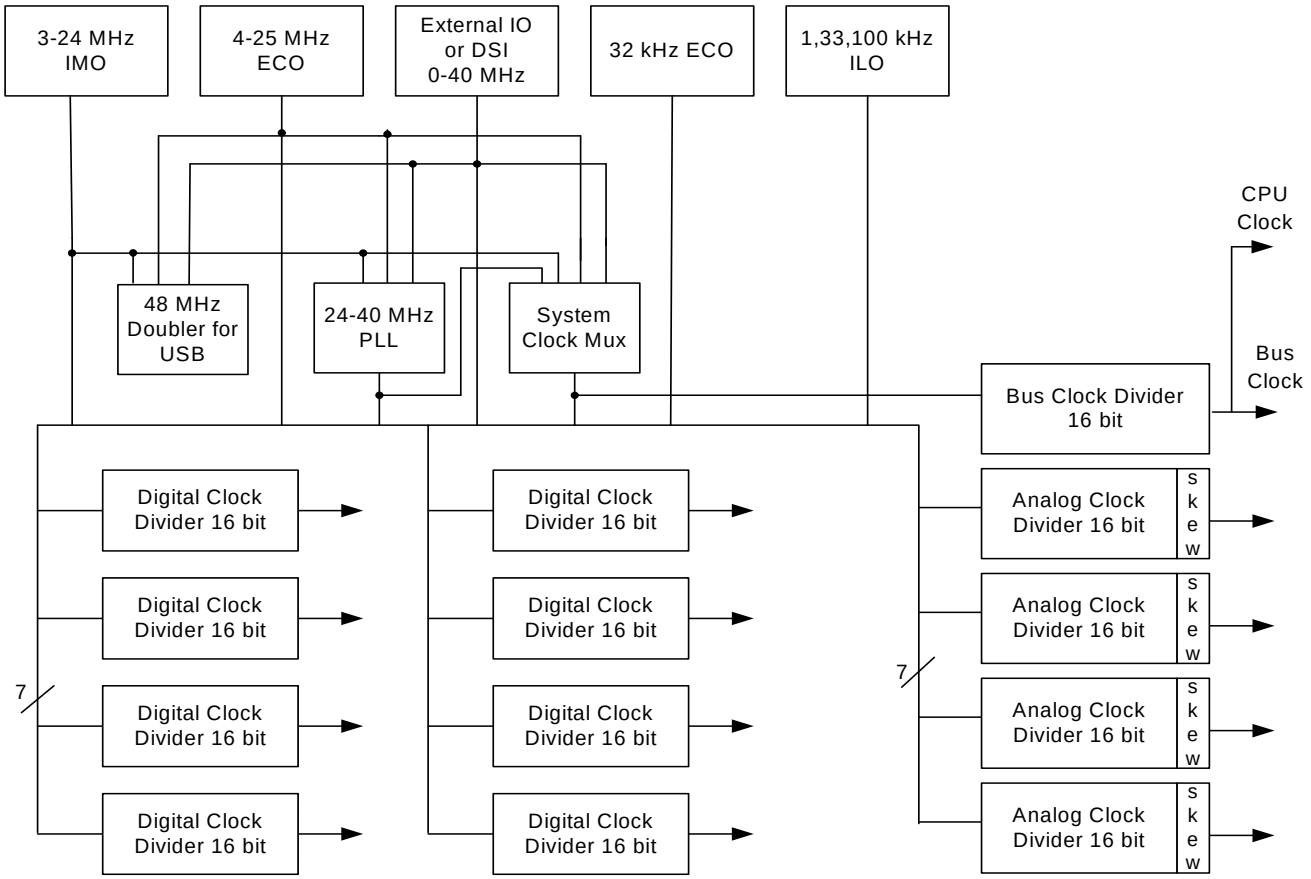
The clock system includes these clock resources:

- Four internal clock sources increase system integration:
  - 3 to 48 MHz Internal Main Oscillator (IMO) ±5% at 3 MHz
  - 1 kHz, 33 kHz, 100 kHz Internal Low Speed Oscillator (ILO) outputs
  - USB Clock Domain, sourced from IMO, MHz External Crystal Oscillator (MHzECO), and Digital System Interconnect (DSI)
  - 24 to 67 MHz fractional Phase-Locked Loop (PLL) sourced from IMO, MHzECO, and DSI
- Clock generated using a DSI signal from an external I/O pin or other logic
- Two external clock sources provide high precision clocks:
  - 4 to 25 MHz External Crystal Oscillator (MHzECO)
  - 32.768 kHz External Crystal Oscillator (kHzECO) for Real Time Clock (RTC)
- Dedicated 16-bit divider for bus clock
- Eight individually sourced 16-bit clock dividers for the digital system peripherals
- Four individually sourced 16-bit clock dividers with skew for the analog system peripherals
- IMO has a USB mode that synchronizes to USB host traffic, requiring no external crystal for USB. (USB equipped parts only)

16.2 Block Diagram

Figure 16-1 gives a generic view of the Clocking System in PSoC® 5 devices.

Figure 16-1. Clocking System Block



The components of the clocking system block diagram are defined as follows:

- Internal Main Oscillator (IMO)
- Internal Low-speed Oscillator (ILO)
- A 4 to 25 MHz External Crystal Oscillator (MHzECO)
- A 32 kHz External Crystal Oscillator (kHzECO)
- Digital System Interconnect (DSI) signal, which can be derived from the clocks developed in UDBs or off-chip clocks routed through pins
- A PLL to boost the clock frequency of some select internal and external sources
- Five types of clock outputs:
  - Digital clocks
  - Analog clocks
  - Special purpose clocks
  - System clock
  - USB clock

16.3 Clock Sources

Clock sources for the device are classified as internal oscillators and external crystal oscillators. There is an option of using a PLL to derive higher frequency outputs from existing clocks. Signals can be routed from the DSI and used as clocks in the clock trees.

16.3.1 Internal Oscillators

PSoC devices have two internal oscillators: the Internal Main Oscillator (IMO) and the Internal Low Speed Oscillator (ILO).

16.3.1.1 Internal Main Oscillator

The IMO operates with no external components and outputs a stable clock, clk_imo, at a variety of user-selectable frequencies: 3, 6, 12, 24, and 48 MHz. Frequencies are selected using the register FASTCLK_IMO_CR [2:0]. The clock accuracy is 4% typical at 3 MHz and it varies with fre-

quency. See the device data sheet for IMO accuracy specification.

USB Clock Domain

The USB clock domain is unique in that it operates largely asynchronously from the main clock network. The USB logic contains a synchronous bus interface to the chip, while running on an asynchronous clock to process USB data. The USB logic requires a 48 MHz frequency. This frequency can be generated from different sources, including DSI clock at 48 MHz or doubled value of 24 MHz from internal oscillator, DSI signal, or crystal oscillator.

Fast Start IMO (FIMO)

An alternate mode of the IMO is available for fast start-up out of sleep modes. This fast-start IMO (FIMO) mode provides a clock output within 1 µs after exiting the power down state. The fast-start IMO uses a special fast bias circuit that is stable more quickly than the high accuracy bias that is used during normal operation. This fast bias is less accurate than the normal bias, resulting in a less accurate clock frequency. The normal, high-accuracy bias is always used when user code is being run.

During the transition from FIMO to regular IMO, glitches can occur if the frequency selection for the two configurations are not the same. Stated explicitly, at the transition, FASTCLK_IMO_CR[2:0] should match PWRSYS_WAKE_TR1[2:0].

16.3.1.2 Internal Low Speed Oscillator

The ILO produces two primary independent output clocks with no external components and with very low power consumption. These two outputs operate at nominal frequencies of 1 kHz and 100 kHz. The two clocks run independently, are not synchronized to each other, and can be enabled or disabled together or independently. The 1 kHz clock is typically used for a background central timewheel and also for the watchdog timer. The 100 kHz clock can provide a low power system clock. A third 33-kHZ clock output is available — a divide-by-3 of the 100 kHz output.

In addition to the multiplexed output that can enter the clock distribution, the output clocks route to the following functions:

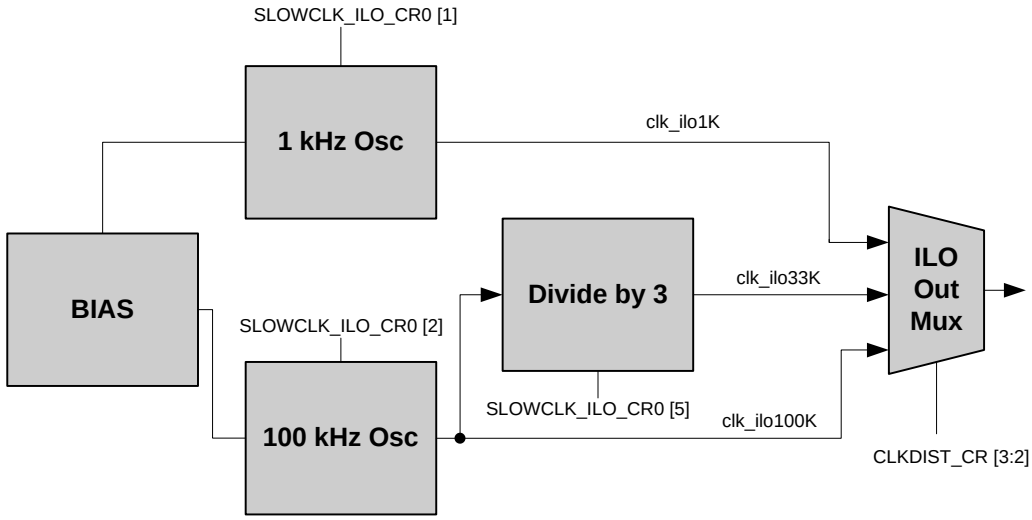
- clk_ilo1K – to the central timewheel (also called the sleep timer) and watchdog timer.
- clk_ilo100K – to the fast timewheel.

This oscillator operates at very low current and is, therefore, the best fit for use in low power modes. The two sources, 1 kHz and 100 kHz, can be enabled and disabled, using the SLOWCLK_ILO_CR0 [1] and SLOWCLK_ILO_CR0 [2], respectively. SLOWCLK_ILO_CR0 [5] enables the divide by 3 to create the 33 kHz output. The out puts from the ILO can be routed to the clock distribution network. CLKDIST_CR [3:2] is responsible for this selection.

Figure 16-2 is a summary block diagram of the ILO. There are dedicated routes for some of the clock outputs that are not shown in the figure.

The ILO clocks are all disabled in the Hibernate mode. SLOWCLK_ILO_CR0 [4] is the power down mode bit governing the wakeup speeds of the device. Setting the bit slows down the startup, but it provides a low power operation.

Figure 16-2. ILO Block Diagram



16.3.2 External Oscillators

PSoC devices have two external crystal oscillators: the MHz Crystal Oscillator (MHzECO) and the 32.768 kHz Crystal Oscillator (kHzECO).

16.3.2.1 MHz Crystal Oscillator

The 4-25 MHz external crystal oscillator MHzECO circuit provides for precision clock signals. The block supports a variety of fundamental mode parallel resonance crystals. When used in conjunction with the on-chip PLL, a wide range of precision clock frequencies can be synthesized, up to 67 MHz.

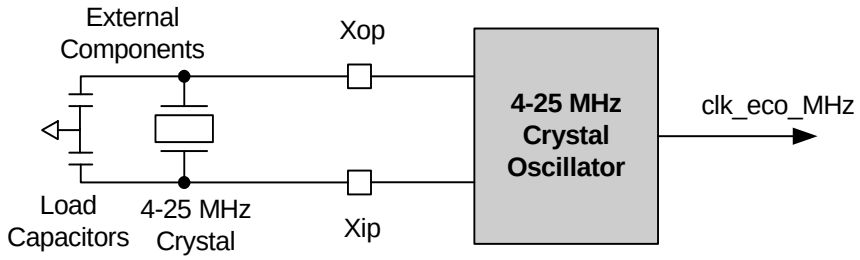
The crystal pins are shared with a standard I/O function (GPIO, LCD, Analog Global), which must be tristated to operate the crystal oscillator with an attached external crystal.

The crystal output routes to the clock distribution network as a clock source option.

The oscillator allows for a wide range of crystal types and frequencies. Startup times vary with frequency and crystal quality. The xcfg bits of the FASTCLK_XMHZ_CFG0 [4:0] register are used to match the oscillator settings to the crystal. The oscillator can be enabled by FASTCLK_XMHZ_CSR [0].

Figure 16-3 is a block diagram of the MHzECO.

Figure 16-3. MHzECO Block Diagram



Low Power Operation

The MHz crystal oscillator does not operate in the SLEEP and HIBERNATE modes. This means that you need to disable the oscillator to enter SLEEP and HIBERNATE modes. If the MHz crystal oscillator is not disabled when the device is put into any of these modes, the mode entry is skipped, and the code continues to execute in active mode. Because this clock must be disabled to enter the SLEEP and HIBERNATE modes, a typical approach is to switch clock trees to the IMO source and then disable the crystal oscillator (and the PLL also, if it is on). Then SLEEP or HIBERNATE mode can be entered. After waking up from a sleep mode, the crystal oscillator can be reenabled and used as a clock source when stable.

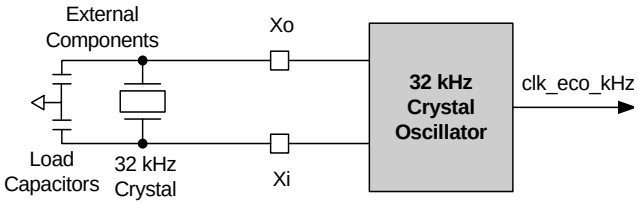
16.3.2.2 32.768 kHz Crystal Oscillator

The 32.768 kHz external crystal oscillator kHzECO circuit produces a precision timing signal at very low power. The circuit uses an inexpensive external 32.768 kHz crystal and associated load capacitors that can be used to produce a real time clock. Current consumption can be much less than 1 µA.

This clock routes to the clock distribution network as an input clock source and also to the RTC timer. This oscillator is one of the clock sources available to the clock distribution logic. The kHzECO is enabled and disabled by the register SLOWCLK_X32_CR [0].

Figure 16-4 is a block diagram of the kHzECO.

Figure 16-4. kHzECO Block Diagram



Low Power Operation

The oscillator operates at two power levels, depending on the state of the LPM bit (SLOWCLK_X32_CR [1]) and the device sleep mode status. In Active mode, by default, a hardware interlock forces the oscillator into its high power mode, which consumes 1-2 µA and minimizes sensitivity to noise. If the LPM mode is set for a low power mode, the oscillator goes into Low power only when the device goes to SLEEP/HIBERNATE. If LP_ALLOW (SLOWCLK_X32_CFG [7]) is set, the oscillator enters low power mode immediately when the LPM bit is set.

When enabled, the oscillator requires some time to stabilize. The status bit ANA_STAT (SLOWCLK_X32_CR[5]) uses an

internal analog monitor to measure oscillator amplitude and gets set when oscillation is stable. The oscillator must always be started in high power mode to avoid excessively long startup delays.

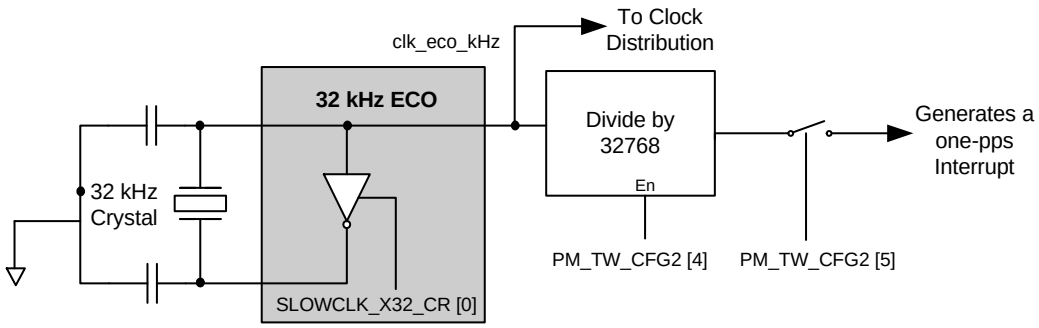
Real Time Clock

One of the major uses of the kHzECO oscillator is for RTC implementation. The block level illustration of the RTC implementation is shown in Figure 16-5.

The RTC timing is derived from the 32 kHz external crystal oscillator, as shown in Figure 16-5. Therefore, for the functioning of the RTC, the 32 kHz external crystal must be enabled through the register SLOWCLK_X32_CR [0]. The generated 32 kHz is divided to achieve a one pulse per second. The register PM_TW_CFG2[4] enables one pulse per second functionality.

By enabling the bit PM_TW_CFG2[5], the RTC generates an interrupt every second. The interrupt is routed through the DSI and is brought out as an interrupt. Refer to the UDB Array and Digital System Interconnect chapter on page 191 for more details about usage. RTC functionality is only available in the active mode.

Figure 16-5. RTC Implementation



16.3.3 Oscillator Summary

A summary of PSoC 5 oscillator output frequency ratings is listed in Table 16-1.

Table 16-1. Oscillator Summary

Source	Fmin	Fmax
IMO	3 MHz	48 MHz
ILO	1 kHz	100 kHz
MHzECO	4 MHz	25 MHz
kHzECO	32.768 kHz	
PLL	24 MHz	67 MHz

16.3.4 DSI Clocks

Signals can be routed from the Digital Signal Interconnect (DSI) and used as clocks in the clock trees. The sources include:

- Clocks developed in UDBs
- Off-chip clocks routed through pins
- Clock outputs from the clock distribution; fed directly back into the network through the routing fabric

16.3.5 Phase-Locked Loop

The on-chip Phase-Locked Loop (PLL) can be used to boost the clock frequency of the selected clock input (IMO, MHzECO, and DSI clock) to run the device at maximum operating frequency. The PLL can synthesize clock frequencies in the range of 24 to 67 MHz. Its input and feedback dividers allow fine enough resolution to create many system clock frequencies. The PLL output routes to the clock distribution network as one of the possible input sources. The PLL is shown in Figure 16-6.

The PLL uses a 4-bit input divider Q (FASTCLK_PLL_Q) on the reference clock and an 8-bit feedback divider P (FASTCLK_PLL_P). The outputs of these two dividers are compared and locked, resulting in an output frequency that is P/Q times the input reference clock. The P and Q divider settings should be changed only while the PLL is disabled. The PLL achieves frequency lock in less than 250 µs, and provides a bit that shows lock status (FASTCLK_PLL_SR[0]). When lock is achieved, the PLL output clock can be routed into the clock trees. Note that when a PLL parameter is changed, it takes four bus clock cycles for the corresponding status to be reflected in the FASTCLK_PLL_SR[0] status bit. This delay must be incorporated in the firmware before reading the status bit.

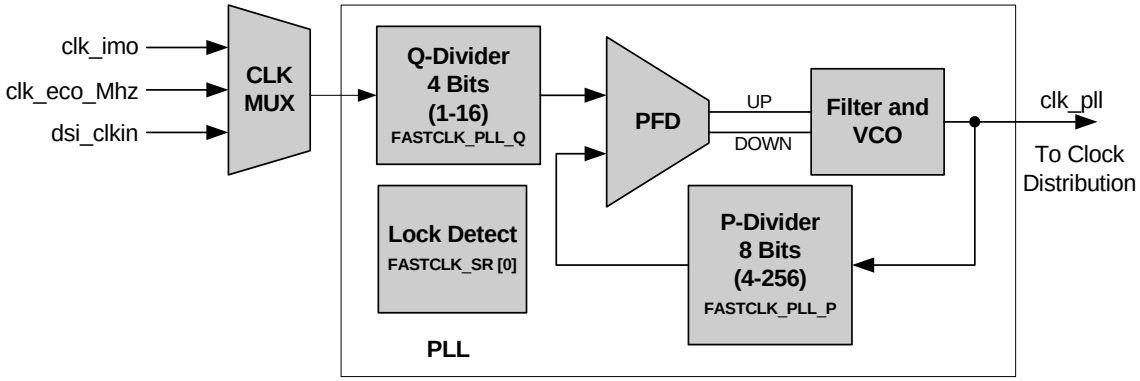
The PLL's charge pump current (Icp) can be configured using bits 6:4 of register FASTCLK_PLL_CFG1. This bit-field should be set to 0x01 for all configurations.

The PLL takes inputs from the IMO, the crystal oscillator MHzECO, or the DSI, which can be an external clock.

Low Power Operation

The PLL must be disabled before going into SLEEP/HIBERNATE mode. This allows clean entry into SLEEP/HIBERNATE and wakeup. The PLL can be reenabled after wakeup and when it is locked; then it can be used as a system clock. The device is designed not to go into SLEEP/HIBERNATE mode if the PLL is enabled when mode entry is attempted. (Execution continues without entering SLEEP/HIBERNATE mode in this case.)

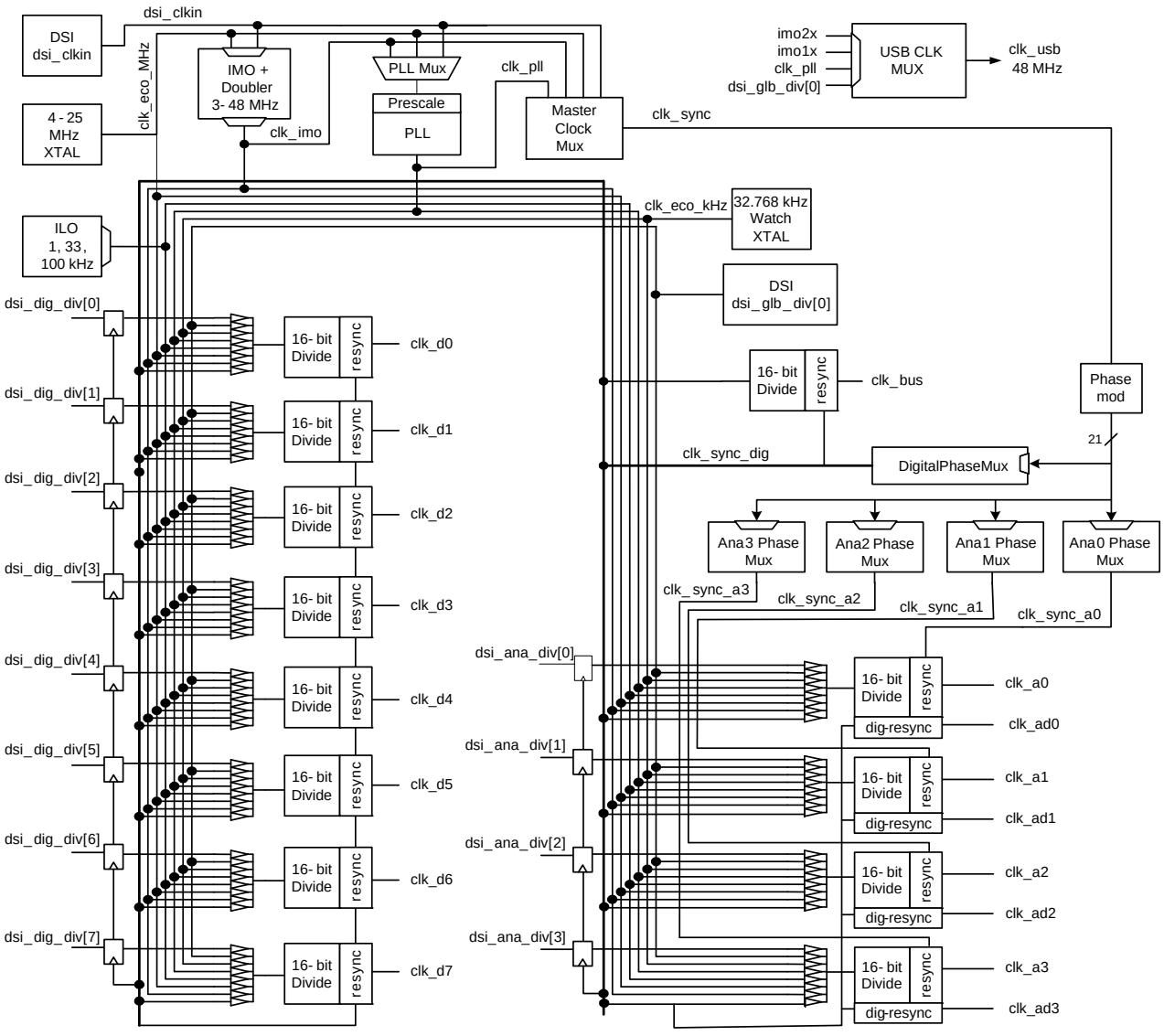
Figure 16-6. PLL Block Diagram



16.4 Clock Distribution

All the clock sources discussed are distributed into the various domains of the device through clock distribution logic. Figure 16-7 shows a block diagram of the clock distribution system.

Figure 16-7. Clock Distribution System



All the clocks available in the device are routed across the device through digital and analog clock dividers. There are certain peripherals that require specific clock source for its operation. For example, Watchdog Timer (WDT) requires Internal Low Speed Oscillator (ILO). In such cases, the corresponding clock source is directly routed to the peripheral.

The clock distribution can be considered to be a combination of the following clock trees.

- System clocks
- Digital clocks
- Analog clocks
- USB clock

The clock distribution provides a set of eight dividers for the digital clock tree and four analog clock dividers for the analog clock tree. All of the clock sources come as input options for all of the clock dividers through 8-input mux. Also, the divider outputs are synchronized to their respective domain clocks.

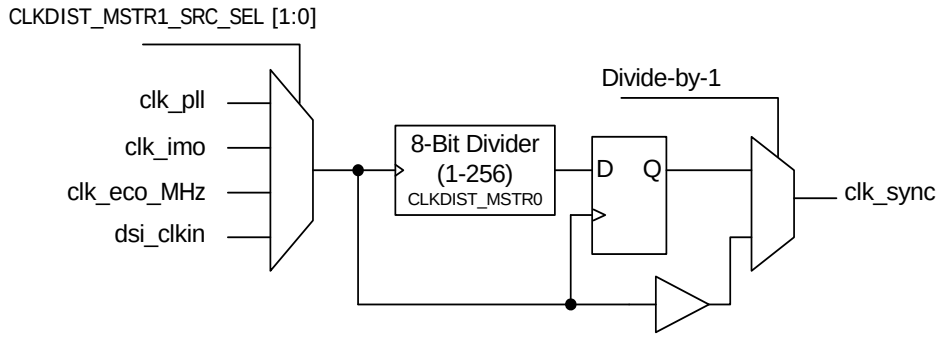
A Master Clock Mux is available for distributing the sync clocks. There are options to provide delay on the digital sync clock. All eight digital dividers are synchronized to the same digital clock, but each of the analog clock divider outputs can be synchronized to analog clocks of different delays. The clock distribution also is responsible for the generation of the major clock domains in the device, such as the system clock, bus clock, and others.

16.4.1 Master Clock Mux

The Master Clock Mux, shown in Figure 16-8, selects one clock from among the PLL, selected IMO output, the MHz crystal oscillator, and the DSI input (dsi_clkin). This clock source feeds the phase mod circuit to produce skewed clocks that are selected by the digital and analog phase mux blocks. The Master Clock Mux provides the re-sync clocks for the network: clk_sync_dig and the analog system clocks, clk_sync_a. The master clock must be configured to be the fastest clock in the system. The master clock also provides a mechanism for switching the clock source for multiple clock trees instantaneously, while maintaining clock alignments. For systems that must maintain known clock relationships, clock trees select the clk_sync_dig (or clk_sync_a*) clock as their input source.

Therefore, when the source is changed (for example, when moving from the IMO source initially to a new PLL- synthesized frequency), all clocks change together through the Master Clock Mux output. The Master Clock Mux contains an 8-bit divider to generate lower frequency clocks, (CLKDIST_MSTR0[7:0]). It outputs an approximately 50% clock.

Figure 16-8. Master Clock Mux

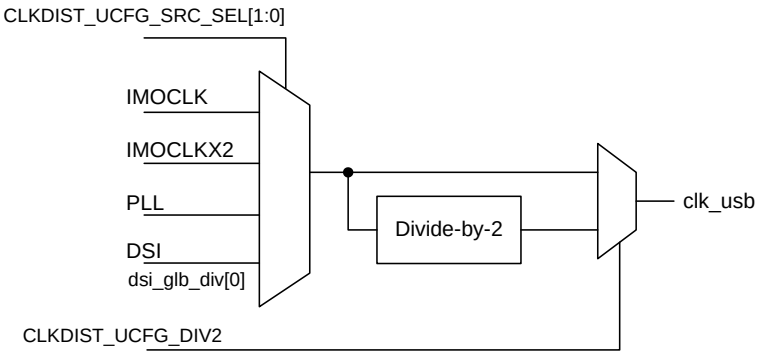


16.4.2 USB Clock

The USB clock domain is unique because it can operate largely asynchronously from the main clock network. The USB logic contains a synchronous bus interface to the device while being able to run on a potentially asynchronous clock to process USB data. For full speed USB, the clock must have an accuracy of ±0.25%.

The USB Clock Mux, shown in Figure 16-9, provides the clock to the USB logic.

Figure 16-9. USB Clock Mux



The USB clock mux selects the USB clock from these clock sources.

- imo1x (these options are available inside the IMO block):
  - 48 MHz DSI clock subjected to the accuracy of the source of the clock
- imo2x (these options are available inside the IMO block):
  - 24 MHz crystal with doubler
  - 24 MHz IMO with doubler with USB lock
  - 24 MHz DSI input with doubler
- clk_pll:
  - Crystal with PLL to generate 48 MHz
  - IMO with PLL to generate 48 MHz
  - DSI input with PLL to generate 48 MHz
- DSI input:
  - 48 MHz

In this situation, any of the choices can produce a valid 48 MHz clock for the USB. If the internal main oscillator is selected, it must be run with the oscillator locking function enabled, in which case it self tunes to the required USB accuracy when USB traffic arrives at the device.

USB Mode Operation

This device works with an automatic clock frequency locking circuit for USB operation. This design allows small frequency adjustments based on measurements of the incoming USB timing (frame markers) versus the IMO clock rate. With this clock locking loop, the clock frequency can stay within spec for the USB Full Speed mode (±0.25% accurate). The IMO must be operated at 24 MHz for proper clock locking, with the doubler supplying 48 MHz for USB logic.

The USB locking feature for the IMO can be enabled by the register bit FASTCLK_IMO_CR [6].

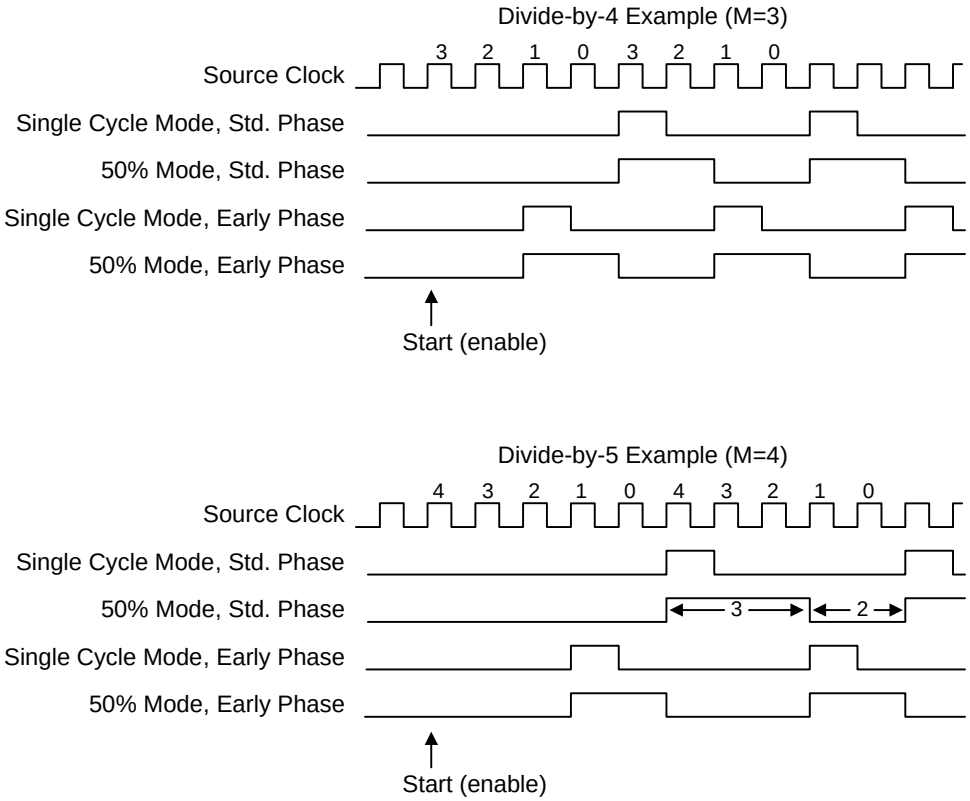
Alternately, a 24 MHz crystal controlled clock doubled to 48 MHz can be supplied for Full Speed USB operation. Other crystal frequencies, such as 4 MHz can be used with the PLL to synthesize the necessary 48 MHz.

Valid frequency for the PLL output, in this case, is 48 MHz. The DSI signal, dsi_glb_div [0], provides another DSI signal choice in addition to the clk_imo option above. As with the PLL, this clock must have USB accuracy and be 48 MHz.

16.4.3 Clock Dividers

Clock dividers form the main part of the clock distribution module and are used to divide and synchronize clock domains. Various clock sources and divider modes may be used together to generate many frequencies with some control over the duty cycle, as depicted in Figure 16-10.

Figure 16-10. Divider Implementation



The divider automatically reloads its divide count after reaching the terminal count of zero. The divider count is set in the register CLKDIST_DCFG[0..7]_CFG0/1 for digital dividers and CLKDIST_ACFG[0..3]_CFG0/1 for analog dividers.The counter is driven by the clock source selected from an 8-input mux, and the source selection is done in the register CLKDIST_DCFG[0..7]_CFG2[2:0] for digital dividers and CLKDIST_ACFG[0..7]_CFG2[2:0] for analog dividers. There are two divider output modes: single-cycle pulse and 50% duty cycle.

In either output mode, a divide value of 0 causes the divider to be bypassed, giving a divide by 1. In this case, the input clock is passed to the output after a resync, if the sync option is selected (see Clock Synchronization on page 101).

For a load value of M, the total period of the output clock is N = M + 1 cycles (of the selected input clock). For example, a load value of 4 gives a 5-cycle long output clock period.

Divider outputs can each be configured to give one of four waveforms, as described below.

16.4.3.1 Single Cycle Pulse Mode

In Single Cycle Pulse mode, by default, the divider generates a single high pulse clock at either the cycle after the terminal (zero) count or the half-count, and is otherwise low. This produces an output clock that is high for one cycle of the input clock, resulting in a 1-of-N duty cycle clock. This is illustrated in Figure 16-10.

16.4.3.2 50% Duty Cycle Mode

In 50% Duty Cycle mode, the output produces a clock that has an approximate 50% duty cycle, depending on whether the total number of counter cycles is even or odd. The 50% clock rising edge occurs at the equivalent rising edge location of the 1/N clock.

For a count of M, there are N = M + 1 input clock cycles in the divider period. If M is odd, the total cycle count N is

even, allowing for a nominal 50% duty cycle. The clock is high for the first (M + 1)/2 cycles, and then goes low for the remaining (M + 1)/2 cycles.

If M is even, the total cycle count is odd, which means that the output clock is high longer than it is low (in standard phase mode). Specifically, it is high for the first (M/2) + 1 cycles and then low for the remaining M/2 cycles. This is illustrated in Figure 16-10 on page 100 for M = 3 and M = 4.

The CLKDIST_DCFG[x]_CFG2[4] or CLKDIST_ACFG[x]_CFG2[4] bit in the configuration register for each clock output can be set high to provide the 50% duty cycle mode. An exact 50% duty cycle cannot be guaranteed in all cases, as it depends on the phase and frequency differences between the output clock and the sync clock.

16.4.3.3 Early Phase Option

In addition to the two duty cycle choices, the outputs can be phase shifted to either go high after the terminal count, or at the half-period cycle. The default is referred to as Standard Phase, with the rising edge of the output after the terminal count.

The other option is referred to as the Early Phase because the output can be considered to be shifted earlier in time to an approximate count that is one-half of the divide value. The CLKDIST_DCFG_CFG2 [5] or CLKDIST_ACFG_CFG2 [5] bit in the configuration register for each clock output can be set high to give the Early Phase mode, with the rising edge near the half count.

Analog clock dividers are similar in their architecture to digital dividers. However, they have an extra resync circuit to synchronize the analog clock to the digital domain clocks. Therefore, each of the analog dividers also has an output synchronized with the digital domain. This clock is synchronized to the output of the digital phase mux. The digital synchronized analog divider output is called clk_ad. This divider is useful for clean communication between analog and digital domain.

16.4.4 Clock Synchronization

All digital and analog divider outputs have an option to be synchronized to the clk_sync_dig signals (CLKDIST_DCFG [x]_CFG2[3] or CLKDIST_ACFG[x]_CFG2[3]), as shown in Figure 16-11.

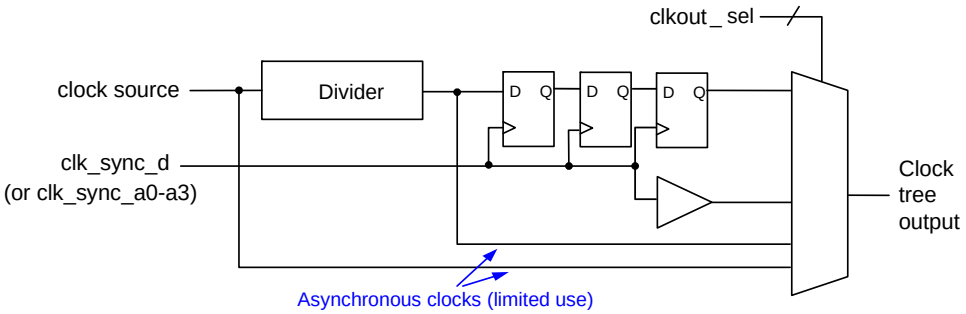
Each digital divider can be synchronized to the digital phase mux output by setting the sync bit (CLKDIST_DCFG [x]_CFG2[3]). The phase delay for the digital divider is based on the phase shift field of Nonvolatile Latch (NVL) bits DIG_PHS_DLY[3:0].

Each of the four analog dividers can be synchronized to four distinct phase shifted clocks. The phase on the respective analog dividers sync clocks can be provided in the PHASE_DLY field (CLKDIST_ACFG[x]_CFG3[3:0]). The analog clocks become synchronized when the SYNC bit is set (CLKDIST_ACFG[x]_CFG2[3]). These divided clocks synchronized to the analog clocks are called clk_a.

The output of each clock tree provides for selection of one of four output clocks:

- **Resynchronized clock** – A clock running at a maximum rate of clk_sync/2 is resynchronized by the phase delayed clk_sync. This output is activated by setting the sync bit.
- **Phase delayed clk_sync (such as clk_sync_dig)** – The clock tree runs at the same rate as clk_sync, but just outputs this clock with proper phase delay. Note that the input clock source is ignored in this case. The output buffer is designed to match the final sync flop delay.
- **Unsynchronized divided clock** – This produces an asynchronous clock, subject to the limitations described in Asynchronous Clocks on page 103. This mode is applicable when the sync bit is reset and the divider has a nonzero divide value.
- **Bypassed clock source** – This routes the clock trees selected source to the output without going through the divider. This happens when the divider value is set to 0 and sync bit is reset. As in the previous case, this also produces an asynchronous clock.

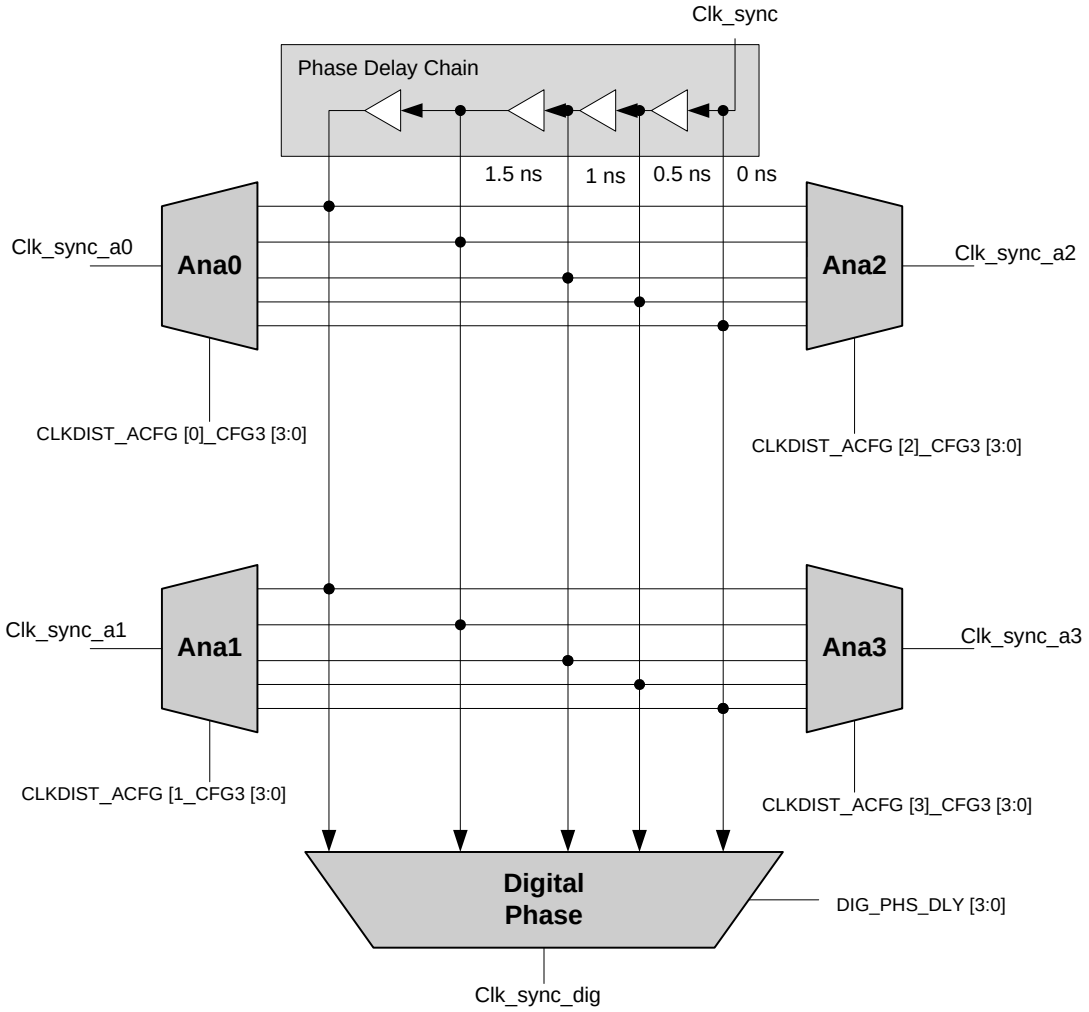
Figure 16-11. Resync Option Diagram



16.4.5 Phase Selection and Control

To keep the environment quiet in the analog processing domain, a phase difference must exist between the analog and digital system clocks. For this reason, in PSoC devices, a delay chain circuit provides taps to control the phase for the digital and analog clocks. This delay chain provides up to a 10 ns phase adjustment with nominal steps of 0.5 ns. The phase shifter is shown in Figure 16-12.

Figure 16-12. Phase Shifter



The phase shifter consists of a chain of (nominally) 0.5 ns buffers connected in cascade, with the output of each buffer ported out of the circuit (21 outputs). The input to this chain is clk_sync from the master clock divider. Five 5-bit muxes select the sync clock to drive the resync circuits. One is clk_sync_dig for the digital clock dividers (clk_bus and all digital clock dividers). The other four are independent delay selections, one for each analog divider. The selected phase value is defined in NVL bits for the digital and ACFG [n]_CLKDIST_ACFG_CFG3}_PHASE_DLY for the analog clocks.

The clk_sync_dig phase shift selection must be applied at power up through NVL settings, because changing its value can cause clock glitching; the clk_bus clock should not be stopped for such a change. The analog phase shift selections can be made dynamically, because their output clocks can be disabled during any phase shift change.

Outputs in the delay chain may have increased jitter. The expectation is that, in systems that need a low-jitter analog clock, the undelayed output (first tap) is selected because it has the lowest jitter.

16.4.6 Divider Update

To allow clean updates of the dividers while running, and to align the starting point for a group of dividers, a load enable mechanism is provided. When a clock is running, it automatically reloads its count value on the terminal count. If a new value has been loaded during countdown of the counter, this new value is loaded at the end of the count, and the next output clock period uses the new value. Because the divide value is 16 bits, there is a possibility that, when updating this register with two 8-bit writes, the full update might not complete when the terminal count occurs. This leads to an unexpected period being reloaded.

To avoid this problem, a 16-bit shadow value (contained in registers {CLKDIST_WRK0*} and {CLKDIST_WRK1*}) allows atomic loads of the dividers, so the 16-bit dividers can be safely updated dynamically (while running). The shadow value can be loaded with two separate 8-bit operations.

The mask registers ({CLKDIST_DMASK*} and {CLKDIST_AMASK*}) allow the user to select the target dividers for this shadow value. When the load bit, {CLKDIST_LD}_LOAD, register is written with a 1, all dividers selected in the mask registers have their period count updated to the shadow value. (If the divider is not enabled, it is safe to do partial writes directly to the divider period register without using the shadow register.)

To align clocks, the mask registers are used again, but this time, they select dividers for auto-alignment. When the {CLKDIST_LD}_SYNC_EN bit register is written with a 1, all dividers selected in mask registers start (or re-start) together. If the dividers are already enabled, they immediately reload and continue counting from this value. If they are not enabled, writing the SYNC_EN bit also sets any corresponding enable bits in the divider enable registers ({PM_ACT_CFG*}), and the dividers begin counting.

Writing a 1 to both of the {CLKDIST_LD}_LOAD and {CLKDIST_LD}_SYNC_EN bits can combine these two operations. This causes all selected dividers to load the shadow register value into their count value, to set all selected divider register enables (if not already enabled), and then to start (or restart) with this setting. The sync loading feature is not supported for clocks that are asynchronous to clk_bus. For instance, an external clock coming from the DSI that is not generated from clk_bus cannot have its divide value changed on the fly reliably. Glitching or transient improper divider loads may occur in this scenario.

16.4.7 Power Gating of Clock Outputs

Clock trees may be gated off (disabled). These gating signals come from the power manager, which contains a register, {PM_ACT_CFG1, PM_ACT_CFG2*}, to allow user selection of trees to enable or disable.

When a clock tree is disabled, its divider is reset so that when reenabled, it reloads its count value. That is, the divider counters do not pause and hold their counts when disabled; they always start over with the latest configured divide count when reenabled.

16.4.8 System Clock

The System Clock is derived from the clk_sync_dig, which is a phase shifted version of clk_sync. The System Clock, also named clk_bus, is the clock that drives the PHUB and associated bus logic. This must be the fastest synchronous clock that outputs to the system. There is an option for a 16-bit divider on the clk_sync_dig to generate the clk_bus CLKDIST_BCFG1/2. This also has the same resynchronization options as the other digital dividers.

16.4.9 Asynchronous Clocks

Generally, all clocks used in the device must be derived from the same source, or synchronized to the main clk_sync clock. However there are possible exceptions:

- A signal that comes on-chip routes through a GPIO, routes to the UDB array, interacts only with self-contained UDB functions, and routes out of the device.
- Similar to the previous, but the signal routes to the interrupt controller instead of off-chip. The interrupt controller is able to handle arbitrarily phased events.
- USB operation with the IMO locking to USB traffic. Although unlikely, in this case, the rest of the device may run off of a different clock, because the USB circuitry contains its own clk_bus synchronous interface, even if its USB clock is not synchronous.

16.5 Low Power Mode Operation

During sleep modes, clock network outputs are gated off, and most clock sources are disabled automatically by the power manager. The low frequency (kHz) clocks may still run, and various clocks are configured by the power manager to support wakeup mode. Refer to the Low Power Modes chapter on page 111 for more details.

The system will not go into a sleep mode if either the MHz crystal oscillator or the PLL are enabled. If either of these clocks are enabled, the part will simply continue execution without entering a sleep mode. Therefore, to enter a sleep

mode when using either the MHz crystal oscillator or PLL, the user must configure the part to run from the IMO and then disable those clock sources. When entering and exiting low power modes, the IMO should be set to 12 MHz with a post divide of 1. To achieve robust clocking into and out of sleep and hibernate modes, the clocks and clock dividers must be sequenced in firmware. This will also meet the wake up time specifications. PSoC Creator provides APIs to do this sequencing both before entering and after exiting low power modes.

16.6 Clock Naming Summary

Table 16-2 lists clock signals and their descriptions.

Table 16-2. Clock Signals

Clock Signal	Description
clk_sync_d	Synchronization clock from the Master clock mux used to synchronize the dividers in the distribution
dsi_clkin	Clocks that are taken as input into the clock distribution from DSI
clk_bus	Bus clock for all peripherals
clk_d[0:7]	Output clock from the seven digital dividers
clk_ad[0:3]	Output clock from the four analog dividers synchronized to the digital domain clock
clk_a[0:3]	Output clock from the four analog dividers synchronized to the analog synchronization clock
clk_usb	Clock for USB block
clk_imo2x	Output of the doubler in the IMO block
clk_imo	IMO output clock
clk_ilo1k	1 kHz output from ILO
clk_ilo100k	100 kHz output from ILO
clk_ilo33k	33 kHz output from ILO
clk_eco_kHz	32.768 kHz output from the kHz ECO
clk_eco_MHz	4-25 MHz output of the MHz ECO
clk_pll	PLL output
dsi_glb_div	DSI global clock source to USB block

17. Power Supply and Monitoring

PSoC® 5 devices have separate external analog and digital supply pins, labeled respectively Vdda and Vddd. The devices have two internal 1.8 V regulators that provide the digital (Vccd) and analog (Vcca) supplies for the internal core logic. The output pins of the regulators (Vccd and Vcca) have very specific capacitor requirements that are listed in the data sheet.

17.1 Features

These regulators are available:

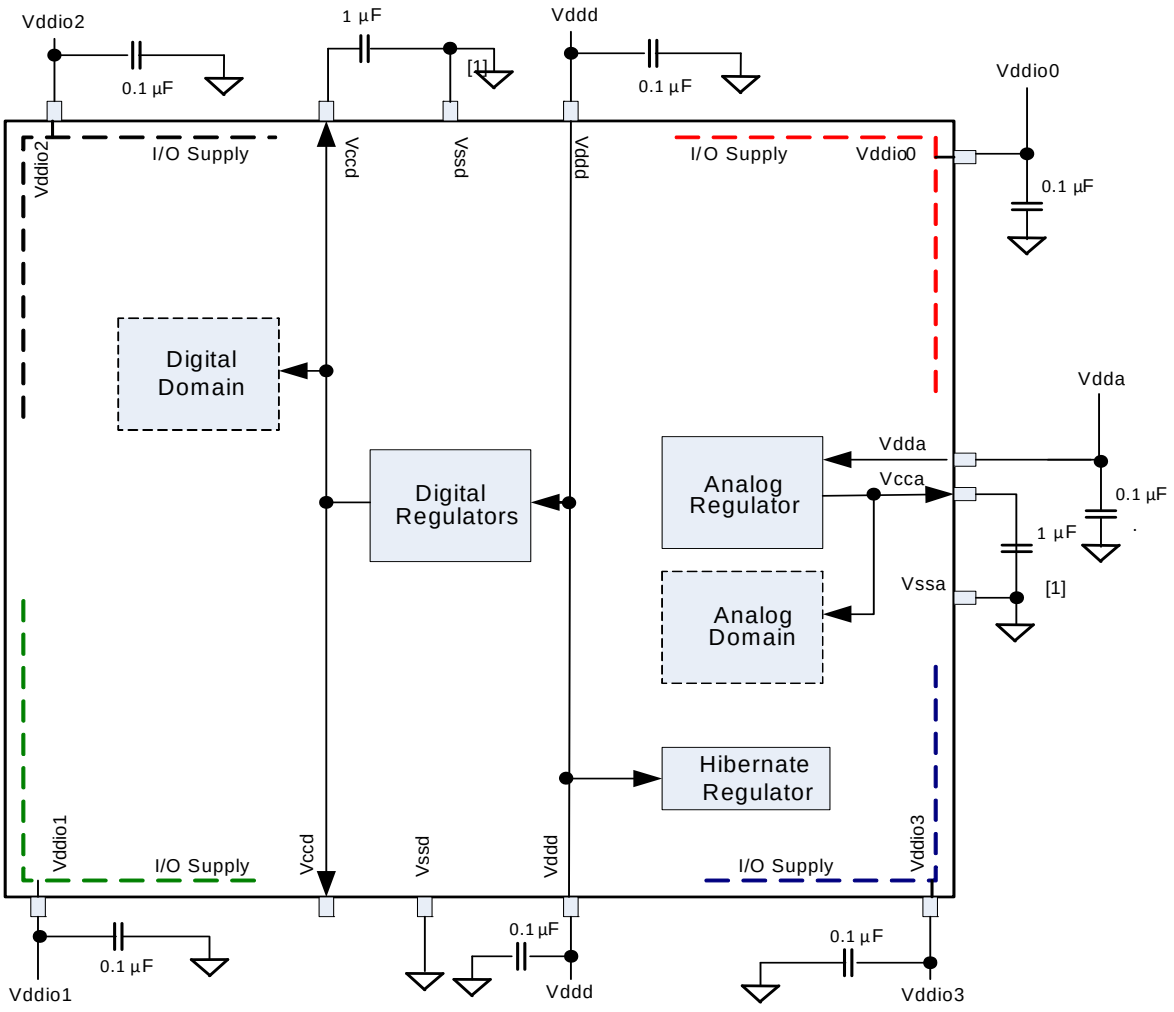
- Analog regulator for analog domain supply
- Digital regulator for digital domain supply
- Hibernate regulator to supply keep alive power for state retention during hibernate mode

17.2 Block Diagram

The power system consists of separate analog, digital, and I/O supply pins, labeled Vdda, Vddd, and Vddiox, respectively. It also includes two internal 1.8 V regulators that provide the digital (Vccd) and analog (Vcca) supplies for the internal core logic. The output pins of the regulators (Vccd and Vcca) and the Vddio pins must have capacitors connected as shown in Figure 17-1. The two VCCD pins must be shorted together, with as short a trace as possible. If the application involves using the sleep mode of PSoC 5, then a 10-µF capacitor should be present on the regulator output pins instead of the 1-µF capacitor. This is explained in Figure 17-1 and the footnote 1. The power system also contains a hibernate regulator.

Vdda must be greater than or equal to all other power supply pins (Vddd, Vddios) in PSoC 5. This power supply condition is required for the proper ON/OFF condition of the analog switches inside the device, and also for the implementation of the internal level switching logic when signals transition between multiple supply voltage domains.

Figure 17-1. Power Domain Block Diagram



Note The two Vccd pins must be connected together with as short a trace as possible. A trace under the device is recommended.

1. For sleep mode, an external capacitor of 10 µF must be connected to the Vccd and Vcca pins.

17.3 How It Works

The regulators shown in Figure 17-1 on page 106 power the various domains of the device. All regulators, except the analog regulator, draw their input power from the Vddd pin supply.

17.3.1 Power Supply Sequencing and Dependencies

The customer must follow some restrictions regarding power supply sequencing with the externally generated supplies.

- Vddd and Vdda must be brought up in synchronization with each other, that is, at the same rates and levels.
- Vdda and Vddd should be within 200 mV and 5 µs of each other during the power ramp.
- Vdda supply must be greater than or equal to all other supplies (Vddd, Vddiox). The Vddiox supply voltage must be greater than the maximum voltage on the associated GPIO pins.
- Maximum voltage on GPIO pin ≤ Vddiox ≤ Vdda.

17.3.2 Regulator Summary

Digital and analog regulators are active during the active or alternate device active modes. They go into a low power mode of operation in sleep or hibernate mode. The hibernate regulator is designed to fulfill power requirements in the hibernate mode of the device. In sleep mode, the combination of periodic wakeup based on central timewheel (CTW) and the 10-µF regulator output capacitors ensures that the regulator outputs are within the operating range.

17.3.2.1 Internal Regulators

The internal regulators are powered by external power supply (2.7 V to 5.5 V) and the supply is provided through the Vddd / Vdda pins. An external capacitor of ~1 µF is connected to the Vccd and Vcca pins. For sleep mode, an external capacitor of 10 µF must be connected to the Vccd and Vcca pins.

17.3.2.2 Hibernate Regulator

The Hibernate regulator, whose output is called Keep Alive power (V_{pwrKA}), powers domains of the device responsible for the state retention in hibernate mode. The V_{pwrKA} is shorted to the active domain during active mode.

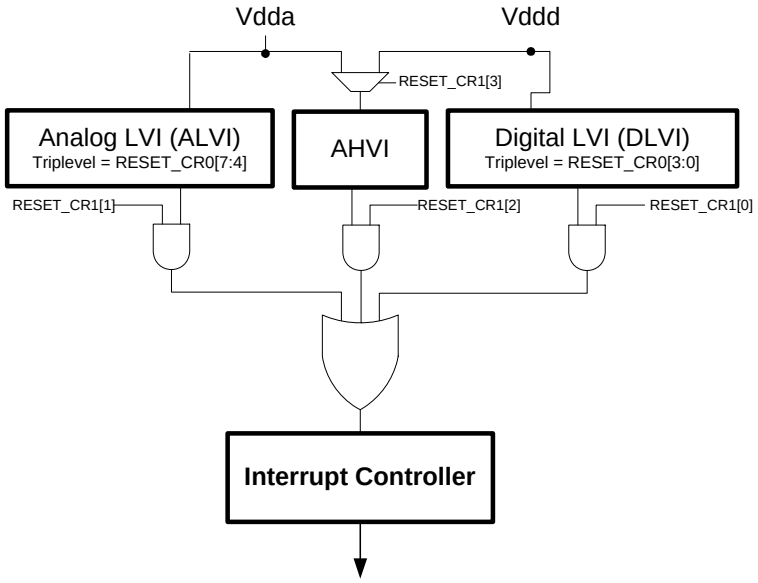
17.3.3 Voltage Monitoring

The device has two circuits to detect voltages that deviate from the selected threshold on the external digital / analog supplies:

- **Low Voltage Interrupt (LVI)** – The LVI circuit generates an interrupt when it detects a voltage below the set value.
- **High Voltage Interrupt (HVI)** – The HVI circuit generates an interrupt when it detects a voltage above the set value.

The basic block diagram of voltage monitoring is shown in Figure 17-2.

Figure 17-2. Voltage Monitoring Block Diagram



17.3.3.1 Low Voltage Interrupt

The LVI circuit generates an interrupt when it detects a voltage below the set value. These low voltage monitors are off by default, but the trip level for the LVI can be set in the register RESET_CRO from 2.45 V to 5.45 V in steps of 250 mV.

The LVI circuit has a persistent status register bit in RESET_SR0 that is set until cleared by the user by reading from the register or until a POR. This bit is set whenever the voltage goes below the set value. There is distinct monitoring for low voltage on the analog and digital supply. The analog low voltage interrupt (ALVI), enabled by RESET_CR1[1] and RESET_CR0[7:4], sets the ALVI threshold. The digital low voltage interrupt (DLVI), enabled by RESET_CR1[0] and RESET_CR0[3:0], sets the DLVI threshold.

The interrupt is generated only when the corresponding bit in register RESET_CR1 is set and corresponding bits in RESET_CR3[7:6] cleared. Even if the interrupt output is not used to generate a processor interrupt, the status registers are updated by the circuit whenever LVI functions are enabled. In addition, the real-time status of each LVI circuit is available and captured in a real-time status register bit in RESET_SR2, so that the user can determine if an under / over voltage condition is still in effect.

The monitors are not available in low-power modes. To monitor voltages in sleep mode, wake up periodically using the CTW. After wakeup, the 2.45 V LVI interrupt may trigger. Voltage monitoring is not available in hibernate mode.

17.3.3.2 High Voltage Interrupt

The HVI circuit generates an interrupt when it detects a voltage above the fixed, safe operating value of 5.75 V on the external analog supply. There is just one HVI for both analog and digital supplies. The selection between monitoring the digital or analog supply is done by the RESET_CR1[3] bit. These high voltage monitors are off by default, but this feature can be enabled in the register RESET_CR1 [2].

The HVI circuit has a persistent status register bit in RESET_SR0 that is set until it is cleared by the user by reading or writing to the register or until a POR reset. This bit is set when the analog voltage value goes beyond the threshold value.

The interrupt is generated only when the corresponding bit in the register RESET_CR1 [2] is unmasked. Even if the interrupt output is not used to generate a processor interrupt, the status registers are updated by the circuit whenever HVI functions are enabled. In addition the real-time output of each HVI circuit is available and captured in a real-time register bit in RESET_SR2, so that the user can determine if an overvoltage condition is still in effect. Similar to the LVI voltage monitors, the HVI monitors are available only in active mode.

17.3.3.3 Processing a Low/High Voltage Detect Interrupt

Both low and high voltage interrupt circuits (LVI, HVI) cause the same interrupt output signal, which is made available to the Interrupt Controller.

Further execution of the interrupt depends on the enable status for the interrupt line in the Interrupt Controller. After the interrupt occurs, the user code can interrogate status registers to determine which LVI or HVI circuit detected an under or over voltage condition.

The actual interrupt output (LVD) is an OR function of the three persistent status register bits corresponding to LVI-D, LVI-A, and HVI. Therefore, to clear the interrupt, the ISR must clear these three register bits.

The LVI and HVI circuits in PSoC 5 generate a glitch when they are enabled. This may cause an unintended interrupt when LVI/HVI is enabled. The following workaround must be done in firmware for this glitch condition.

1. Enable the LVI/HVI circuits by appropriately setting bits [2:0] of the RESET_CR1 register. When this enabling is done, a glitch will be generated.
2. Continuously read the RESET_SR0 register until bits [2:0] of the register are zero. This will ensure the effect of glitch is overcome.
3. Set the voltage thresholds for the LVI circuit by writing appropriate values to the RESET_CR0 register.

LVI/HVI interrupts are now configured to generate an interrupt. The status register RESET_SR0 must be read in the interrupt service routine because LVI/HVI are level

interrupts that must be cleared by reading the status register.

17.4 Register Summary

Table 17-1. Power Supply Register Summary

Register	Function
PWRSYS_CR0	Regulator control
PWRSYS_CR1	Analog regulator control
RESET_CR0	LVI trip value setting
RESET_CR1	Voltage monitoring control
RESET_SR0	voltage monitoring status
RESET_SR2	Real-time voltage monitoring status

18. Low Power Modes

The PSoC® 5 devices feature a set of four power modes with a goal of reducing the average power consumption of the device.

18.1 Features

The PSoC 5 power mode features, in order of decreasing power consumption, are:

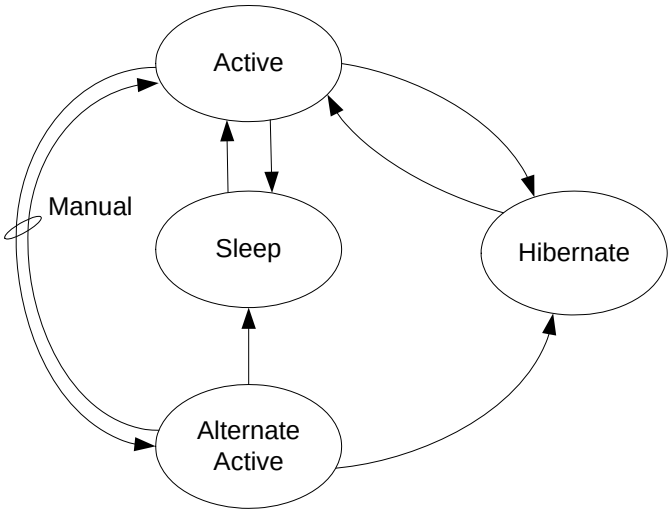
- Active
- Alternative Active
- Sleep
- Hibernate

Active and alternative active are the main processing modes, and the list of enabled peripherals is programmable for each mode.

Sleep and Hibernate modes are used when processing is not necessary for an extended time. All subsystems are automatically disabled in these two modes, regardless of the settings in the active template register. Some subsystems have an additional available bit [PM_Avail_CRx] that can mark a subsystem as unused and prevent it from waking back up. This reduces the power overhead of waking up the part, in that not all subsystems are repowered.

The allowable transitions between power modes are illustrated in Figure 18-1.

Figure 18-1. State Diagram of Allowable Power Mode Transitions



The various power modes reduce power by affecting the following resources:

- Regulators for the digital and analog supply in the device
- Clocks such as the IMO, ILO, and External crystal oscillator (kHzECO, ECOM)
- Central Processing Unit (CPU) and all other peripherals

Power savings, resume time, and supported wakeup sources depend on the particular mode. The four global power-reducing modes are described in Table 18-1 and are listed in decreasing order of power consumption.

Table 18-1. Power Modes

Power Modes	Description	Entry Condition	Wakeup Source	Active Clocks	Regulator
Active	Primary mode of operation, all peripherals available (programmable)	Wakeup, reset, manual register entry	Any interrupt	Any (programmable)	All regulators available.
Alternate Active	Similar to Active mode, and is typically configured to have fewer peripherals active to reduce power. One possible configuration is to use the UDBs for processing, with the CPU turned off	Manual register entry	Any interrupt	Any (programmable)	All regulators available.
Sleep	All subsystems automatically disabled	Manual register entry	CTW	ILO	All regulators available.
Hibernate	All subsystems automatically disabled Lowest power consuming mode with all peripherals and internal regulators disabled, except hibernate regulator is enabled Configuration and memory contents retained	Manual register entry			Only hibernate regulator active.

18.2 Active Mode

Active mode is the primary power mode of the PSoC device. This mode provides the user with an option to use every possible subsystem/peripheral in the device. All of the clocks in the device are available for use in this mode.

Each power-controllable subsystem is enabled or disabled in Active mode, using the active power configuration template bits [PM_ACT_CFGx registers]. This is a set of 14 registers in which each bit is allocated to enable/disable a distinct power controllable subsystem. When a subsystem is disabled, the clocks are gated and/or analog bias currents are reduced.

Firmware may be used to dynamically enable or disable subsystems by setting or clearing bits in the active configuration template. It is possible for the CPU to disable itself, while the rest of the system remains in Active mode. The CPU Active mode bit is not sticky; therefore the CPU is always awakened whenever the system returns to Active mode.

18.2.1 Entering Active Mode

Any wakeup event, any reset, or writing 0 into PM_MODE_CSR [2:0] register while in alternate active mode transitions the device into active mode. When a wakeup event occurs in alternate active/sleep/hibernate mode, the global mode always returns to active and the CPU is automatically enabled, regardless of its template settings. Active mode is the default global power mode upon boot.

18.2.2 Exiting Active Mode

A register write into PM_MODE_CSR [2:0] can transition to another mode. Firmware must ensure the SPC Idle bit in the SPC_SR[1] register is '1' prior to writing to the PM_MODE_CSR [2:0] register to ensure any SPC commands have completed. Any pending wakeup source prevents the device from exiting Active mode.

18.3 Alternative Active Mode

Alternative active mode is similar to active mode in most of its functionality. Alternative active mode also has its own additional set of subsystem template bits

[PM_STBY_CFGx], which determine whether a subsystem is enabled or disabled. This mode is made available for quick transitions between Active and an alternate low power mode.

For example, the user can write to the template bits to disable CPU and enable certain peripherals to operate in alternate active mode. While in alternate active mode, if any interrupt is generated, the device automatically transitions to active mode and begins executing the firmware in active mode.

18.3.1 Entering Alternative Active Mode

Alternative Active mode is entered by writing into [PM_MODE_CSR]. Firmware must ensure the SPC Idle bit in the SPC_SR[1] register is '1' prior to writing to the PM_MODE_CSR [2:0] register to ensure any SPC commands have completed.

The essential difference between Active and Alternative Active mode is that the device cannot wake up from Sleep/ Hibernate mode into the Alternative Active mode.

18.3.2 Exiting Alternative Active Mode

Any interrupt or writing the [PM_MODE_CSR] register can return the system to Active mode.

18.4 Sleep Mode

Sleep mode powers down the CPU and other internal circuitry to reduce power consumption. However, supervisory services such as the central timewheel (CTW) remain available in this mode. The device can wake up using CTW or system reset. The wake up time from sleep mode is 125 µs (typical).

If sleep mode is required in the application, then a 10-µF capacitor should be connected on the respective regulator output pins. The two Vccd pins should be shorted externally and there should be a 10-µF capacitor on one of the Vccd pins. The Vcca pins should also have a 10 uf capacitor. See Figure 17-1 and the adjoining footnote for pictorial representation.

18.4.1 Entering Sleep Mode

Sleep mode is entered by writing the appropriate code into PM_MODE_CSR [2:0]. Firmware must ensure the SPC Idle bit in the SPC_SR[1] register is '1' prior to writing to the PM_MODE_CSR [2:0] register to ensure any SPC commands have completed. Entry must be from a state where the CPU is available (active). The system ignores any request to enter sleep mode for the first 1 ms after POR.

Central time wheel (CTW) is the only wake up source from sleep mode. The CTW period must be configured to the required time interval and the CTW interrupt must be enabled before entering sleep mode. The CTW counter should be reset before entering sleep mode to wake up the device after the configured time period. The CTW time period can be from 4 ms to 128 ms.

In PSoC 5, there are other register configurations that should be done before entering sleep mode. These include saving the clock configuration register settings in active mode, configuring the hardware buzz for the maximum duration of 512 ms. Also, other wakeup events other than CTW such as PICU, 1 pulse per second (1PPS) timer events should be disabled before entering sleep mode. They can be re-enabled after exiting from sleep mode if required, to operate in the active mode of the device. The power management functions provided in PSoC Creator implement the above steps before entering sleep mode. It is recommended to use those functions to enter the device low power modes.

When an IRQ is generated simultaneously with sleep request, the CPU halts and prevents further execution. To avoid this, all interrupts should be cleared and disabled before requesting sleep mode.

18.4.2 Exiting Sleep Mode

Wakeup event can come from the central timewheel event. A wakeup event restores the system to active mode. The central timewheel allows the system to periodically wake up, poll peripherals, do voltage monitoring, or perform real-time functions. XRES event can also wake up the device by causing a device reset.

18.5 Hibernate Mode

Hibernate mode consumes/dissipates the lowest power, and nearly all internal functions are disabled. The hibernate-regulator is always active to generate the keep-alive voltage (V_{pwrka}) used to retain the system state. Refer to 17.3.3 Voltage Monitoring on page 107.

Configuration state and all memory contents are preserved in Hibernate mode. GPIOs configured as digital outputs maintain their previous values, and pin interrupt settings are preserved. The voltage used to retain state is lower than the nominal core voltage.

In Hibernate mode, voltage is monitored with a lower degree of precision than in the other power modes. The hibernate mode has a higher probability of having soft errors. Hence for safety critical applications the MFGCFG.PWRSYS.HIB.TR1[7] can be programmed to prevent hibernate

mode. When this bit is asserted, the command to enter Hibernate will put the system into Sleep mode.

To achieve an extremely low current, a hibernate regulator with limited capacity is used. This limits the frequency of any signal present on the input pins - no GPIO should toggle at a rate greater than 10 kHz while in hibernate mode. Because hibernate mode is intended to implement a dormant state in the application, this is not a practical limitation. Any system that has signals toggling at high rates in low power modes can use the sleep mode without seeing a significant difference in total power consumption.

18.5.1 Entering Hibernate Mode

Hibernate mode is entered by a write into PM_MODE_CSR [2:0]. Firmware must ensure the SPC Idle bit in the SPC_SR[1] register is '1' prior to writing to the PM_MODE_CSR [2:0] register to ensure any SPC commands have completed. The extremely low current hibernate regulator requires at least 1 ms to start up after a reset. During this time, the system ignores requests to enter Hibernate mode.

When an IRQ is generated simultaneously with hibernate request, the CPU halts and prevents further execution. To avoid this, all interrupts should be cleared and disabled before requesting sleep mode. If SWD is enabled, it prevents the part from entering sleep mode. To avoid this, the SWD must be disabled before requesting sleep mode.

18.5.2 Exiting Hibernate Mode

Return from Hibernate mode can occur only in response to a reset event on the external reset (XRES) pin. The digital, analog, and sleep regulators are disabled in Hibernate mode. Upon wakeup, the system activates all previously available domains, unless the {PM_MODE_CFG1 [2]} field is set.

18.6 Timewheel

Timers and timewheels schedule events. They can be programmed to generate periodic interrupts for timing or to wake the system from a low power mode.

18.6.1 Central Timewheel (CTW)

The Central Timewheel (CTW) is a 1 kHz, free-running, 13-bit counter clocked by the ILO. The CTW is always available, except in Hibernate mode and when the CPU is stopped during Debug on-Chip (DoC) mode. The main functions of the CTW are:

- Watchdog timer (WDT)
- General timing purposes

CTW settings are programmable, using PM_TW_CFG1[3:0].

Although the CTW is free-running, separate settings are used for the wakeup and watchdog timeouts. The CTW can be programmed, using the {PM_TW_CFG2[2]} registers, to wake the system periodically and optionally issue an interrupt by programming the bit {PM_TW_CFG2[3]}.

18.6.2 Fast Timewheel (FTW)

The Fast Timewheel (FTW) is a 100 kHz, 5-bit counter clocked by the ILO. The FTW settings are programmable, using PM_TW_CFG0 [4:0], and the counter automatically resets when the terminal count is reached. If the associated FTW interrupt is enabled using PM_TW_CFG0 [1], an interrupt is generated each time the terminal count is reached. The FTW can wake up the device only from standby mode. It is not available in sleep mode.

18.7 Register List

Table 18-2. Low Power Modes Register List

Register Name	Description
General Registers	
PM_ACT_CFGx	Active mode template
PM_STBY_CFGx	Alternate Active mode template
PM_AVAIL_CRx	Available settings for limited Active mode transition
PM_AVAIL_SRx	Availability Status register
PM_MODE_CFG0	Not used
PM_MODE_CFG1	Interrupt and settings for low power modes
PM_MODE_CSR	Power Mode Control and Status register

Table 18-2. Low Power Modes Register List *(continued)*

Register Name	Description
PM_INT_SR	Power Mode Interrupt Status register
PM_TW_CFG0	Fast Timewheel (FTW) Configuration register
PM_TW_CFG1	Central Timewheel (CTW) Configuration register
PM_TW_CFG2	Configuration settings for CTW and FTW

19. Watchdog Timer

The Watchdog Timer (WDT) circuit automatically reboots the system in the event of an unexpected execution path. This timer must be serviced periodically. If not, the CPU resets after a specified period of time. When the WDT is enabled, it cannot be disabled except during a reset event. This is done to prevent any errant code from disabling the WDT reset function. To use the WDT function, enable the WDT function during their startup code.

19.1 Features

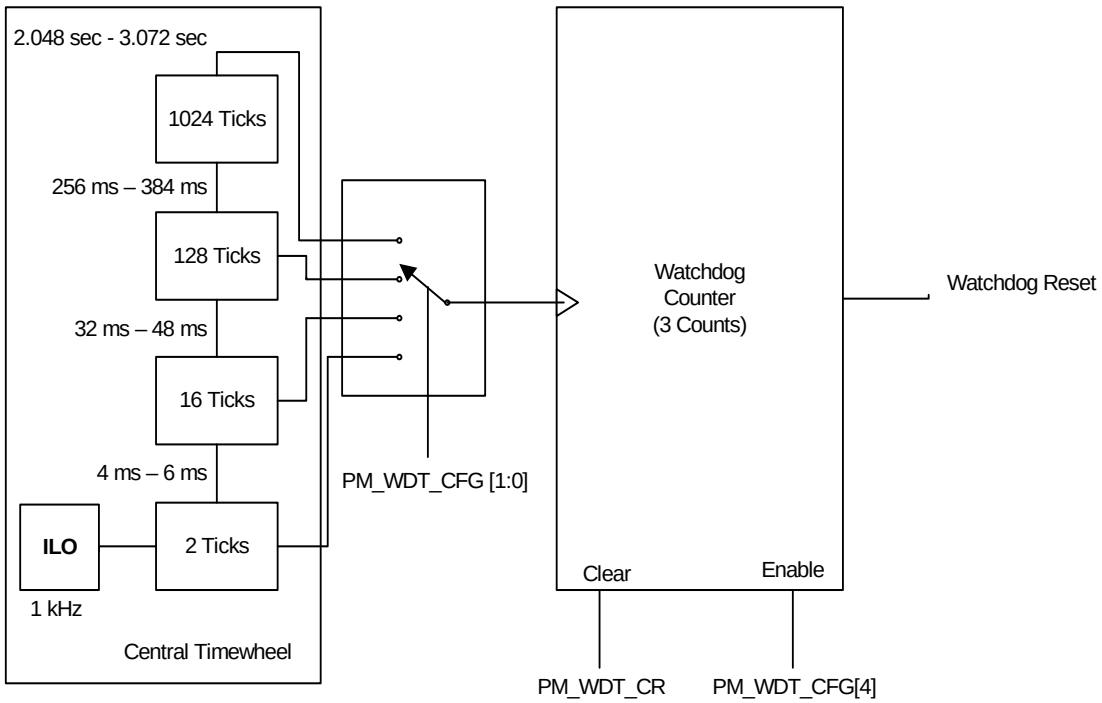
The WDT has the following features:

- Protection settings to prevent accidental corruption of the WDT
- Optionally-protected servicing (feeding) of the WDT
- A status bit for the watchdog event that shows the status even after a watchdog reset

19.2 Block Diagram

Figure 19-1 is a block diagram of the WDT circuit.

Figure 19-1. Watchdog Timer Circuit



19.3 How It Works

The WDT circuit asserts a hardware reset to the device after a pre-programmed interval, unless it is periodically serviced in firmware. If an unexpected execution path is taken through the code and the pre-programmed interval times out, the system is restarted.

The WDT timeout is between two and three programmable tap periods, based on the free-running Central Timewheel. See the *PSoC® 5 Registers TRM (Technical Reference Manual)*.

Each time the central timewheel crosses the programmed tap point, the Watchdog counter increments. When the counter reaches three, a Watchdog reset is asserted, and the counter is reset. When the WDT is serviced in software, the counter is reset to zero.

The time between servicing and the first tap crossing is usually less than the complete tap period; therefore, software should be programmed to service the WDT within two tap periods. Actual WDT timeouts may differ slightly from nominal, caused by inaccuracy of the frequency of the ILO.

IPOR disables the watchdog function. The program must enable the watchdog function at an appropriate point in the code by setting a register bit explained in the next section. When this bit is set, it cannot be cleared again except by an IPOR power on reset event. The watchdog timer can be used only when the part remains in active mode.

19.3.1 Enabling and Disabling the WDT

The WDT is enabled by setting the PM_WDT_CFG [4] register bit. After this bit is set, it cannot be cleared again except by a power reset event. This is done so that errant code cannot accidentally disable the Watchdog.

Users must either reenable the Watchdog function at startup after a reset occurs or include code to reenable the function should a reset occur, allowing a dynamic choice whether to enable the Watchdog.

A status bit (RESET_SR0[3]) is set on the occurrence of a Watchdog reset. This bit remains set until cleared by the user, by reading or writing to the register, or until a POR reset. All other resets leave this bit untouched.

19.3.2 Setting the WDT Time Period and Clearing the WDT

The user can select a tap from the central timewheel using the register PM_WDT_CFG[1:0]. Based on the tap selected, the WDT is timed at various periods, shown in Figure 19-1 on page 117. The Watchdog Timer counts until reaching three counts, based on the tap from the central timewheel. If the firmware does not clear the WDT before this time, a Watchdog reset is initiated.

To prevent an automatic reset, the WDT must be periodically serviced by firmware. In the default mode, this is accomplished by writing any value to the PM_WDT_CR field. It is a good idea to service the WDT in a firmware main loop, that is, not in an interrupt handler. If the WDT is serviced in an interrupt handler, and the main loop code goes astray, the WDT may never generate a reset because the interrupt may still be active, causing the interrupt handler to continue to service the WDT.

For proper operation of the WDT, after clearing WDT immediately write ‘0’ to the WDT_CFG register. The WDT timeout period should be greater than the sleep wakeup period and the WDT must be serviced upon wakeup from sleep.

19.4 Register List

Table 19-1. Reset Register List

Register Name	Comments
PM_WDT_CFG	Configuration register for Watchdog
PM_WDT_CR	Watchdog clear
RESET_SRO	Persistent Status register for Watchdog reset

20. Reset

PSoC® 5 architecture supports several types of resets that allow error-free operation during power up for any voltage ramping profile, user-supplied external or software resets, and recovery from errant code operation.

20.1 Reset Sources

The following is a description of reset sources. For power up supply monitoring, PSoC 5 devices support POR (power on reset). They also support WRES (watchdog reset) for recovery from errant code, and SRES and XRES_N for user-supplied software and external resets, respectively. When a reset is initiated, all registers are restored to their default states with minor exceptions, such as some of the persistent status registers.

20.1.1 Initial Power-on Reset

At initial power on, IPOR monitors the power voltages VDDD and VDDA, both directly at the pins and at the outputs of the corresponding internal regulators. The trip level is not precise. It is set to approximately 1 volt, which is below the lowest specified operating voltage but high enough for the internal circuits to be reset and to hold their reset state. The monitor generates a reset pulse that is at least 100 ns wide. It may be much wider if one or more of the voltages ramps up slowly. To save power, the IPOR circuit is disabled when the internal digital supply is stable. When the voltage is high enough, the IMO starts.

20.1.2 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. The user must always set the WRES initialization code. This is done to allow the user to dynamically choose whether to enable the watchdog timer.

This feature is enabled by setting the PM_WDT_CFG [4] register bit. After this bit is set, it cannot be cleared again except by a reset event. When a watchdog timer event occurs, device reset occurs normally, but the watchdog timer enable bit is not cleared. This scheme allows the watchdog timer enable bit to be a flag available to firmware or software to indicate that a watchdog timer event occurred. See Watchdog Timer chapter on page 117.

The RESET_SR0 [3] status bit becomes set on the occurrence of a watchdog reset. This bit remains set until cleared by the user or until a POR reset. All other resets leave this bit untouched. Except for the status bit, the watchdog reset functions as all other system resets.

20.1.3 Software Initiated Reset

Software Initiated Reset (SRES) is a mechanism that allows a software-driven reset. The RESET_CR2 register forces a device reset when a ‘1’ is written into bit 0. This setting can be made by firmware or with a DMA.

The RESET_SR0 [5] status bit becomes set on the occurrence of a software reset. This bit remains set until cleared by the user or until a POR reset.

20.1.4 External Reset

External Reset (XRES_N) is a user-supplied reset that causes immediate system reset when asserted. XRES_N is available on a dedicated pin on some devices, and on a shared GPIO pin P1[2] on all devices. The shared pin is available through a customer-programmed NV latch setting and supports low pin count parts that don't have a dedicated XRES_N pin. This path is typically configured during the boot phase immediately after power up. See the Nonvolatile Latch chapter on page 79 for more details.

Either the dedicated pin or the GPIO pin, if configured, holds the part in reset while held active. When the pin is released, the part goes through a normal boot sequence. The external reset is active low, so that a low voltage (near ground) on the XRES_N pin causes a reset.

20.1.5 Identifying Reset Sources

When the device comes out of reset, it is beneficial to know the cause of the reset. This is achieved in the device through the registers RESET_SR0 and RESET_SR1.

These two registers have specific status bits allocated for the various RESET sources, except POR and XRES. The bits are set on the occurrence of the corresponding reset, and remain set after the reset, until cleared by the user or a POR reset.

Therefore, all the other RESET sources can be identified after the reset. In the case of POR, the entire register is cleared, indicating a power on reset.

20.2 Reset Diagram

Figure 20-1 is a simplified logic diagram of the RESET module. Any active source of reset will make the System RESET.

Figure 20-1. Logic Diagram of the RESET Module

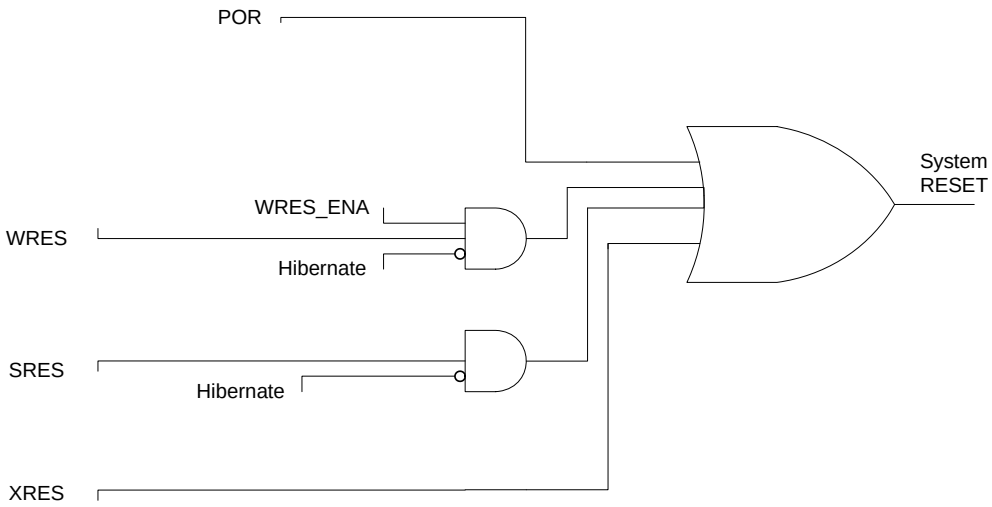
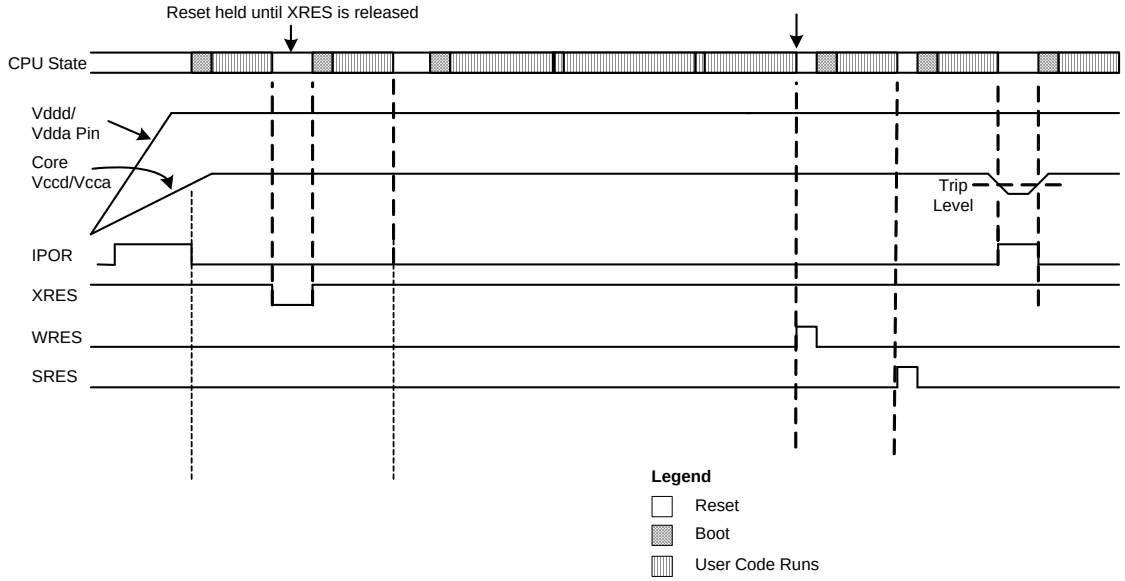


Figure 20-2 shows the operation of various RESETs with the change in V_{dd}/V_{cc}. The diagram also shows the functioning of RESETs in a normal power up.

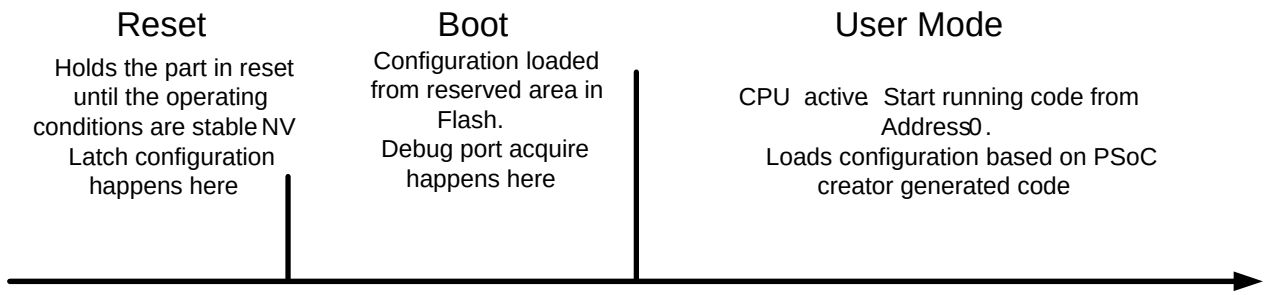
Figure 20-2. Resets Resulting from Various Reset Sources in Active Mode



20.3 Boot Process and Timing

The boot process trims and configures the silicon to its ideal state before the first line of the user code is executed. The PSoC life cycle consists of reset, boot, and user phases. Figure 20-3 gives a brief view of these phases.

Figure 20-3. Boot Process



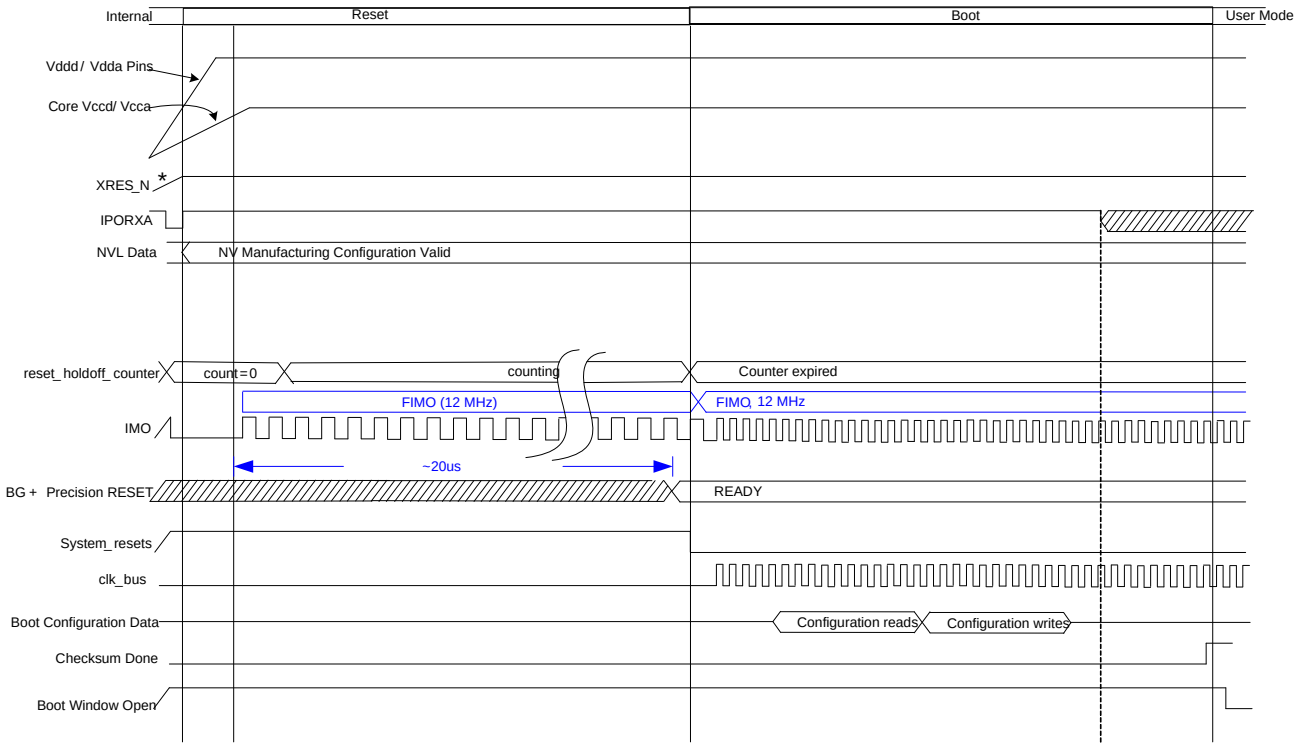
The process from supply voltage stabilization to user code entry is shown in Figure 20-4. After the voltage is high enough, the NVL data load is initiated. The NVL load takes care of loading configuration data stored in the NV latches. These are configuration data that control the reset behavior of the device. The maximum time for this NVL load is 10 µs from the time of initiation. This resets the I/Os to the NVL drive mode settings and sets the other Manufacturing Configuration data for the device. Now, the device enters the reset state. The two types of NVL loads that happen here are explained in section 20.3.1 Manufacturing Configuration NV Latch.

If the external reset pin (XRES_N) is asserted low, the device stays in the reset state. If the external reset pin (XRES_N) is not asserted and all the voltages are at their correct operating values, it triggers the reset hold off circuitry to begin bringing the device out of the reset state.

The IMO clock is then started in a fast IMO (FIMO) mode, which is a faster start up version of the IMO. The reset hold off counter continues to hold the device in reset until the other systems such as band-gap and precision resets stabilize. The length of the hold off is approximately 20 µs to allow enough time for these circuits to stabilize. If the band-gap or precision reset blocks are not ready or there is a problem with any of these devices stabilizing by the end of the hold-off counter, a fresh reset cycle is initiated and the hold-off counter is restarted. If there are no problems and the hold-off counter completes and the device is released from reset.

After releasing from reset, the IMO is switched to 12 MHz, the system bus clock is started, and the boot cycle begins. Until now the bus clock was fed from the FIMO, which has lesser accuracy compared to the IMO. When the reset is released, it moves into the IMO, which is more precise. The boot phase is explained in section 20.3.3 User Mode. During this boot configuration time, if there is no toggling of the external pins P1_0 and P1_1 and the configuration finishes, the system moves into the user mode. Toggling of P1_0 and P1_1 implies a debug port acquire is being attempted, which has to trigger a debug port entry. External resets are not allowed during the boot phase to ensure proper loading of the boot configuration data.

Figure 20-4. Power Up Reset Boot User Mode Cycle



Note* Avoid asserting the reset pin during the boot phase.

In this phase, two types of NV latches are loaded to set reset states and trims in the device. Both the configurations, explained in sections 20.3.1 Manufacturing Configuration NV Latch and 20.3.1.1 Device Configuration NV Latch, occur simultaneously in the reset phase.

20.3.1 Manufacturing Configuration NV Latch

There are some circuits that must receive part specific trim values before the device comes out of reset. Manufacturing NV latches provides these trim values. Conceptually, an example of such a circuit is the power on reset. This circuit is responsible for holding the device in reset until a safe supply voltage is reached. The POR circuit requires a trim value, which is stored in an NV latch. NV latch's output is stable at approximately 1 V while the lowest operating voltage in the PSoC 5 platform is 2.7 V.

20.3.1.1 Device Configuration NV Latch

Device configuration is similar to manufacturing configuration NV in that it occurs while the device is in reset. Manufacturing configuration and device configuration occur in parallel. One such example of a device configuration is the NV latches that determine the I/O drive modes during reset, which determine the reset state of the drive mode registers.

20.3.2 Boot Phase

Though many settings for the device are done using NV latch setting during the preboot process, there are other trim values that require to be written during the boot process. These values are stored in reserved space in the flash memory (I/O System chapter on page 125) and the boot process takes care of moving this data to the corresponding blocks. This loading of the configuration happens using the DMA and the PHUB. A DMA channel fetches the configuration bytes from the flash and places them in the SRAM. The check sum block does a check sum to determine integrity. When the data is verified, it is then transferred using the DMA to the corresponding configuration register. If the check sum fails, it triggers a system reset.

Note that some circuits have mode dependent trim values; for example, the IMO's trim value depends on the speed setting of the IMO. For circuits with mode dependent trim values, the boot process loads the trim value that matches the default mode. When the user's firmware or configuration changes the mode, the firmware also retrieves the correct trim value corresponding to the modes from the tables stored in flash and writes them to the appropriate register.

The CPU halts until boot completes; therefore, you cannot use the CPU to complete the boot process. The PHUB, DMA, and a special checksum block are used for boot to move the manufacturing configuration data from the flash to the appropriate registers. These three blocks work together to accomplish these objectives:

- Minimize boot time, giving you the quickest path to firmware execution
- Provide a data integrity check on the manufacturing configuration data
- Provide flexibility in the order and addresses manufacturing configuration data is written to

20.3.3 User Mode

When the boot phase is complete, the device enters the User mode to enable firmware code execution. This is where code execution starts for the startup/configuration code developed by PSoC Creator. Only after executing this part of the PSoC Creator generated code does the code execution reach the main().

20.4 Register List

Table 20-1. Reset Register List

Register Name	Comments
RESET_CR2	
RESET_SR0	Persistent status bits for WRES, SRES, XRES_N, and so on
RESET_SR1	Persistent status bits for Segment reset
RESET_SR2	Real-time Reset Status

21. I/O System

The I/O system provides the interface between the CPU core and peripheral components to the outside world. The flexibility of PSoC® devices and the capability of its I/O to route any signal to any pin greatly simplifies circuit design and board layout. There are two types of I/O pins on every device, general purpose I/O (GPIO) and special I/O (SIO); those with USB provide a third type. Both GPIO and SIO provide similar digital functionality. The primary differences are their analog capability and drive strength. Devices that include USB also provide two USBIO pins that support specific USB functionality and specialized general purpose capability.

All I/O pins are available for use as digital inputs and outputs for both the CPU and digital peripherals. In addition, all I/O pins can generate an interrupt. All GPIO pins can be used for analog input, CapSense®, and LCD segment drive, while SIO pins are used for voltages in excess of Vdda and for programmable output voltages and input thresholds.

21.1 Features

The PSoC I/O system has these features, depending on the pin type.

- Features supported by both GPIO and SIO:
  - Separate I/O supplies and voltages for up to four groups of I/O
  - Digital peripherals use DSI to connect the pins
  - Input or output or both for CPU and DMA
  - Eight drive modes
  - Every pin can be an interrupt source configured as rising edge, falling edge or both edges. If required, level sensitive interrupts are supported through the DSI
  - Dedicated port interrupt vector for each port
  - Slew rate controlled digital output drive mode
  - Access port control and configuration registers on either port basis or pin basis
  - Separate port read (PS) and write (DR) data registers to avoid read modify write errors
  - Special functionality on a pin by pin basis
- Additional features only provided on the GPIO pins:
  - LCD segment drive on LCD equipped devices
  - CapSense on CapSense equipped devices
  - Analog input and output capability
  - Continuous 100 µA clamp current capability
  - Standard drive strength down to 2.7 V
- Additional features only provided on SIO pins:
  - Higher drive strength than GPIO
  - Hot swap capability (5 V tolerance at any operating VDD)
  - Programmable and regulated high input and output drive levels down to 1.2 V
  - No analog input or LCD capability
  - Over voltage tolerance up to 5.5 V
  - SIO can act as a general purpose analog comparator

- USBIO features:
  - Full speed USB 2.0 I/O
  - Highest drive strength for general purpose use
  - Input, output, or both for CPU and DMA
  - Input, output, or both for digital peripherals
  - Digital output (CMOS) drive mode
  - Each pin can be an interrupt source configured as rising edge, falling edge, or both edges

21.2 Block Diagrams

Figure 21-1, Figure 21-2 on page 127, and Figure 21-3 on page 128 are block diagrams of three main categories of I/Os: GPIO, SIO, and USBIO, respectively. Each diagram emphasizes the main blocks that drive the system, and the signals and register settings that control the main blocks.

Figure 21-1. GPIO Block Diagram

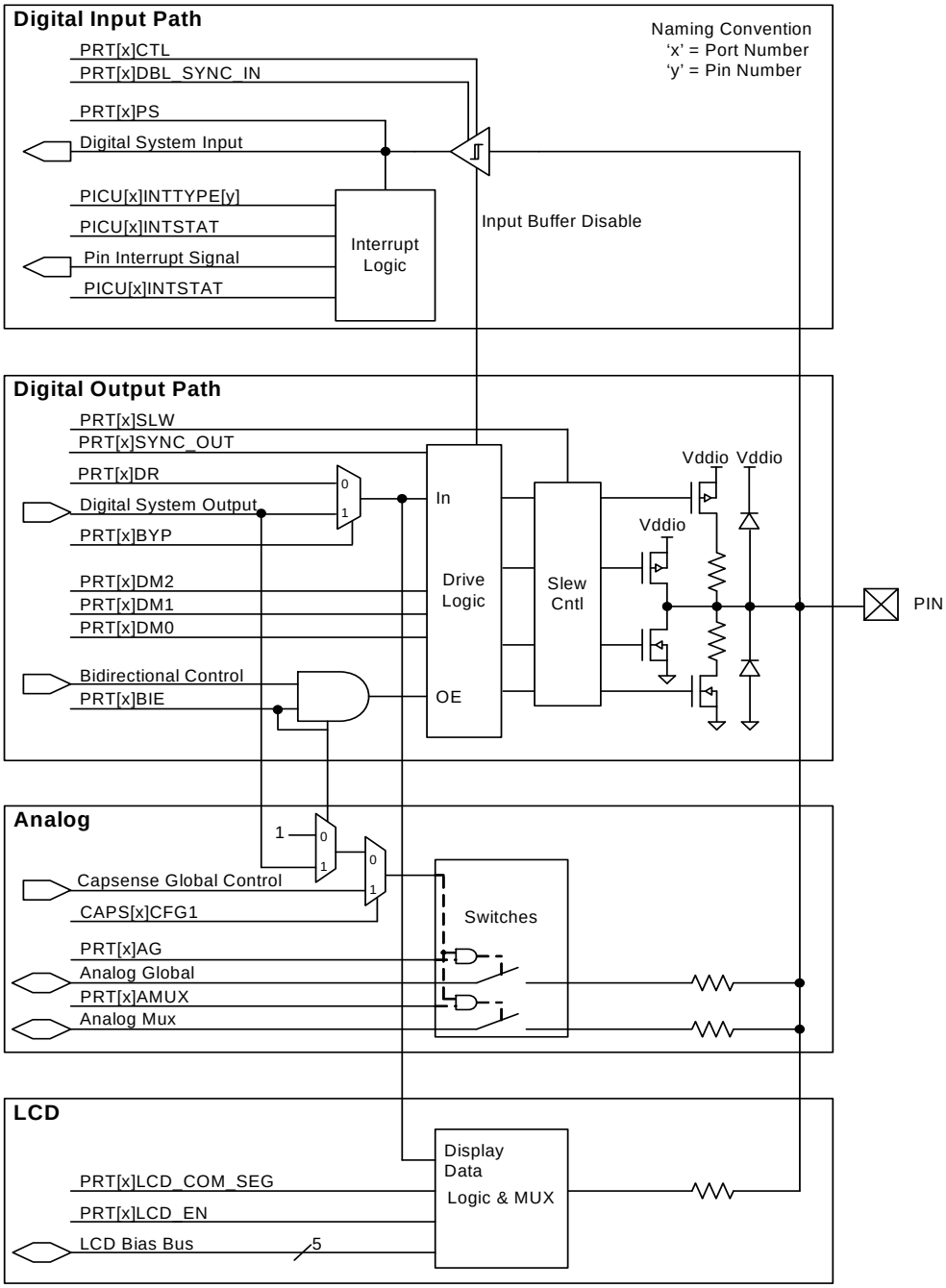


Figure 21-2. SIO Block Diagram

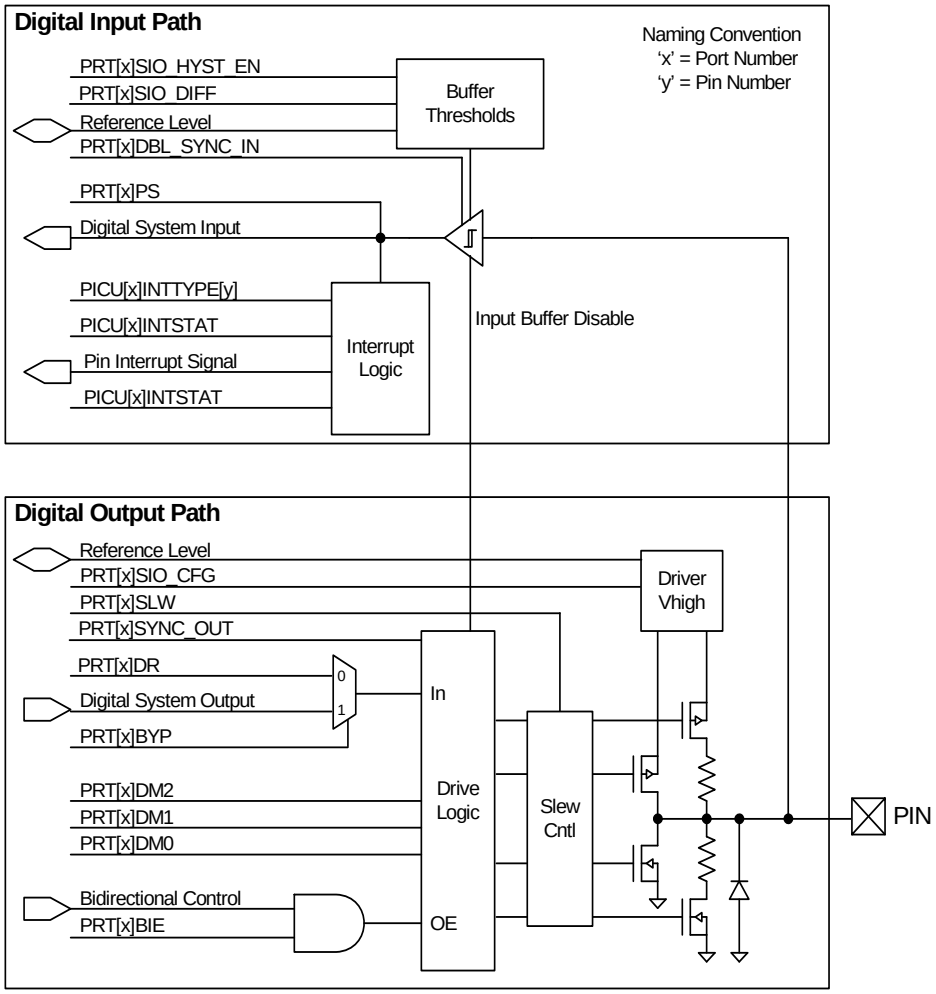
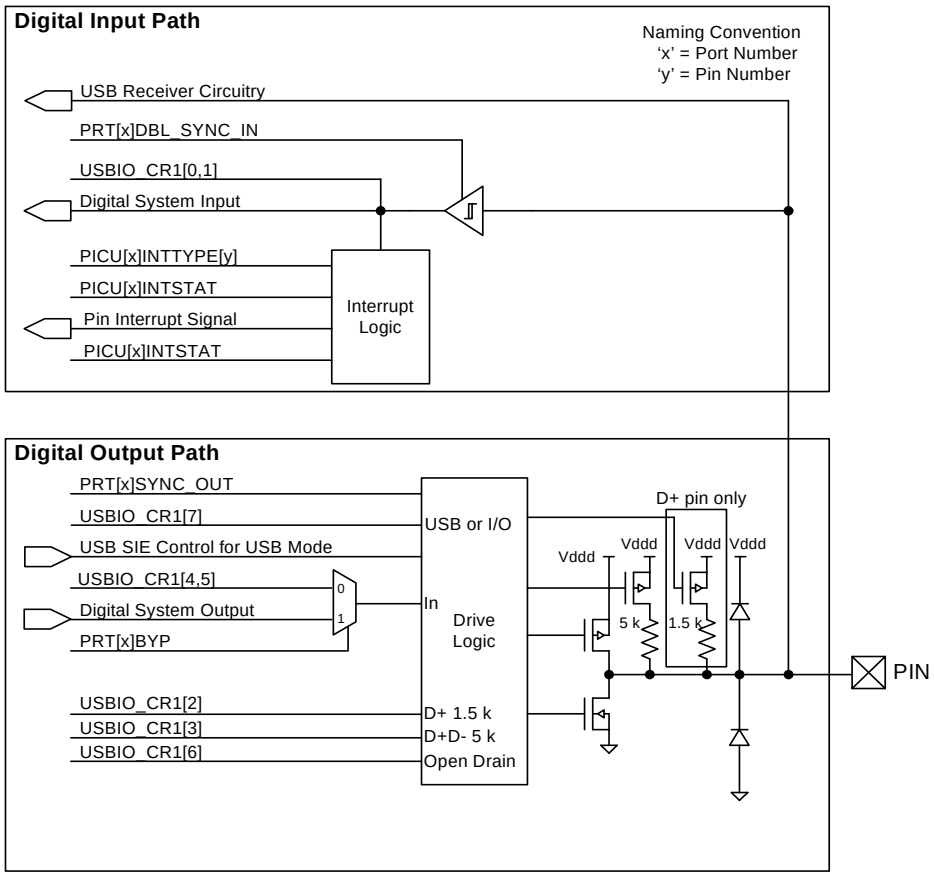


Figure 21-3. USBIO Block Diagram



21.3 How It Works

PSoC I/Os provide:

- Digital input sensing
- Digital output drive
- Pin interrupts
- Connectivity for analog inputs and outputs
- Connectivity for LCD segment drive
- Access to internal peripherals:
  - Directly for defined ports
  - Through the Universal Digital Blocks (UDB) via the Digital System Interconnect (DSI)

The I/Os are arranged into ports, with up to eight pins per port. Some of the I/O pins are multiplexed with special functions (USB, debug port). Special functions are enabled using control registers associated with the specific functions. For example, the Crystal Oscillator control register enables the crystal oscillator function for the I/O pin multiplexed with the crystal oscillator function.

21.3.1 Usage Modes and Configuration

Because of the variety of I/O capabilities, it is necessary to understand the modes thoroughly and the configuration for each function.

21.3.2 I/O Drive Modes

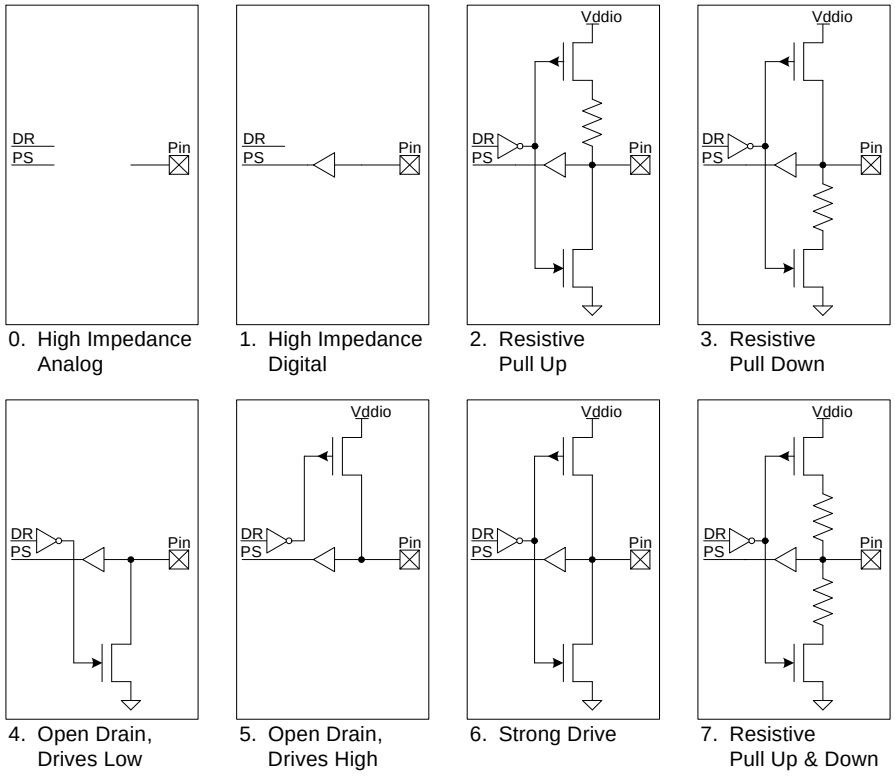
Each GPIO and SIO pin is individually configurable into one of the eight drive modes listed in Table 21-1 and shown in Figure 21-4, which depicts a simplified pin view based on each of the eight drive modes.

The I/O pin drive state is based on the port data register value (DR) or on a DSI signal, if bypass mode is selected. The actual I/O pin voltage is determined by a combination of the DR value, the selected drive mode, and the load at the pin. The state of the pin can be read from the Port Status register (PS) or routed to a DSI signal, or both. Three configuration bits are used for each pin (DM [2:0]) and set in the PRTxDM [2:0] registers.

Table 21-1. I/O Drive Modes

Mode Number	Drive Mode	PRTxDM2 DM2	PRTxDM1 DM1	PRTxDM0 DM0	Data = 1	Data = 0
0	High Impedance Analog	0	0	0	High Z	High Z
1	High Impedance Digital	0	0	1	High Z	High Z
2	Resistive Pull Up	0	1	0	Res 1 (5k)	Strong 0
3	Resistive Pull Down	0	1	1	Strong 1	Res 0 (5k)
4	Open Drain, Drives Low	1	0	0	High Z	Strong 0
5	Open Drain, Drives High	1	0	1	Strong 1	High Z
6	Strong Drive	1	1	0	Strong 1	Strong 0
7	Resistive Pull Up and Down	1	1	1	Res 1 (5k)	Res 0 (5k)

Figure 21-4. I/O Drive Mode Diagram



0. High Impedance Analog
1. High Impedance Digital
2. Resistive Pull Up
3. Resistive Pull Down
4. Open Drain, Drives Low
5. Open Drain, Drives High
6. Strong Drive
7. Resistive Pull Up & Down

21.3.2.1 Drive Mode on Reset

The factory drive mode during device reset is high impedance analog mode. The Reset drive mode is set at POR release. Each pin is individually configured during the device configuration step after POR release; this setting overwrites the reset drive mode.

21.3.2.2 High Impedance Analog

In High Impedance Analog mode, both output driver and digital input buffer are turned off. This state prevents a floating voltage from causing a current to flow into the I/O digital input buffer. This drive mode is recommended for pins that are floating or that support an analog voltage. High impedance analog pins cannot be used for digital inputs. Reading the pin state register returns a 0x00 regardless of the data register value.

To achieve the lowest device current in sleep modes, all I/Os must either be configured to the high impedance analog mode, or they must have their pins driven to a power supply rail (ground) by the PSoC device or by external circuitry.

21.3.2.3 High Impedance Digital

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

21.3.2.4 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 digital input and output in these modes. Interfacing to mechanical switches is a common application for these modes. If a pull up is needed with the Resistive Pull Up Drive mode, a ‘1’ must be written to that pin’s Data Register bit. If a pull down is required with the Resistive Pull Down Drive mode, a ‘0’ must be written to that pin’s Data Register bit.

21.3.2.5 Open Drain, Drives High and Drives Low

Open Drain modes provide high impedance in one of the data states and strong drive in the other. Pins are used for digital input and output in these modes. A common application for these modes is driving I^2C bus signal lines.

21.3.2.6 Strong Drive

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

21.3.2.7 Resistive Pull Up and Pull Down

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

21.3.3 Slew Rate Control

GPIO and SIO pins have fast and slow output slew rate options for strong drive modes – not resistive drive modes. The fast slew rate is for signals between 1 MHz and 33 MHz.

Because it results in reduced EMI, the slow option is recommended for signals that are not speed critical – generally less than 1 MHz. Slew rate is individually configurable for each pin and is set by the PRTxSLW registers.

21.3.4 Digital I/O Controlled by Port Register

The Port Control registers (see Table 21-2 on page 131) have separate configuration bit for each port pins. In addition to port control registers, the device also provides register for port-wide and pin wise configuration.

The port wide configuration register writes the same configuration for all the port pins in a single write. This is useful to configure all the port pins to a specific configuration.

The pin wise configuration register writes to all configuration bits for a specific I/O pin in a single write. This is useful to configure individual port pins to a specific configuration.

Outputs are driven from the CPU by writing to the port data registers (PRTx_DR) Digital inputs are read by the CPU through the pin state registers (PRTx_PS}).

21.3.4.1 Port Configuration Registers

Table 21-2 lists port control registers. Note that DMA cannot be used to write to pin configuration registers; always use CPU to write to these registers.

Table 21-2. Functional Registers Accessed through Pin and Port Configuration Registers

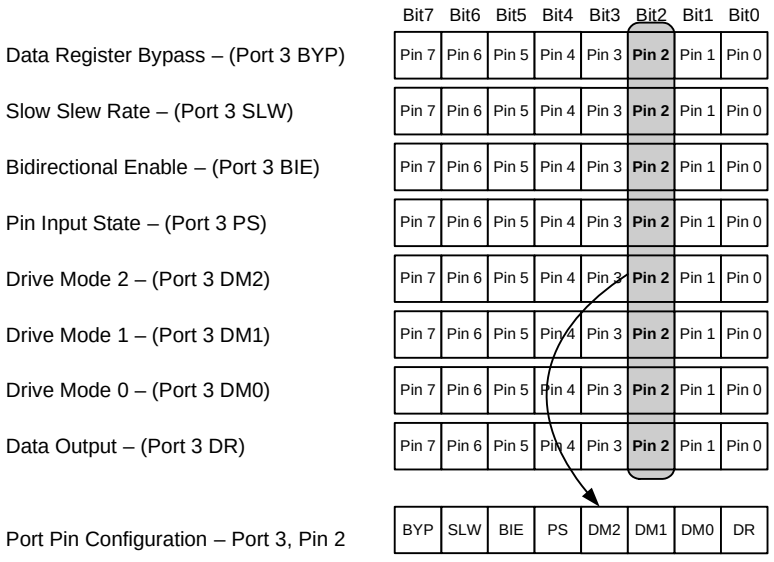
Address	Description
PRT[0..11]_BYP	A bit set in this register connects the corresponding port pin to the Digital System Interconnect (DSI), and disconnects it from the DR register.
PRT[0..11]_SLW	Each bit controls the output edge rate of the corresponding port pin – fast edge rate mode (Slew=0) or slow edge rate mode (Slew=1)
PRT[0..11]_BIE	Each bit set controls the bidirectional mode of the corresponding port pin. 0 = Output always enabled 1 = Output Enable controlled by DSI input
PRT[0..11]_PS	This register reads the logical pin state for the corresponding GPIO port.
PRT[0..11]_DM[0..2]	The combined value of these registers – PRTx_DM2, PRTx_DM1, and PRTx_DM0 – determines the unique drive mode of each pin in a GPIO port.
PRT[0..11]_DR	Data written to this register specifies the high (Data=1) or low (Data=0) state for the GPIO pin at each bit location of the selected port.

21.3.4.2 Pin Wise Configuration Register Alias

The port pin configuration registers (PRTxPC0 through PRTxPC7) access several configuration or status bits of a single I/O port pin at once, as shown in Figure 21-5.

Figure 21-5 shows an example of a read from {PRT*_PC[4]}. Bit four of the port control registers associated with the port configuration register is read and driven onto the data bus.

Figure 21-5. Effect of a Read of the Pin Configuration Register {PRT*_PC[4]}

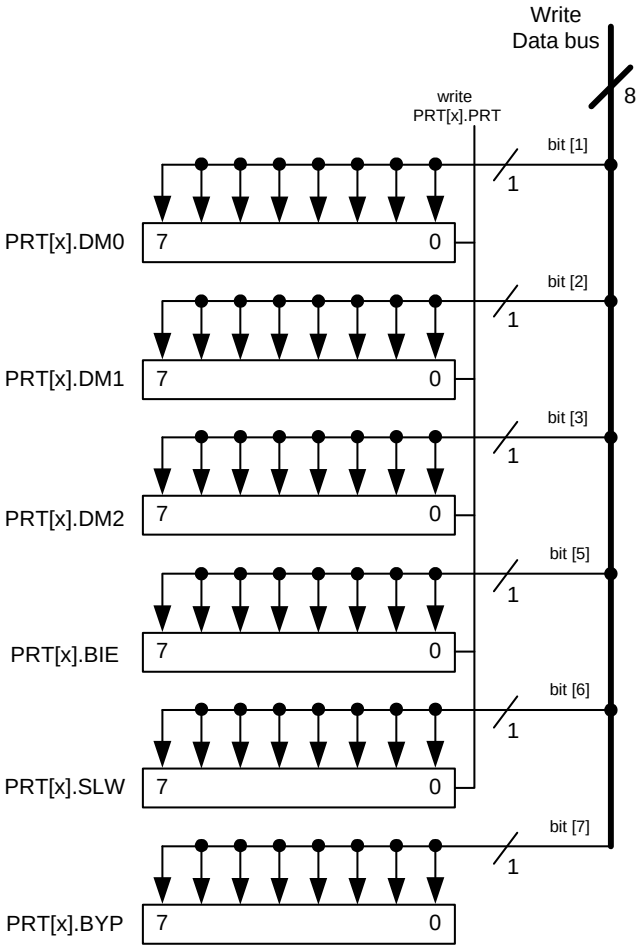


21.3.4.3 Port Wide Configuration Register Alias

The Port Configuration Register accesses several available configuration registers on a port-wide basis with a single bit write.

This register PRT*_PRT aliases a subset of the configuration registers, allowing the user to configure a complete port in a single write.

Figure 21-6. {PRT*.PRT} Write Example



21.3.5 Digital I/O Controlled Through DSI

GPIO, USBIO, and SIO pins are connected to the internal peripheral blocks through the UDB via the digital system interconnect (DSI). Any peripheral connected to the UDB can be connected to any I/O pin through the DSI.

Each port has 20 unique connections to the UDB through DSI: eight inputs, eight outputs, and four output control signals.

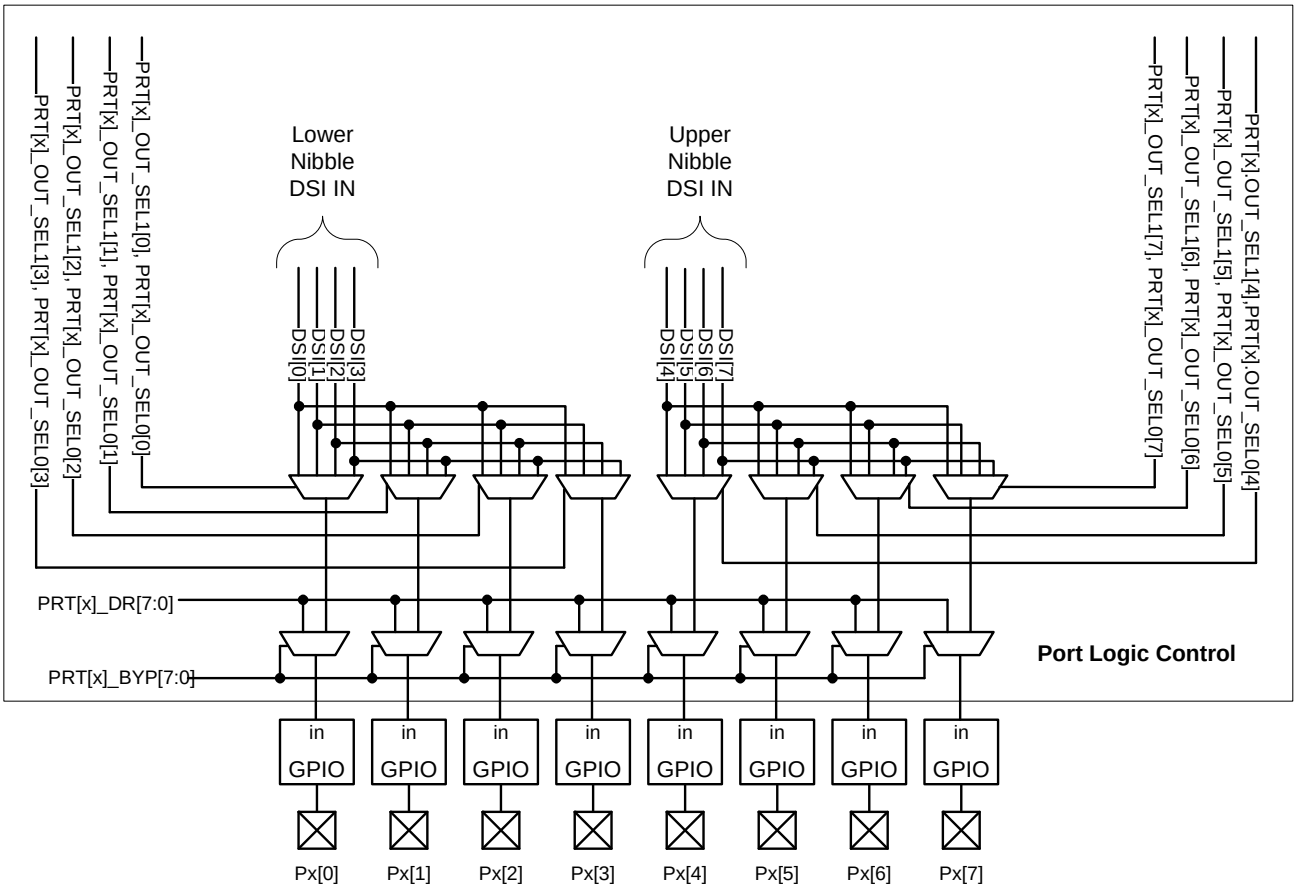
21.3.5.1 DSI Output

The bypass register {PRTx_BYP} selects either the selected DSI output signal or the data register (PRTx_DR) to drive the port pin.

Mapping of the DSI signal to the output pin is illustrated in Figure 21-7 on page 133.

Together, output select registers PRTx_OUT_SEL1 and PRTx_OUT_SEL0 select the DSI output signal to drive the corresponding output port pin.

Figure 21-7. Digital System Input to Pad Selection



21.3.5.2 DSI Input

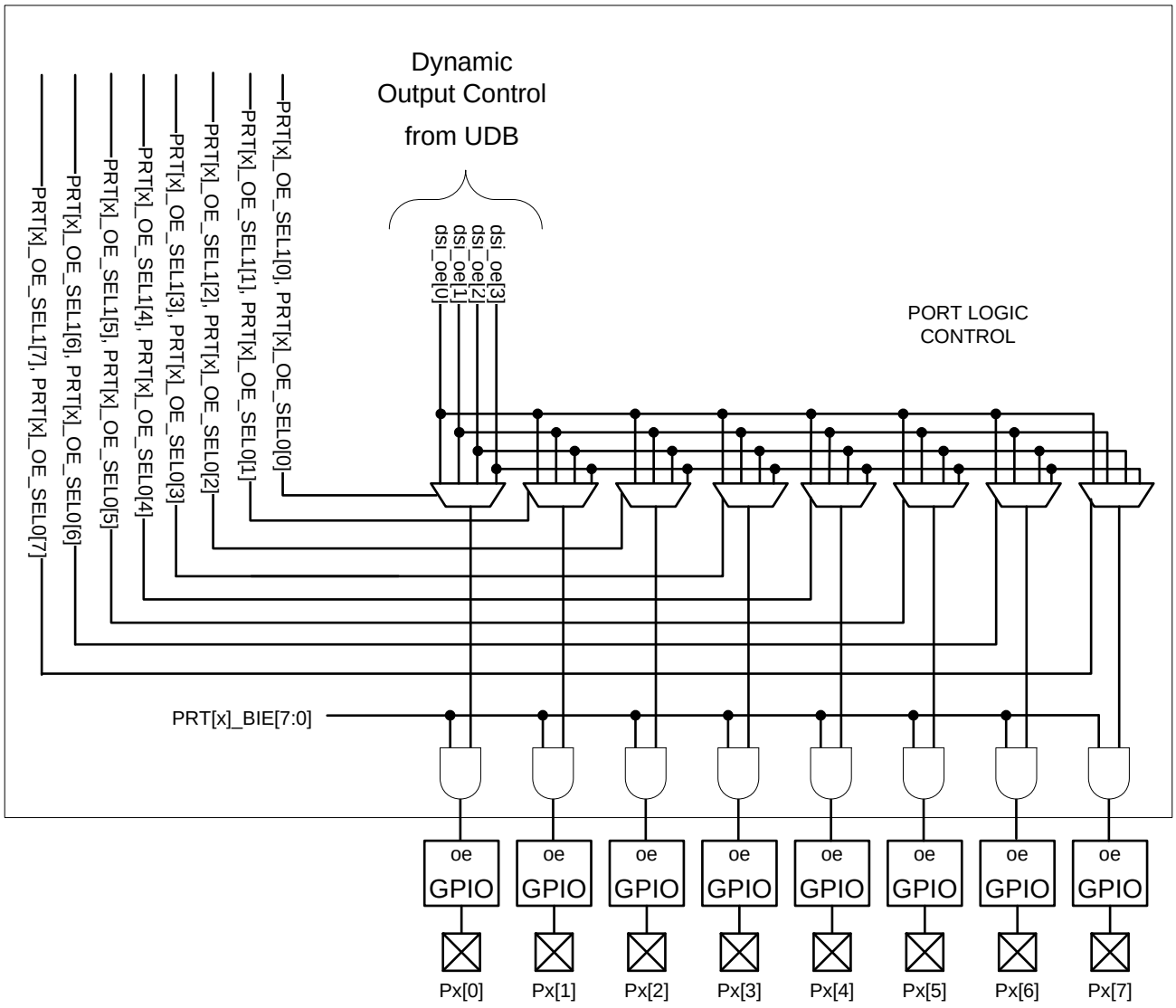
The port pin input is directly connected to the UDB array through DSI for routing the input to various internal peripheral blocks. The control for these port inputs are at the DSI inputs. See the Universal Digital Blocks (UDBs) chapter on page 149 for port-to-DSI connections.

21.3.5.3 DSI for Output Enable Control

High-speed bidirectional capability is provided through the {PRT*_BIE} register. When this mode is enabled and the auxiliary control signal is high, the I/O pin immediately goes into a High Z output drive state with input buffer enabled. When this signal is low (or returns low), the I/O pin assumes the pin state configured through the {PRT*_DM[2]}, {PRT*_DM[1]}, and {PRT*_DM[0]} registers.This allows fast turnaround of the I/O pin. Four DSI control signals are available for dynamic drive control of the pins. Mapping of the DSI control signal to port pin output enable is shown in Figure 21-8 on page 134.

Together, dynamic output enable select registers PRTx_OE_SEL1 and PRTx_OE_SEL0 select the DSI control signal for each port pin.

Figure 21-8. Mapping of DSI Control Signal to Port Pin Output Enable



21.3.6 Analog I/O

The only way that analog signals can pass to and from the PSoC core is through GPIO.

To connect a pin to an internal analog resource through analog global bus or analog mux line, each GPIO connects to one of the analog global lines and to one of the analog mux lines. The switches that connect the I/O pin to analog global lines and analog mux line are configured by the {PRT*_AG} and {PRT*_AMUX} registers.

Refer to the Analog Routing chapter on page 273 for a description of the analog global network configuration. Selected pins provide direct connections to specific analog features, such as DACs or uncommitted opamps.

For analog I/O pins, the drive mode should be configured to High Z Analog in most situations, which disables the input buffer. The input buffer can also be disabled using the port input disable (PRTx_INP_DIS) register. The buffer should remain enabled to allow simultaneous use of the pin as a digital input and analog input or output.

21.3.7 LCD Drive

All GPIO pins can be configured for LCD drive capabilities. {PRT*_LCD_EN} registers are used to enable individual pins for LCD drive. {PRT*_LCD_COM_SEG} registers are used to select whether a pin is set as a common or segment drive pin.

In LCD mode, the GPIO pins are configured to a High Z output mode, allowing the LCD drivers to control the pin state.

21.3.8 CapSense

All GPIO pins can be used to create CapSense buttons and sliders. The primary analog bus for CapSense is the AMUXBUS, which has two nets (AMUXBUSL and AMUXBUSR) for two simultaneous sensing operations. These can also be shorted to form a single net that connects to all GPIOs. See the CapSense® chapter on page 305 for more information.

21.3.9 SIO Functions and Features

GPIO and SIO provide similar digital functionality. The primary differences are in their analog capability and drive strength. This section describes adjustable input and output level and hot swap features that are available only with SIO. Note that all SIO registers should be accessed as byte for reading or writing to them.

21.3.9.1 Regulated Output Level

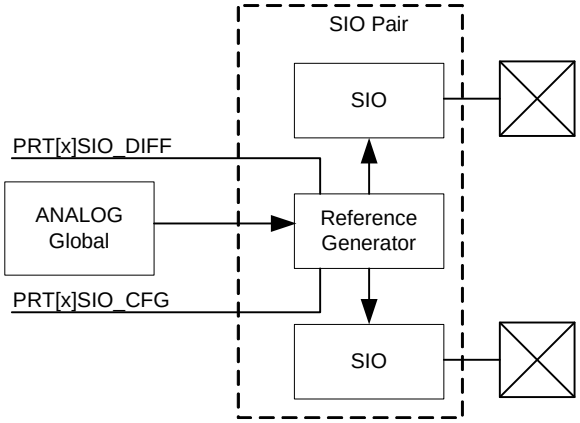
SIO port pins support the ability to provide a regulated high output level. This is useful for interfacing to external signals with voltages lower than the SIO V_{ddio}. This regulated output sets the V_{oh} for the SIO pair. The SIO are grouped into pairs. Each pair shares the same reference generator, thus the regulated output level applies for both pins.

Configuration is provided for each SIO pair through the {PRT*_SIO_CFG} registers, as shown in the following table.

Table 21-3. SIO Input and Output Configuration

vreg_en[y]	ibuf_sel[y]	Mode Description
0	0	Single Ended Input Buffer Non-Regulated Output Buffer
0	1	Differential Input Buffer Non-Regulated Output Buffer
1	0	Single Ended Input Buffer Regulated Output Buffer
1	1	Differential Input Buffer Regulated Output Buffer

Figure 21-9. SIO Configuration Diagram



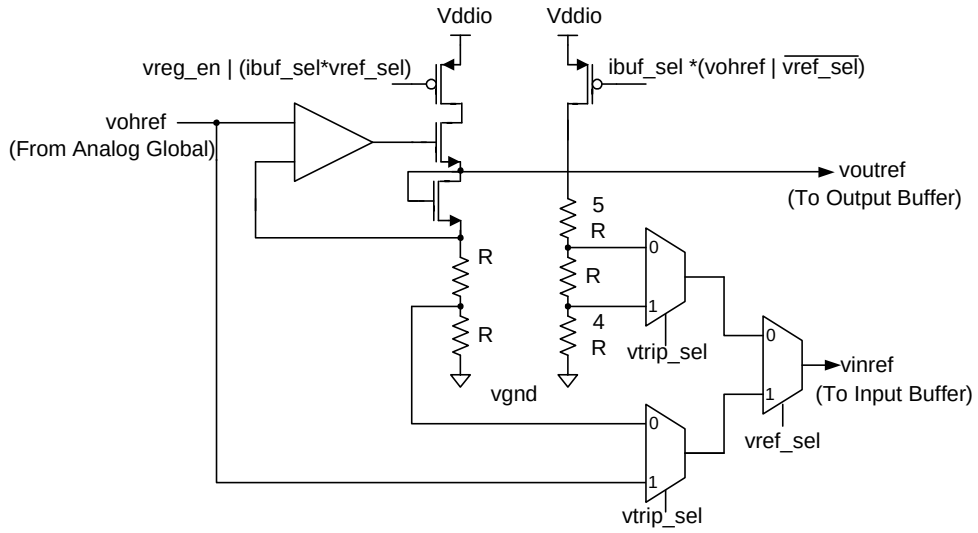
21.3.9.2 Adjustable Input Level

SIO pins support a differential input mode with programmable thresholds. The SIO pair input buffer voltage levels are set by the vref_sel and vtrip_sel bits of the {PRT*_SIO_DIFF} register. See the following table.

Table 21-4. SIO Differential Input Buffer Reference Voltage Selection

vref_sel[y]	vtrip_sel[y]	Mode Description
0	0	0.5 × vddio
0	1	0.4 × vddio
1	0	0.5 × vohref
1	1	vohref

Figure 21-10. SIO Reference Voltage



21.3.9.3 Hot Swap

SIO pins support hot swap capability. It is possible to connect to another system without loading the signals connected to the SIO pins and without applying power to the PSoC device.

The unpowered PSoC device can maintain a high impedance load to the external device while preventing the PSoC device from being powered through a GPIO pin's protection diode.

21.3.10 Special Functionality

Special purpose capability may uniquely exist on some pins such as:

- 4 to 25 MHz crystal input and output
- 32 kHz crystal input and output
- Test modes
- I^2C
- SPI
- USB

Special functions and peripherals such as I^2C, crystal oscillators, USB, XRES_N, SWD, high-current DAC outputs, V_{REF} inputs, and high drive analog output buffers have fixed pin assignments.

The I^2C block supports three pin assignment options: SIO pin pair P12[0:1], SIO pin pair P12[4:5], or any GPIO/SIO pin pair routed via the DSI.

System reset (XRES_N, active low, resistive pull up) functionality is supported on the dedicated XRES_N pin

Serial wire debug is supported over the USBIO pins (P15[6:7]) or the port 1 GPIO pins (P1[0:1]). Analog function fixed pin assignments include two pairs of VIDAC outputs to support high-current mode, two V_{REF} inputs, and four sets of analog output buffer pins. The "left side" VIDAC and analog buffer pins are assigned to port 0 and are available on all package options. The "right side" VIDAC and linear buffer pins are assigned to port 3 and are available on all package options.

Table 21-5. Fixed Pin Assignments

Function	Signal Name	Pad #	Pad Name	Pad Type	TQPF 100	QFN 68	Comment
I^2C	SCL	4	P12[4]	SIO	4	3	SIO pair on Vio2
	SDA	5	P12[5]	SIO	5	4	
	SCL	61	P12[0]	SIO	53	38	SIO pair on Vio3
	SDA	62	P12[1]	SIO	54	39	
MHz ECO	Xo	49	P15[0]	GPIO / Xtal	42	27	
	Xi	50	P15[1]	GPIO / Xtal	43	28	
32 kHz ECO	Xo	62	P15[2]	GPIO / Xtal	55	40	
	Xi	63	P15[3]	GPIO / Xtal	56	41	
FS USB	D+	39	P15[6]	USBIO	35	22	
	D-	40	P15[7]	USBIO	36	23	
XRES_N	XRES_N	19	XRES_N	XRES_N	15	10	Fixed function XRES_N pin
Serial Wire Debug	SWDIO	24	P1[0]	GPIO	20	11	SWD on GPIO pins option
		39	P15[6]	USBIO	35	22	SWD on USB pins option
	SWDCK	25	P1[1]	GPIO	21	12	SWD on GPIO pins option
		40	P15[7]	USBIO	36	23	SWD on USB pins option
	SWO	27	P1[3]	GPIO	23	14	
VIDAC High Current Output	Abuffer0L	82	P0[6]	GPIO	78	55	
	Abuffer1L	83	P0[7]	GPIO	79	56	
	Abuffer0R	51	P3[0]	GPIO	44	29	
	Abuffer1R	52	P3[1]	GPIO	45	30	
External Vref	Extref0	78	P0[3]	GPIO	74	51	
	Extref1	53	P3[2]	GPIO	46	31	
Analog Linear Output Buffer	Abuf0+	77	P0[2]	GPIO	73	50	
	Abuf0-	78	P0[3]	GPIO	74	51	
	Abuf0out	76	P0[1]	GPIO	72	49	
	Abuf1-	55	P3[4]	GPIO	48	33	
	Abuf1+	56	P3[5]	GPIO	49	34	
	Abuf1out	58	P3[6]	GPIO	51	36	
	Abuf2+	80	P0[4]	GPIO	76	53	
	Abuf2-	81	P0[5]	GPIO	77	54	
	Abuf2out	75	P0[0]	GPIO	71	48	
	Abuf3-	53	P3[2]	GPIO	46	31	
	Abuf3+	54	P3[3]	GPIO	47	32	
	Abuf3out	59	P3[7]	GPIO	52	37	

21.3.11 I/O Port Reconfiguration

Care must be taken not to lose the current configuration during reconfiguration of pins when the device is connected directly to a digital peripheral. The I/O pins should hold their current configurations during a reconfiguration. If the ports are driven by the data registers, configuration maintenance is automatic.

However, if the ports are bypassed and driven by the DSI, the current value must be read and written to the data register ({PRT*_DR}) before initiating reconfiguration. Saving of the current configuration occurs as follows:

1. The software reads the GPIO/SIO pin state, {PRT*_PS}.
2. The software writes this value into the data registers, {PRT*_DR}.
3. I/O ports driven by the DSI must be driven by the data register by de-asserting the bypass register value, {PRT*_BYP}.

At this point, it is safe to reconfigure the device. When reconfiguration is complete, the I/O sources can be driven by the DSI by setting the {PRT*_BYP} register value.

21.3.12 Power Up I/O Configuration

By default, all I/Os power up in a known state set to High Z. Input buffers are disabled during power up.

21.3.13 Overvoltage Tolerance

All I/O pins provide an overvoltage ($V_{ddio} < V_{in} < V_{dda}$) tolerance feature at any operating voltage. Limitations include the following:

- No current limitations for the SIO pins, because they present a high impedance load to the external circuit.
- GPIO pins must be limited to 100 µA, using a current limiting resistor. Outside the current limitation, GPIO pins clamp the pin voltage to approximately one diode above the V_{ddio} supply.

A common application for this feature is connection to a bus such as I^2C, where different devices are running from different supply voltages. In the I^2C case, the PSoC device is configured into the Open Drain, Drives Low mode using an SIO pin. This allows an external pull up to pull the I^2C bus voltage above the pin's V_{ddio} supply. For example, the PSoC device can operate at 3.3 V, and an external device can run from 5 V. The SIO pin's V_{IH} and V_{IL} levels are determined by the associated V_{ddio} supply pin.

The I/O pin must be configured into a High Impedance drive mode, Open Drain Low mode, or Resistive Pull Down mode, for overvoltage tolerance to work properly.

Absolute maximum ratings for the device must be observed for all I/O pins.

SIO pins can tolerate up to 5.5 V on them by posing high impedance to the voltages higher than V_{ddio}.

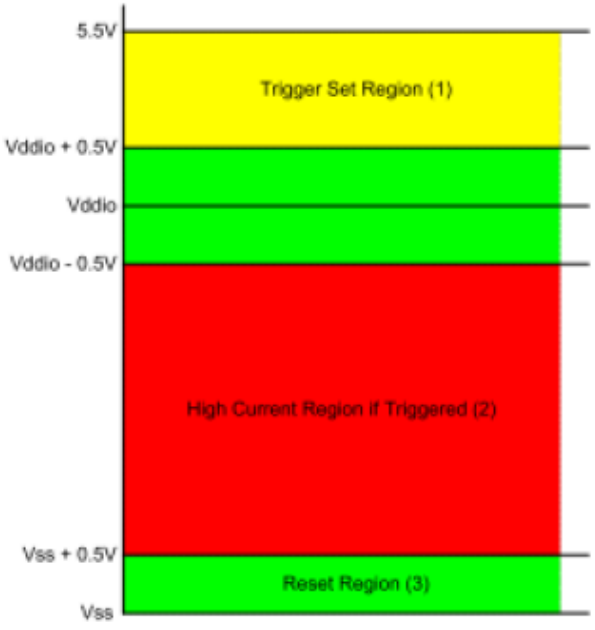
If an SIO pin's voltage exceeds its V_{ddio} supply by 0.5 V, the trigger condition is set (region 1). After the trigger condition is set, the SIO pin causes increased current up to 1 mA, when its voltage is between V_{ss} + 0.5 V and V_{ddio} – 0.5 V (region 2). The trigger condition is reset when the SIO pin is brought within the range of V_{ss} to V_{ss} + 0.5 V (region 3). The trigger condition may be met unknowingly during device power-up due to differences in supply ramps.

The following methods help to avoid high current consumption:

- The SIO pin should be made to have quick transition through the high current region. Higher current is seen during the brief transition period through the high current region from high to low logic levels if the trigger condition is met.
- If the SIO can be brought back between V_{ss} and V_{ss} + 0.5 V, the trigger condition can be reset until the pin re-triggers the condition.

The SIO can be brought back to V_{ss} by setting the pin to a low logic level by using API, DMA, or hardware.

Figure 21-11. SIO High Current Region



21.3.14 I/O Power Supply

The V_{ddio} supply must be less than or equal to the voltage on the device's V_{dda} pin. This feature allows users to pro-

vide different I/O interface levels for different pins on the device. Refer to the data sheet to determine V_{ddio} capability for a given device and pin.

SIO port pins support an additional regulated high output capability, as discussed in 21.3.9.2 Adjustable Input Level.

21.3.15 Sleep Mode Behavior

The GPIO/SIO pad will maintain the current pin state during sleep modes.

21.3.16 Low Power Behavior

In all low power modes, I/O pins retain their states until the part is awakened and changed or reset.

21.4 Port Interrupt Controller Unit

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

21.4.1 Features

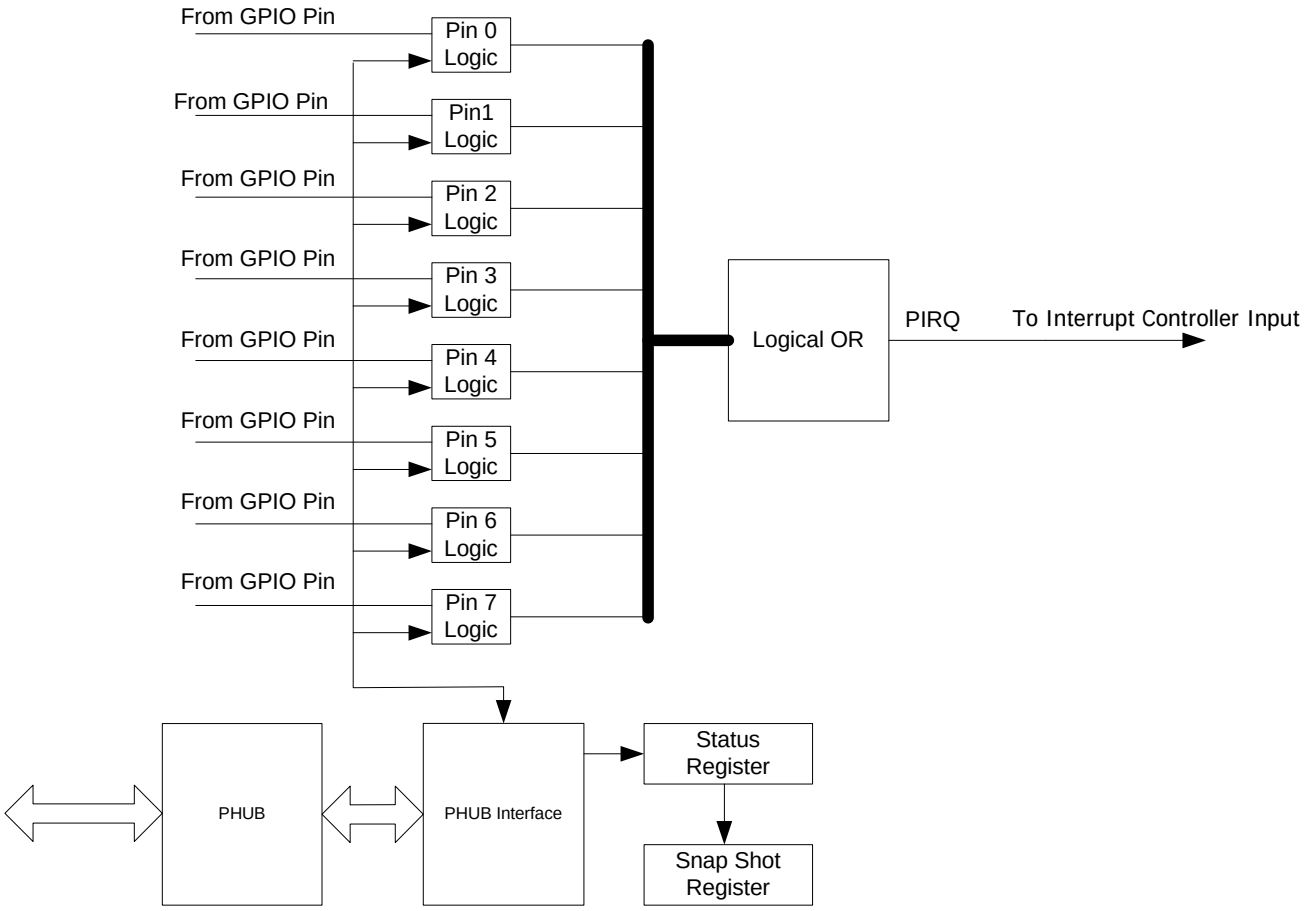
The features of the PICU are as follows:

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

21.4.2 Interrupt Controller Block Diagram

Figure 21-12 is a block diagram of the PICU showing the function of control signal generation and data manipulation blocks. These blocks send appropriate control signals to interrupt-generating pin logic blocks, simultaneously recording these signals in status and snap registers.

Figure 21-12. PICU Block Diagram



21.4.3 Function and Configuration

Each pin of the port can be configured independently to generate interrupt on rising edge, falling edge, or either edge. Level sensitive interrupts are not directly supported. UDB provides this functionality to the system when needed. This configuration is done by writing into the interrupt type register corresponding to each pin. The sequence is as follows:

1. Depending on the configured mode for each pin, whenever the selected edge occurs on a pin, its corresponding status bit in the status register is set to ‘1’, and an interrupt request is sent to the interrupt controller.
2. Status bits that have ‘1’ are cleared upon a read of the status register. Other bits of the status register can still respond to incoming interrupt sources.
3. If an interrupt is pending, and the status register is being read, all of the incoming events on the same interrupt source (GPIO) are blocked until the read is complete. However, all of the other interrupt sources that were not pending an interrupt in status register are not blocked.

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

21.5 Register Summary

Registers shown in Table 21-6 are associated with a single I/O port and are specific to both the GPIO and SIO ports.

Table 21-6. GPIO and SIO Port Registers

Address	Description
{PRT*_DR}	The Port Data Output register sets the data output state for the corresponding GPIO port. It is aliased to continuous address space in the PRT*_DR_ALIASED registers.
{PRT*_PS}	The Port Pin State register reads the logical pin state for the corresponding GPIO port. It is aliased to continuous address space in the PRT*_PS_ALIASED registers.
{PRT*_DM*}	The Port Drive Mode registers ({PRT*_DM[0]}, {PRT*_DM[1]}, and (PRT*_DM[2]}) specify the drive mode for I/O pins.
{PRT*_SLW}	The Port Slew Control register sets the slew rate for pin outputs.
{PRT*_BYP}	The Port Bypass register selects port output data from either the data output register or digital global input.
(PRT*_BIE}	The Port Bidirectional Enable register enables dynamic bidirectional mode at any pin.
{PRT*_INP_DIS}	The Port Input buffer disable allows the user to override the input buffer default drive mode settings.
{PRT*_BIT_MSK}	Mask of which bits within the {PRT*_DR} and {PRT*_PS} are accessible via read / writes to {PRT*_DR_ALIAS} and reads of {PRT*_PS_ALIAS}.
{PRT*_AG}	The Analog global control enable register selects on a pin-by-pin basis whether to connect the pin to the analog global bus.
{PRT*_AMUX}	The Analog Global Multiplexer Register selects on a pin-by-pin basis whether to connect the pin to the analog mux bus.
{PRT*_PRT}	The Port Configuration Register allows configuration of several configuration bits of the entire I/O port simultaneously. This register aliases the port functional registers on a port-wide basis.
{PRT*_PC*}	The Port Pin Configuration Registers ({PRT*_PC[0] through {PRT*_PC[7]}) access several configuration or status bits of a single I/O port pin simultaneously. These registers alias the functional registers on a pin-by-pin basis.
{PRT*_DR_ALIAS}	Aliased port data. Allows read / write access to {PRT*_DR} if {PRT*_BIT_MSK} is set. Allows access to all port data registers as a contiguous block simplifying DMA access.
{PRT*_PS_ALIAS}	Aliased port data. Allows read access to {PRT*_PS} if {PRT*_BIT_MSK} is set. Allows access to all port state registers as a contiguous block simplifying DMA access.

Table 21-7 shows registers specific to a GPIO port.

Table 21-7. GPIO Registers

Address	Description
{PRT*_CTL}	Port-wide configuration register. This contains the port-wide vtrip_sel for the corresponding GPIO register.
{PRT*_LCD_COM_SEG}	LCD com_seg setting. This selects common or segment mode when the LCD is enabled.
{PRT*_LCD_EN}	LCD enable, allows port pins not connected to LCD to be used for other functions.

Table 21-8 shows registers specific to an SIO port.

Table 21-8. SIO Port Registers

Address	Description
{PRT*_SIO_DIFF}	Differential input buffer reference voltage select, 2 bits per SIO pair.
{PRT*_SIO_CFG}	Input buffer enable and Output buffer Configuration, 2 bits per SIO pair.
{PRT*_SIO_HYST_EN}	Differential hysteresis enable.

Registers shown in Table 21-9 involve DSI bit selection. These registers are associated with all I/O ports and are located within the port logic.

Table 21-9. DSI Selection Registers

Address	Description
{PRT*_OUT_SEL*}	Data output from UDB to Digital System Array Input Select registers. There are two select lines per port pin.
{PRT*_OE_SEL*}	UDB set dynamic Output Enable control select. There are two select lines per port pin.
{PRT*_DBL_SYNC_IN}	The Port Double Sync In register enables synchronization of the data in from the port before driving the digital system interconnect (DSI) signals to the UDB.
{PRT*_SYNC_OUT}	The Port Sync Out register enables synchronization of the data in from the UDB digital system interconnect (DSI) using the existing {PRT*_DR} register.

Table 21-10 shows the register associated with the PICU.

Table 21-10. PICU-Associated Registers

Address	Description
{PICU*_INTTYPE*}	This register defines the interrupt type to configure the pin interrupt – 1 register for each pin
{PICU*_INTSTAT}	Status register provides information about currently posted interrupts – 1 register for each PICU
{PICU*_SNAP}	The Port Snapshot register provides information about the state of the input pins at the most recent read to the status (INTSTAT) register – 1 register for each PICU

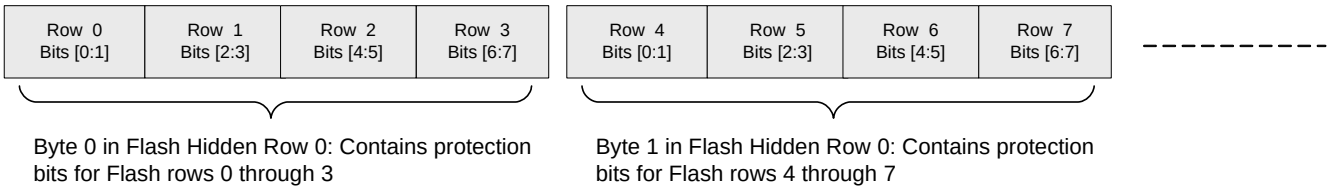
22. Flash Protection

PSoC® 5 devices offer a host of Flash protection options and device security features that can be leveraged to meet the security and protection requirements of an application. These requirements range from protecting configuration settings or Flash data to locking the entire device from external access. The following section discusses in detail these features together with their use cases.

22.1 Flash Protection

The objective of Flash protection is to prevent access or modification to the flash contents. The only nonvolatile (NV) storage on a PSoC 5 device that has protection options is the Flash; there are no EEPROM protection options. Flash memory in PSoC 5 architecture is organized as Flash arrays. Depending on the Flash memory size, there can be one or more than one Flash array. Each Flash array can have a maximum of 256 rows. Each Flash array row has 256 bytes of data. PSoC 5 architecture offers customers the ability to assign one of four protection levels to each row of Flash in a device. For each Flash array, Flash protection bits are stored in a hidden row in that array. In the hidden row, two protection bits per row are packed into a byte, so each byte in the hidden row has protection settings for four Flash rows. The Flash rows are ordered so that the first two bits in the hidden row correspond to the protection settings of Flash row 0 (see Figure 22-1). See the Flash Program Memory chapter on page 83 to learn more about Flash memory organization in PSoC 5 devices.

Figure 22-1. Flash Protection Bit Structure



Protection is cumulative in that modes have successively higher protection levels and include the lower protection modes. Flash protection can only be set once. To change Flash protection settings after they have been set, the Flash contents must be completely erased and reprogrammed, then the protection levels can be set again. See the Nonvolatile Memory Programming chapter on page 357 for erasing and programming Flash. Table 22-1 shows the protection modes.

Table 22-1. Flash Protection Modes

Mode	Description	Read[a]	External Write[b]	Internal Write[c]
00	Unprotected	Yes	Yes	Yes
01	Read Protect	No	Yes	Yes
10	Disable External Write	No	No	Yes
11	Disable Internal Write	No	No	No

a. Applies to Test Controller and Read commands.
b. Test controller/3rd party programmers.
c. Boot loading or writes due to firmware execution.

When a read/write/erase operation is done for a row, the corresponding protection bits are checked. The command is executed only if allowed under the current protection mode. If the command is not allowed, then the command fails.

As shown in Table 22-1, four Flash protection levels are available for every row of Flash in a device. The user may

choose any one of these protection levels independent of the protection choice for all other rows in the Flash.

The following list provides a few additional details about the features and use cases for each of these protection levels.

- 00 – No Protection
- 01 – Read Protect

  No external device can read a flash block that is read protected.

  The SPC Read commands cannot be used to read a block that is read protected.

  Only the processor and the PHUB can access a block of Flash that is read protected.

  Offers only read protection.
- 10 – External Write Protection

  No external device can erase or write a row of Flash that is external write protected.

  Includes all Read Protect restrictions.

  Boot loaders work at this protection level.
- 11 – Fully Protected

  The processor cannot erase or write a block of Flash that is fully protected.

  Includes all protections from lower levels of Flash data protection.

  This level is used when a block of Flash should never be modified by an internal process or external device.

Note that when the debug controller is enabled, it can read the entire flash memory regardless of the flash protection setting. Therefore, if flash protection is required, the debug controller also needs to be disabled.

22.2 Device Security

The objective of device security is to prevent the PSoC 5 device in an application from being used as a host to compromise the application. The device security feature is enabled by writing to the Write Once (WO) latch.

The WO latch is a type of nonvolatile latch. When the output is ‘1’, the Write Once NVL locks the part out of Debug and Test modes; it also permanently gates off the ability to erase or alter the contents of the latch.

The user can write a correct 32-bit key (0x50536F43) into the WO latch to disable the part from entering into Debug and Test modes. This precaution prevents anyone from erasing or altering the content of the internal memory.

If the device is protected with a WO latch setting, Cypress cannot perform failure analysis and, therefore, cannot accept an RMA from customers. The WO latch is read out via serial wire debug (SWD) to electrically identify protected parts. The user writes the key in the WO latch to lock out external access only if no Flash protection is set. However, after setting the values in the WO latch, a user still has access to the device until it is reset. The output of the WOL is only sampled upon reset. Therefore, a user can write the key into the WO latch, program the Flash protection data, and then reset the part to lock it.

See the Nonvolatile Memory Programming chapter on page 357 for information about writing to the Write Once (WO) nonvolatile latch.

22.3 Frequently Asked Questions About Best Practices for Flash Protection and Device Security

Question 1. How do I decide on the Flash protection level needed for the application?

The protection settings for Flash memory must be set based on the following criteria:

- If the application warrants the need for a field upgrade, then set the Disable External Write mode for the Flash rows that are going to be updated in the field. This allows you to use the bootloader application to update the flash using communication interfaces such as I^2C and USB.
- If the application code must be protected from being copied or modified to protect IP, the Flash security level for the rows containing the IP code must be set to Full Protection mode.

Question 2. Is it possible to modify the Flash protection settings that have already been set?

It is not possible to directly alter the Flash protection setting. The only way to change the Flash protection settings is to completely erase the entire Flash memory using the Erase All command, reprogram the Flash memory, and then set the new protection settings. Refer to the Nonvolatile Memory Programming chapter on page 357 to learn more about Flash erase/program commands.

Question 3. Is it possible to reprogram a Flash memory that has been configured with Full Protection?

The only way to reprogram the fully protected rows is to erase the entire Flash memory using the Erase All command, reprogram the Flash memory, and then set the new protection settings as described in Question 2 above.

Question 4. Is it necessary to enable protection for the entire Flash memory, or only the for the region of Flash memory that the application uses?

It is sufficient to configure Flash security for memory regions that are used by the application, leaving the unused locations unprotected, provided that there is no possibility of the program execution going to the unprotected region. If there is a possibility of code executing from the unprotected region (due to, for example, function calls), malicious code can be written in the unprotected region to read the Flash data in the fully protected region. Remember that internal read is permitted in all protection modes; therefore, it is always a good practice to set protection for the entire Flash memory.

Question 5. Is it ever necessary to configure different protection settings for different memory regions?

Yes, depending on the application requirements. Different flash rows may need different protection settings. A typical example is the case of field upgrade using the bootloader component. The portion of Flash that needs to be upgraded in the field with bootloadable code must be configured in External Write Protect mode. The remaining Flash memory (base code or bootloader code, unused flash memory) can be set to Full Protection.

Question 6. Are Flash protection settings obeyed in Debug mode?

The Read Protection setting is not obeyed in Debug mode, which means the Flash memory can be read regardless of Flash protection setting. The Write Protection setting is still intact. Setting Full Protection makes it impossible to write to the Flash memory in Debug mode.

Because the Debug mode is used during the application development phase, there is no need to protect the Flash. After the application development phase is over, and code has been finalized, the user can disable the debug feature.

Question 7. What is device security?

Device security is the feature in PSoC 5 architecture that prevents the device from entering Debug and Test modes. To enable device security, write a 32-bit key (0x50536F43) into the Write Once (WO) latch. After writing this key, the device cannot be reprogrammed by entering test mode. This prevents external access to registers and nonvolatile memory. Refer to Device Security on page 144 of this chapter to learn more about device security.

Question 8. What are the risks associated with enabling device security?

If the device is protected with a WO latch setting, Cypress cannot perform failure analysis and, therefore, cannot accept RMAs from customers. The WO latch can be read via the SWD to electrically identify protected parts.

Question 9. Are device security and flash protection interrelated or independent?

The answer is both. While flash protection settings and device security are configured independently, enabling device security does not allow external read or write of Flash memory, regardless of the flash protection settings. There is one important exception. Even with device security enabled, it is still possible to update the Flash memory using a bootloader application, provided the Flash memory is not fully protected.

Question 10. Is it possible to implement OTP (one time programmable) functionality such that Flash content can never be altered after it is programmed?

The Full Protection setting for Flash memory, along with the device security feature can prevent the Flash from ever being modified. This combination is the highest level of security setting available in PSoC 5 devices. The steps to do this are given below

1. Erase the entire Flash memory using the Erase All command
2. Reprogram the Flash content.
3. Write a 32-bit key (0x50536F43) into the WO latch to enable device security.
4. Set Flash Protection setting to Full Protection.
5. Reset the part to lock it.

Section E: Digital System

The digital subsystems of PSoC® 5 architecture provide these devices their first half of unique configurability. The subsystem connects a digital signal from any peripheral to any pin through the Digital System Interconnect (DSI). It also provides functional flexibility through an array of small, fast, low-power Universal Digital Blocks (UDBs).

PSoC Creator™ provides a library of pre-built and tested standard peripherals that are mapped onto the UDB array by the tool (UART, SPI, LIN, PRS, CRC, timer, counter, PWM, AND, OR, and so on). Nonstandard peripherals are easily implemented using a Hardware Description Language (HDL) such as Verilog. Each UDB contains Programmable Array Logic (PAL) and Programmable Logic Device (PLD) functionality, together with a small state machine engine to support a wide variety of peripherals.

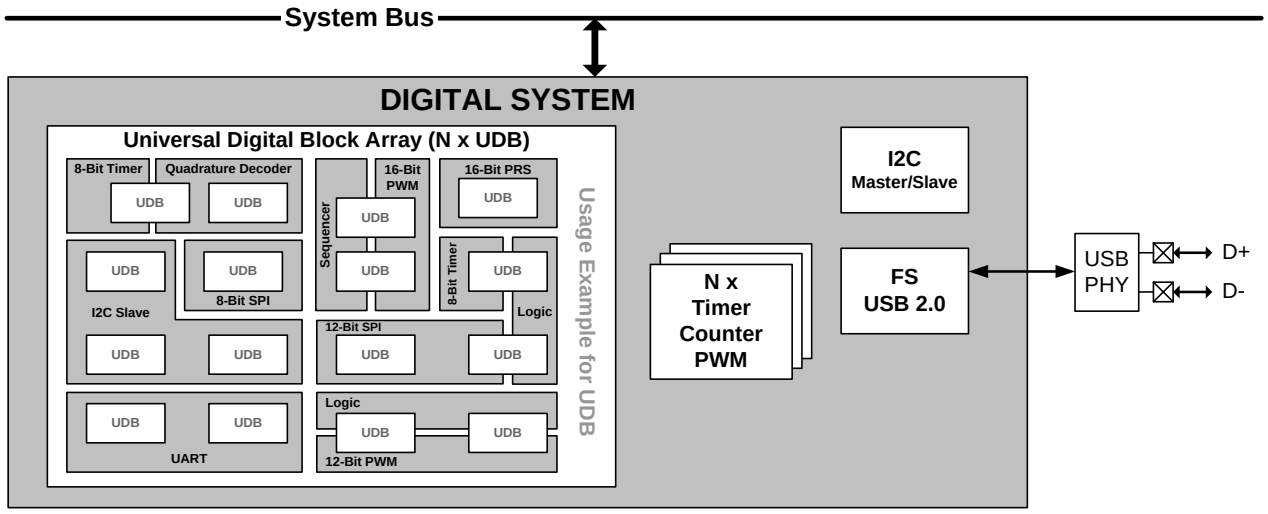
In addition to the flexibility of the UDB array, PSoC devices provide configurable digital blocks targeted at specific functions. These blocks can include 16-bit timer/counter/PWM blocks, I^2C slave/master/multi-master, and Full Speed USB. Refer to the device datasheet for a list of available specific function digital blocks.

This section encompasses the following chapters:

- Universal Digital Blocks (UDBs) chapter on page 149
- UDB Array and Digital System Interconnect chapter on page 191
- USB chapter on page 199
- Timer, Counter, and PWM chapter on page 211
- I^2C chapter on page 227
- Digital Filter Block (DFB) chapter on page 241

Top Level Architecture

Digital System Block Diagram



23. Universal Digital Blocks (UDBs)

This chapter shows how the PSoC® 5 Universal Digital Blocks (UDBs) enable the development of programmable digital peripheral functions. The UDB architecture provides balance between configuration granularity and efficient implementation; UDBs consist of a combination of uncommitted logic similar to programmable logic devices (PLDs), structured logic (datapaths), and a flexible routing scheme.

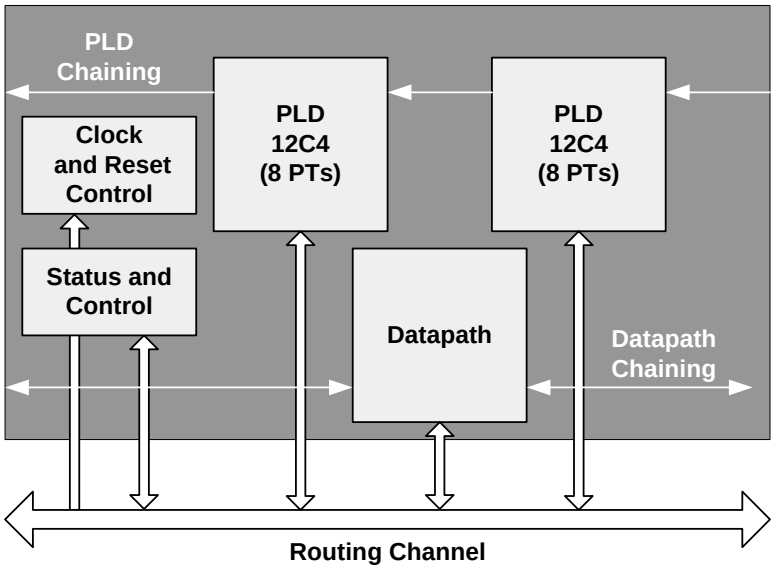
23.1 Features

- For optimal flexibility, each UDB contains several components:
  - ALU-based 8-bit datapath (DP) with an 8-word instruction store and multiple registers and FIFOs
  - Two PLDs, each with 12 inputs, eight product terms and four macrocell outputs
  - Control and status modules
  - Clock and reset modules
- The PSoC 5 device contains an array of up to 24 UDBs
- Flexible routing through the UDB array
- Portions of UDBs can be shared or chained to enable larger functions
- Flexible implementations of multiple digital functions, including timers, counters, PWM (with dead band generator), UART, I^2C, SPI, and CRC generation/checking

23.2 Block Diagram

Figure 23-1 on page 150 illustrates the UDB as a construct containing a pair of basic PLD logic blocks, a datapath, and control, status, clock and reset functions.

Figure 23-1. UDB Block Diagram



23.3 How It Works

The major components of a UDB are:

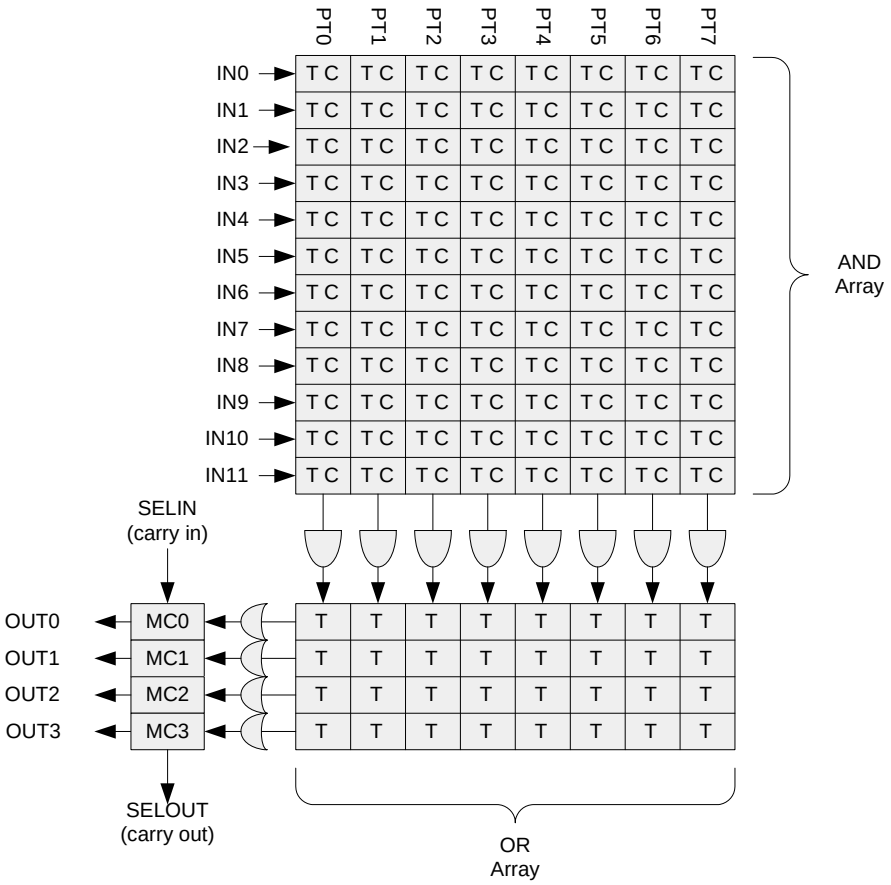
- **PLDs (2)** – These blocks take inputs from the routing channel and form registered or combinational sum-of-products logic to implement state machines, control for datapath operations, conditioning inputs, and driving outputs.
- **Datapath** – This block contains a dynamically programmable ALU, four registers, two FIFOs, comparators, and condition generation.
- **Control and Status** – These modules provide a way for CPU firmware to interact and synchronize with UDB operation. Control registers drive internal routing, and status registers read internal routing.
- **Reset and Clock Control** – These modules provide clock selection and enabling, and reset selection, for the other blocks in the UDB.
- **Chaining Signals** – The PLDs and datapath have chaining signals that enable neighboring blocks to be linked, to create higher precision functions.
- **Routing Channel** – UDBs are connected to the routing channel through a programmable switch matrix for connections between blocks in one UDB, and to all other UDBs in the array. Routing is covered in detail in the UDB Array and Digital System Interconnect chapter on page 191.
- **System Bus Interface** – All registers and RAM in each UDB are mapped into the system address space and are accessible by the CPU and DMA as both 8-bit and 16-bit data.

23.3.1 PLDs

There are two “12C4” PLDs in each UDB. PLD blocks, shown in Figure 23-2 on page 151, can be used to implement state machines, perform input or output data conditioning, and create lookup tables (LUTs). PLDs may also be configured to perform arithmetic functions, sequence the datapath, and generate status. General purpose RTL can be synthesized and mapped to the PLD blocks. This section presents an overview of the PLD design.

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

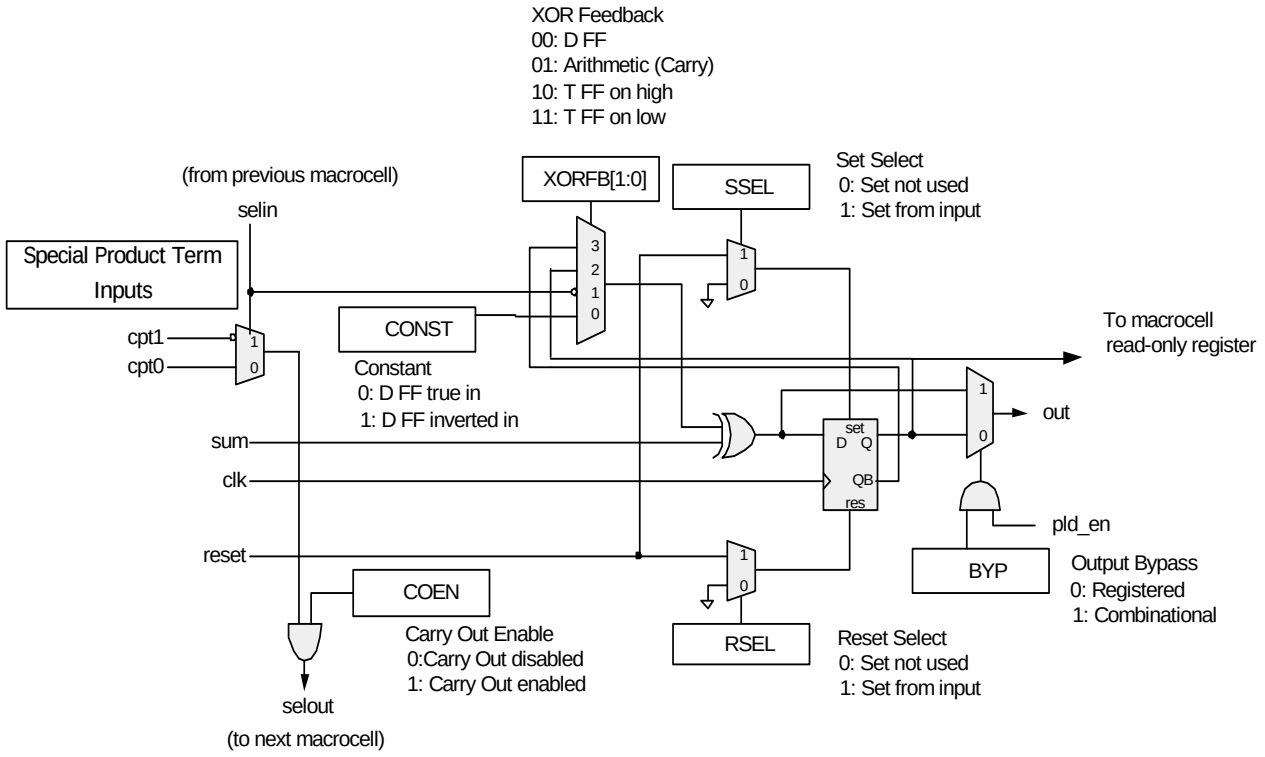
Figure 23-2. PLD 12C4 Structure



23.3.1.1 PLD Macrocells

The macrocell architecture is shown in Figure 23-3 on page 152. The output drives the routing array, and can be registered or combinational. The registered modes are D Flip-Flop with true or inverted input, and Toggle Flip-Flop on input high or low. The output register can be set or reset for purposes of initialization, or asynchronously during operation under control of a routed signal.

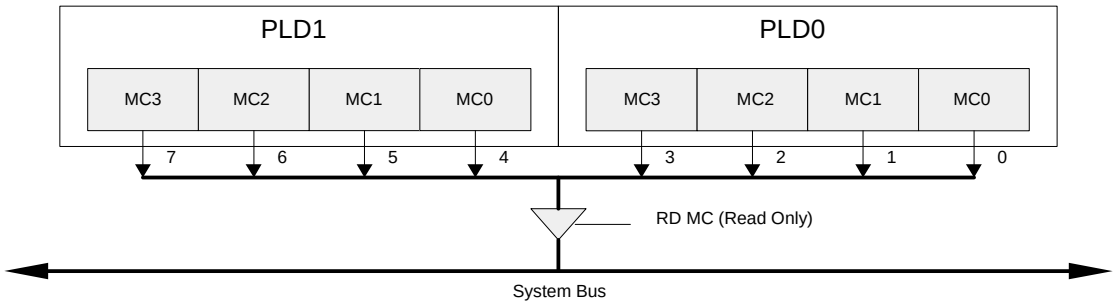
Figure 23-3. Macrocell Architecture



PLD Macrocell Read Only Register

In addition to driving the routing array, the outputs of the macrocells from both PLDs are mapped into the address space as an 8-bit read only register, which can be accessed by the CPU or DMA.

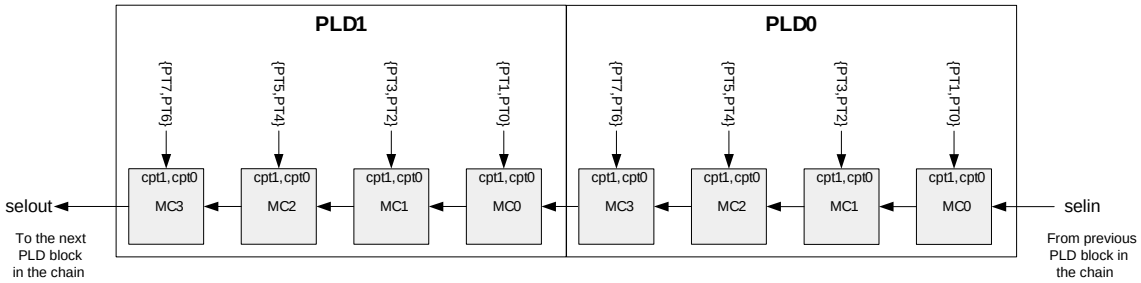
Figure 23-4. PLD Macrocell Read Only Register



23.3.1.2 PLD Carry Chain

PLDs are chained together in UDB address order. As shown in Figure 23-6, the carry chain input “selin” is routed from the previous UDB in the chain, through each macrocell in both of the PLDs, and then to the next UDB as the carry chain out “selout”. To support the efficient mapping of arithmetic functions, special product terms are generated and used in the macrocell in conjunction with the carry chain.

Figure 23-5. PLD Carry Chain and Special Product Term Inputs



23.3.1.3 PLD Configuration

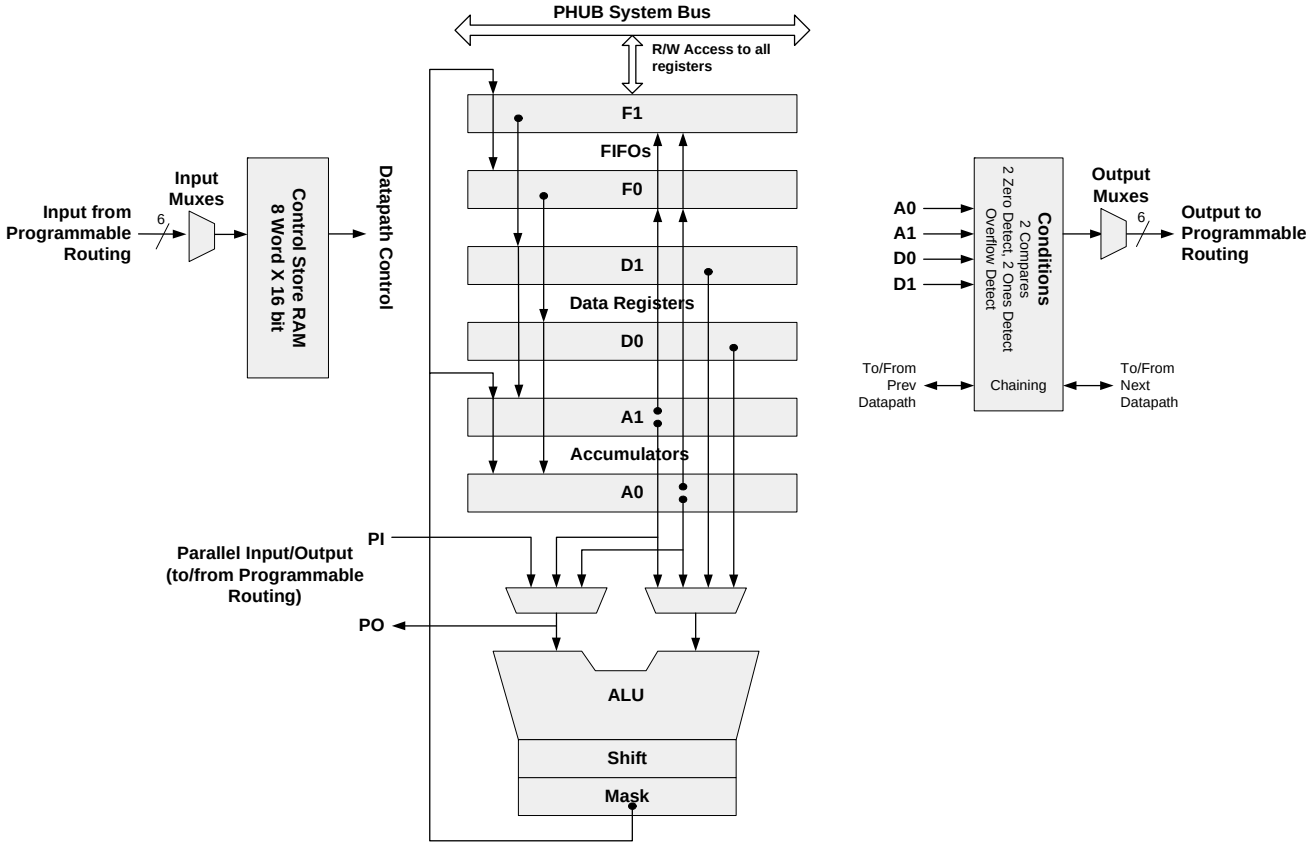
Each PLD appears to the CPU or DMA as a 16-bit wide RAM. The AND array has 12 × 8 × 2 bits, or 24 bytes, for programming, and the OR array has 4 × 8 bits, or 4 bytes, for programming. In addition, each macrocell has one configuration byte, resulting in 32 total configuration bytes per PLD. Because each UDB contains two PLDs, there are 64 total PLD configuration bytes per UDB. See UDB Configuration Address Space on page 187 for more information.

23.3.2 Datapath

The datapath, shown in Figure 23-6, contains an 8-bit single-cycle ALU, with associated compare and condition generation circuits. A datapath may be chained with datapaths in neighboring UDBs to achieve higher precision functions. The datapath includes a small RAM-based control store, which can dynamically select the operation to perform in a given cycle.

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

Figure 23-6. Datapath Top Level



23.3.2.1 Overview

The following sections present an overview description of key datapath features:

Dynamic Configuration

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

ALU

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

Conditionals

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

Built in CRC/PRS

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

Variable MSB

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

Input/Output FIFOs

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

Chaining

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

Time Multiplexing

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

Datapath Inputs

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

Datapath Outputs

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

Datapath Working Registers

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

Table 23-1. Datapath Working Registers

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

23.3.2.2 Datapath FIFOs

FIFO Modes and Configurations

Each FIFO has a variety of operation modes and configurations available:

Table 23-2. FIFO Modes and Configurations

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

Figure 23-7 shows the possible FIFO configurations controlled by the input/output modes. The TX/RX mode has one FIFO in input mode and the other in output mode. The primary usage example of this configuration is SPI. The dual capture configuration provides independent capture of A0 and A1, or two separately controlled captures of either A0 or A1. Finally, the dual buffer mode can provide buffered periods and compares, or two independent periods/compares.

Figure 23-7. FIFO Configurations

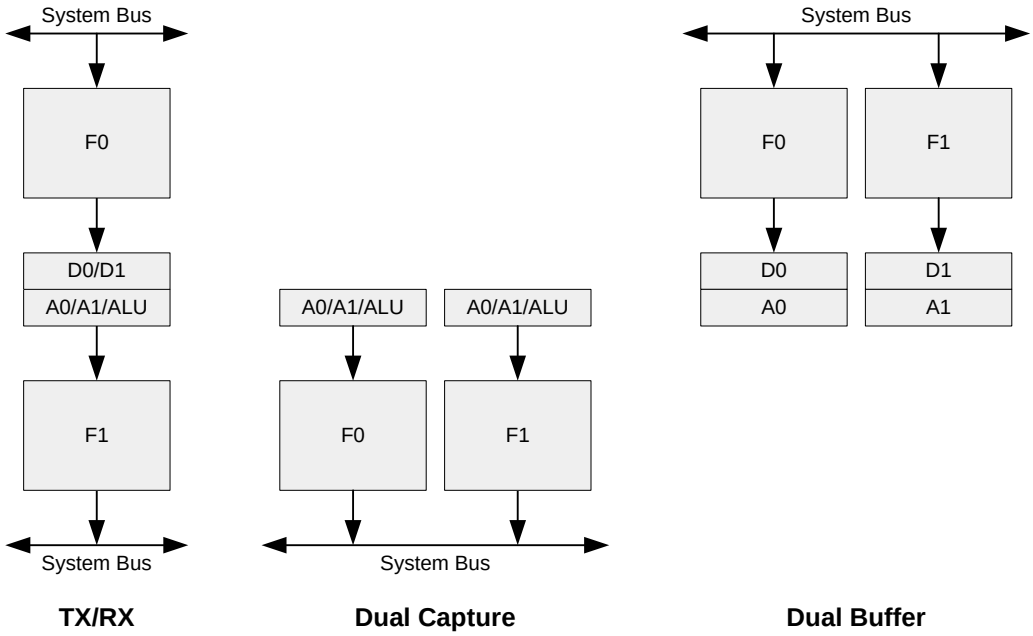
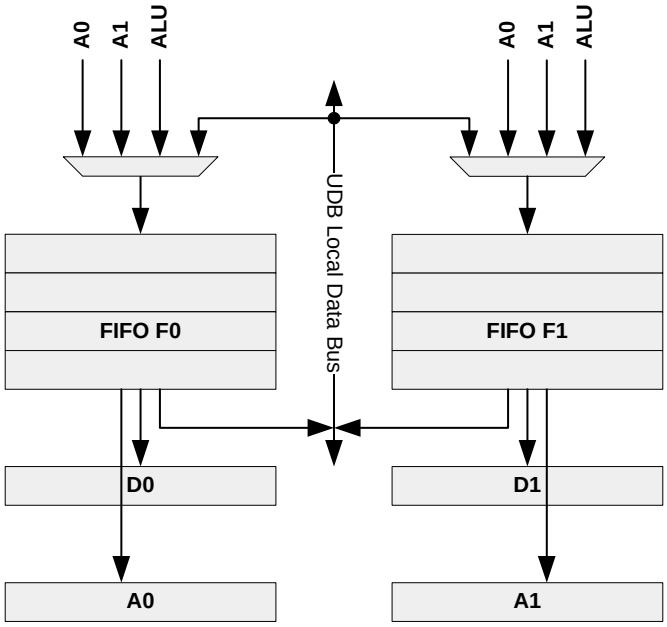


Figure 23-8 shows a detailed view of the FIFO sources and sinks.

Figure 23-8. FIFO Sources and Sinks



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

Table 23-3. FIFO Multiplexer Set in UDB Configuration Register

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

FIFO Status

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

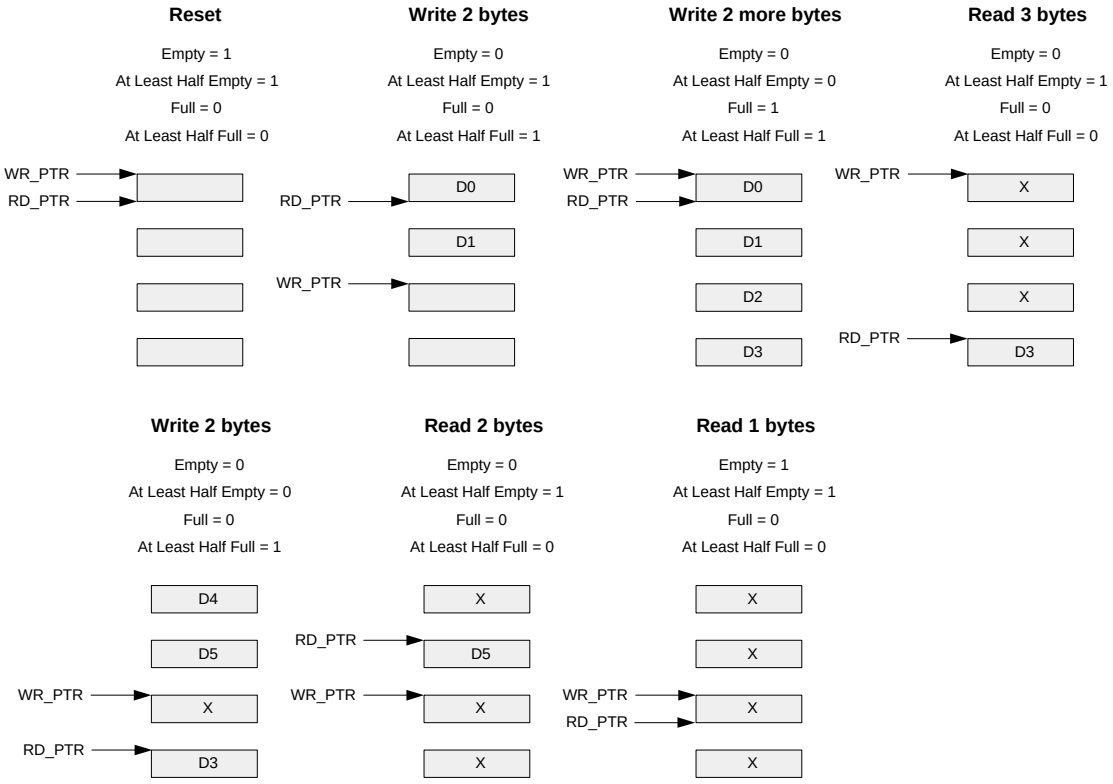
Table 23-4. FIFO Status Options

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

FIFO Illustrated Operation

Figure 23-9 on page 159 illustrates a typical sequence of reads and writes and the associated status generation. Although the figure shows reads and writes occurring at different times, a read and write can also occur simultaneously.

Figure 23-9. Detailed FIFO Operation Sinks

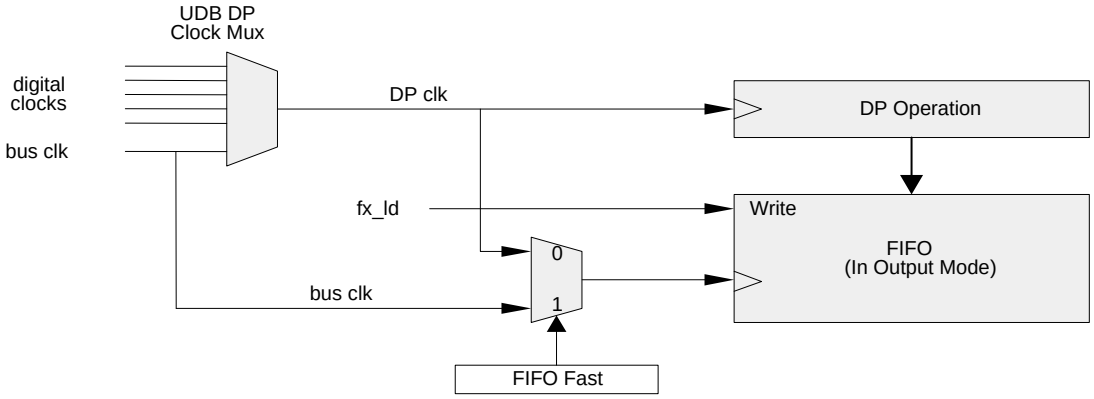


FIFO Fast Mode (FIFO FAST)

When the FIFO is configured for output, the FIFO load operation normally uses the currently selected datapath clock for sampling the write signal. As shown in Figure 23-10, with the FIFO fast mode set, the bus clock can be optionally selected for this operation. Used in conjunction with edge sensitive mode, this operation reduces the latency of accumulator-to-FIFO transfer from the resolution of the DP clock to the resolution of the bus clock, which can be much higher. This allows the CPU or DMA to read the captured result in the FIFO with minimal latency.

As shown in Figure 23-10, the fast load operation is independent of the currently selected datapath clock, however, using the bus clock may cause higher power consumption.

Figure 23-10. FIFO Fast Configuration Sinks

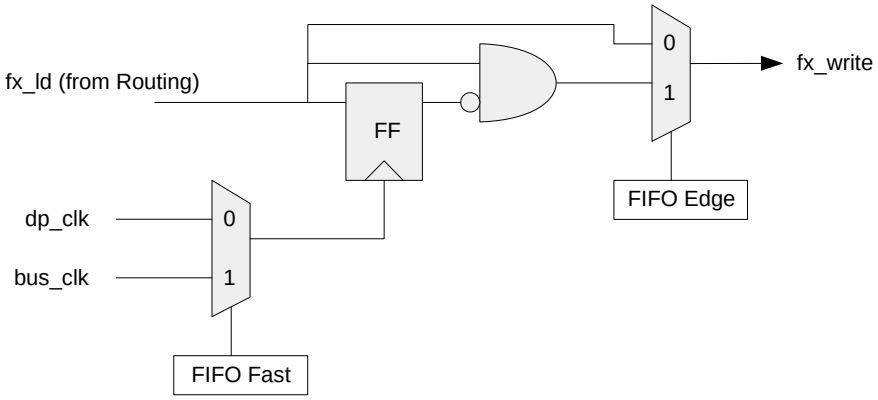


FIFO Edge/Level Write Mode

There are two modes of writing the FIFO from the datapath. In the first mode, data is synchronously transferred from the accumulators to the FIFOs. The control for that write (FX_LD) is typically generated from a state machine or condition that is synchronous to the datapath clock. The FIFO is written in any cycle where the input load control is a '1'. In the second mode, the FIFO is used to capture the value of the accumulator in response to a positive edge of the FX_LD signal. In this mode, the duty cycle of the waveform is arbitrary (however, it must be at least one datapath clock cycle in width). An example of this mode is capturing the value of the accumulator using an external pin input as a trigger. The limitation of this mode is that the input control must revert to '0' for at least one cycle before another positive edge is detected.

Figure 23-11 shows the edge detect option on the FX_LD control input. One bit for this option sets the mode for both FIFOs in a UDB. Note that edge detection is sampled at the rate of the selected FIFO clock.

Figure 23-11. Edge Detect Option for Internal FIFO Write Sinks



FIFO Software Capture Mode

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

As shown in Figure 23-12, reading the accumulator triggers a write to the FIFO from that accumulator. This signal is chained so that a read of a given byte simultaneously captures accumulators in all chained UDBs. This allows an 8-bit processor to reliably read 16 bits or more simultaneously. The data returned in the read of the accumulator should be ignored; the captured value may be read from the FIFOs immediately.

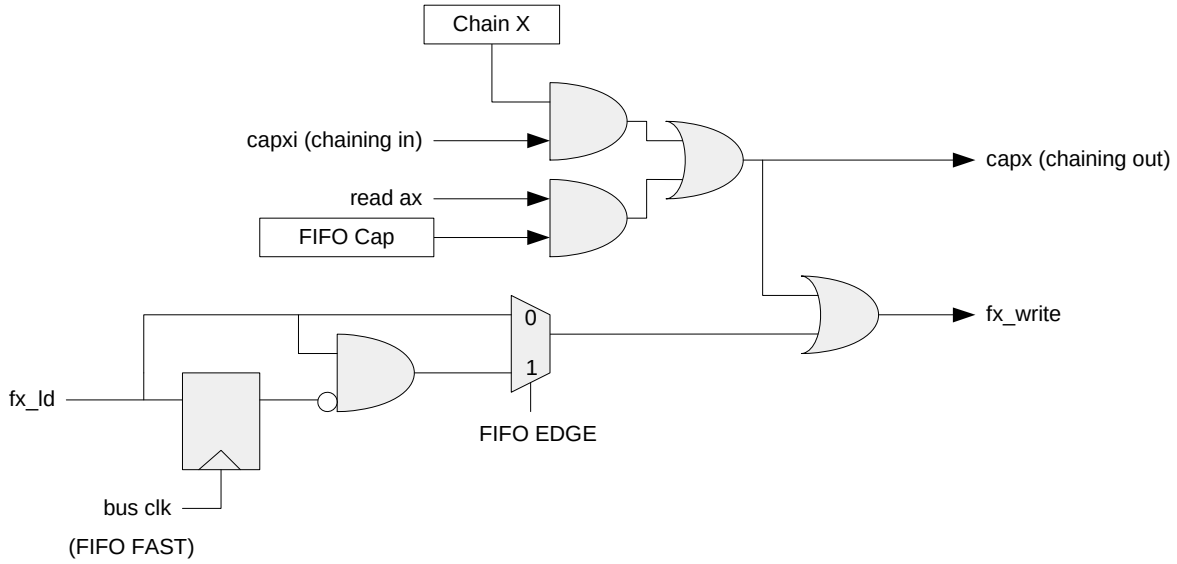
The routed FX_LD signal, which generates a FIFO load, is ORed with the software capture signal; the results can be unpredictable when both hardware and software capture are used at the same time. As a general rule these functions should be mutually exclusive; however, hardware and software capture can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

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

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

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

Figure 23-12. Software Capture Configuration



FIFO Control Bits

There are four bits in the Auxiliary Control register that may be used to control the FIFO during normal operation.

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

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

Table 23-5. FIFO Level Control Bits

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

FIFO Asynchronous Operation

Figure 23-13 illustrates the concept of asynchronous FIFO operation. As an example, assume F0 is set for input mode and F1 is set for output mode, which is a typical configuration for TX and RX registers.

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

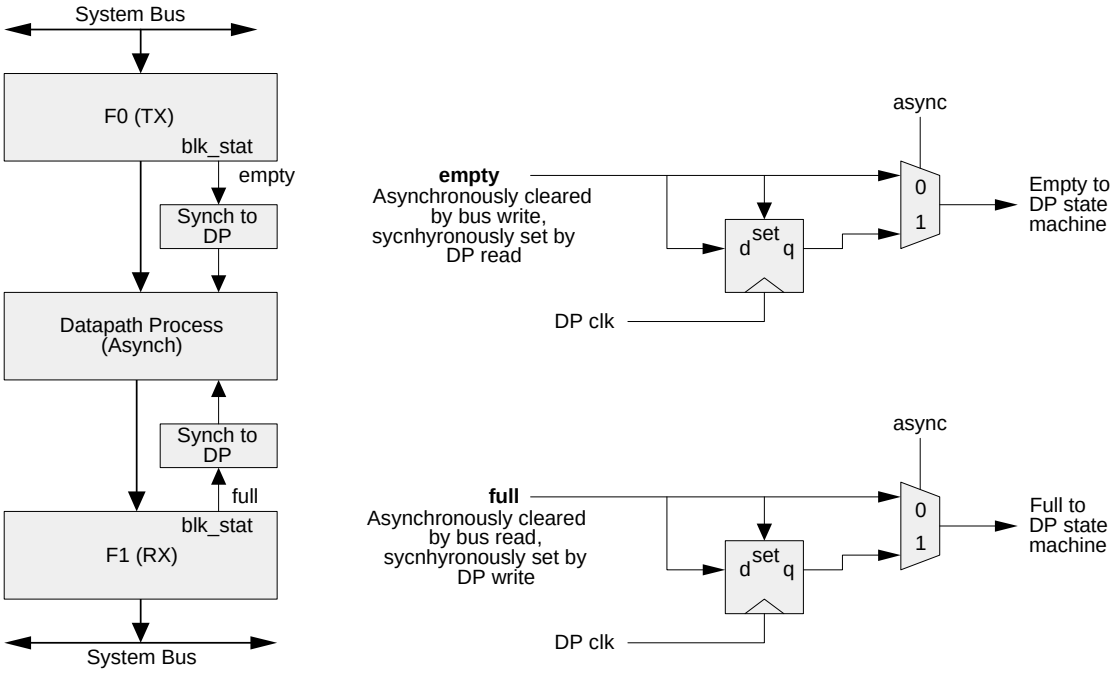
On the RX side, the datapath state machine uses “full” to determine whether there is a space left to write to the FIFO. Full is set synchronously to the DP state machine, but is cleared asynchronously due to a bus read. When cleared, the status is synchronized back to the DP state machine.

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

FIFO Overflow Operation

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

Figure 23-13. FIFO Asynchronous Operation



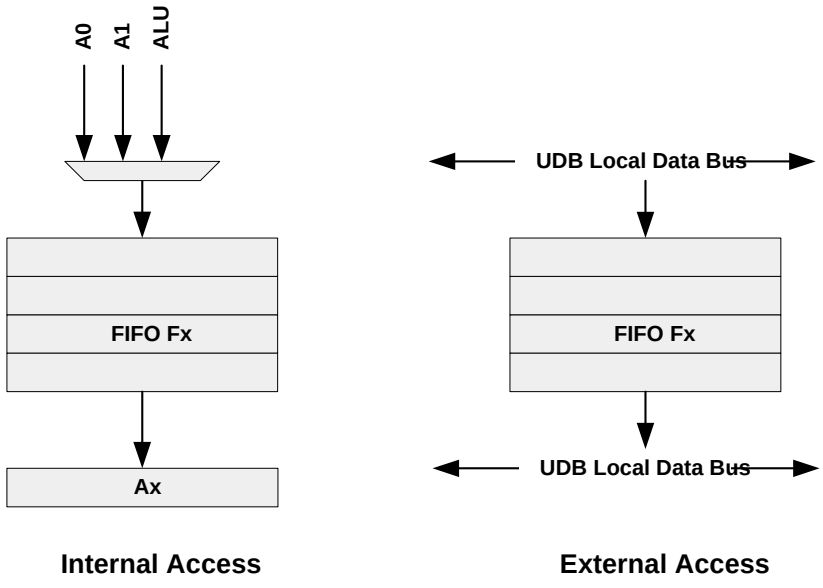
FIFO Clock Inversion Option

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

FIFO Dynamic Control

Normally, the FIFOs are configured statically in either input or output mode. As an alternative, each FIFO can be configured into a mode where the direction is controlled dynamically, that is, by routed signals. One configuration bit per FIFO (Fx DYN) enables the mode. Figure 23-14 on page 162 shows the configurations available in dynamic FIFO mode.

Figure 23-14. FIFO Dynamic Mode



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

In external access mode, the CPU or DMA can both read and write the FIFO.

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

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

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

Table 23-6. FIFO Status

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

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

23.3.2.3 FIFO Status

There are four FIFO status signals, two for each FIFO: fifo0_bus_stat, fifo0_blk_stat, fifo1_bus_stat, and fifo1_blk_stat. The meaning of these signals depends on the direction of the given FIFO, which is determined by static configuration. FIFO status is covered in detail in section 23.3.2.2 Datapath FIFOs on page 156.

23.3.2.4 Datapath ALU

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

Arithmetic and Logic Operation

The ALU functions, which are configured dynamically by the RAM control store, are shown in the following table:

Table 23-7. ALU Functions

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

Carry In

The carry in is used in arithmetic operations. There is a default carry in value for certain functions, as shown in Table 23-8.

Table 23-8. Carry In Functions

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

In addition to this default arithmetic mode for carry operation, there are three additional carry options. The CI SELA and CI SELB configuration bits determine the carry in for a given cycle. Dynamic configuration RAM selects either the A or B configuration on a cycle-by-cycle basis. The options are defined in Table 23-9.

Table 23-9. Additional Carry In Functions

CI SEL A CI SEL B	Carry Mode	Description
00	Default	Default arithmetic mode as described in Table 23-8.
01	Registered	Carry Flag, result of the carry from the previous cycle. This mode is used to implement add with carry and subtract with borrow operations. It can be used in successive cycles to emulate a double precision operation.
10	Routed	Carry is generated elsewhere and routed to this input. This mode can be used to implement controllable counters.
11	Chained	Carry is chained from the previous datapath. This mode can be used to implement single cycle operations of higher precision involving two or more datapaths.

When a routed carry is used, the meaning with respect to each arithmetic function is shown in Table 23-10. Note that in the case of the decrement and subtract functions, the carry is active low (inverted).

Table 23-10. Routed Carry In Functions

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

Carry Out

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

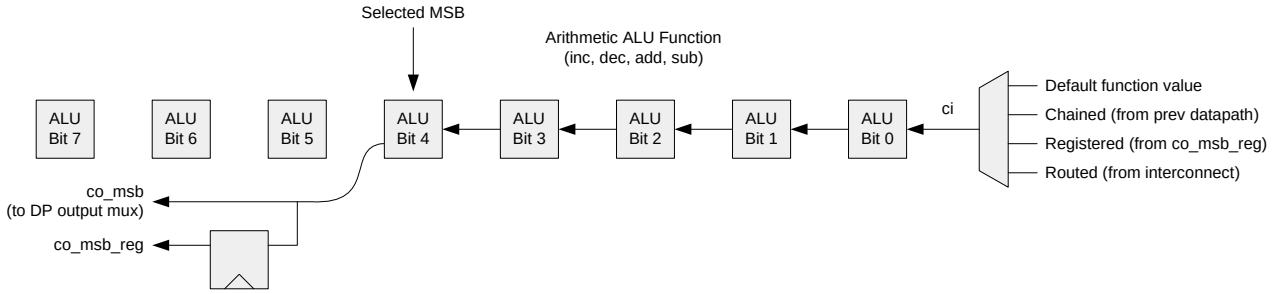
Table 23-11. Carry Out Functions

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

Carry Structure

Options for carry in, and for MSB selection for carry out generation, are shown in Figure 23-15 on page 164. The registered carry out value may be selected as the carry in for a subsequent arithmetic operation. This feature can be used to implement higher precision functions in multiple cycles.

Figure 23-15. Carry Operation



Shift Operation

The shift operation occurs independently of the ALU operation, according to Table 23-12

Table 23-12. Shift Operation Functions

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

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

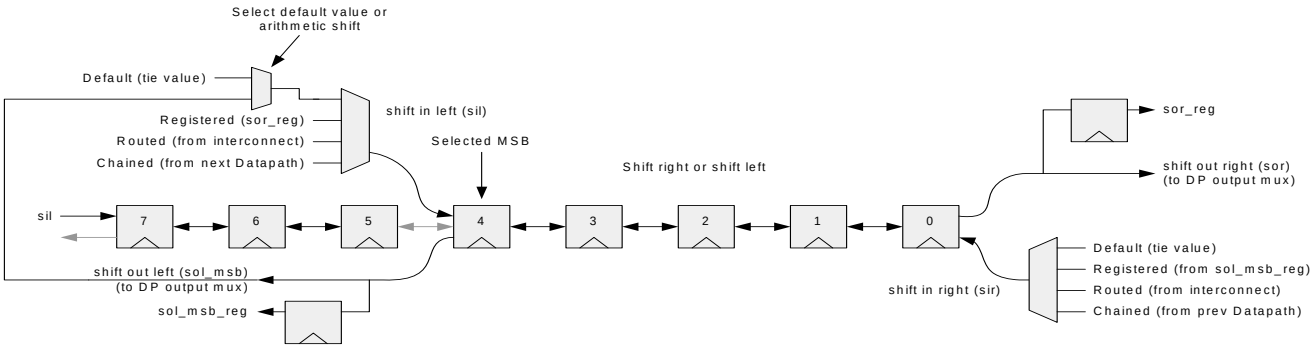
The SI SELA and SI SELB configuration bits determine the shift in data for a given operation. Dynamic configuration RAM selects the A or B configuration on a cycle-by-cycle basis. Shift in data is only valid for left and right shift; it is not used for pass and nibble swap. The selections and usage apply to both left and right shift directions and are shown in Table 23-13.

Table 23-13. Shift In Functions

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

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

Figure 23-16. Shift Operation



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

ALU Masking Operation

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

23.3.2.5 Datapath Inputs and Multiplexing

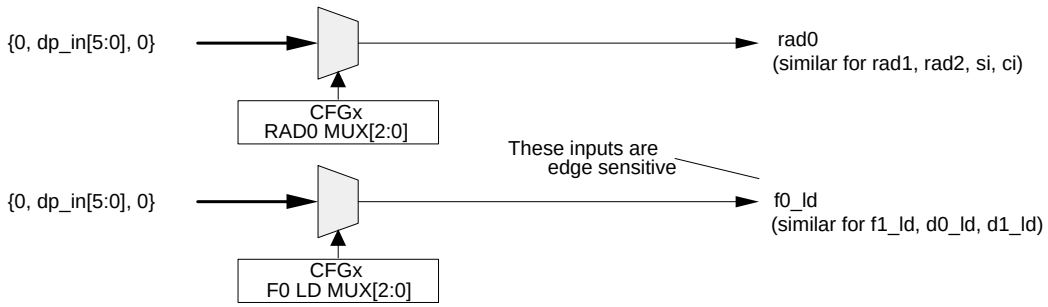
The datapath has a total of nine inputs as shown in Table 24-16, including six inputs from the channel routing. These consist of the configuration RAM address, FIFO and data register load control signals, and the data inputs shift in and carry in.

Table 23-14. Datapath Inputs

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

As shown in Figure 23-17, each input has a 6-to-1 multiplexer, therefore, all inputs are permutable. Inputs are handled in one of two ways, either level sensitive or edge sensitive. RAM address, shift in and data in values are level sensitive; FIFO and data register load signals are edge sensitive.

Figure 23-17. Datapath Input Select



23.3.2.6 CRC/PRS Support

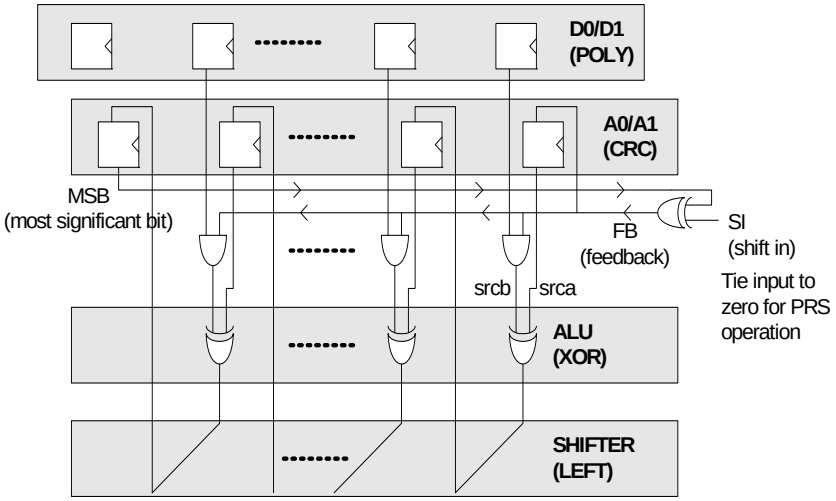
The datapath can support CRC and PRS generation. Chaining signals are routed between datapath blocks to support CRC/PRS bit lengths of longer than eight bits.

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

Figure 23-18 shows the structural configuration for the CRC operation. The PRS configuration is identical except that the shift in (SI) is tied to '0'. In the PRS configuration, D0 or D1 contain the polynomial value, while A0 or A1 contain the initial (seed) value and the CRC residual value at the end of the computation.

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

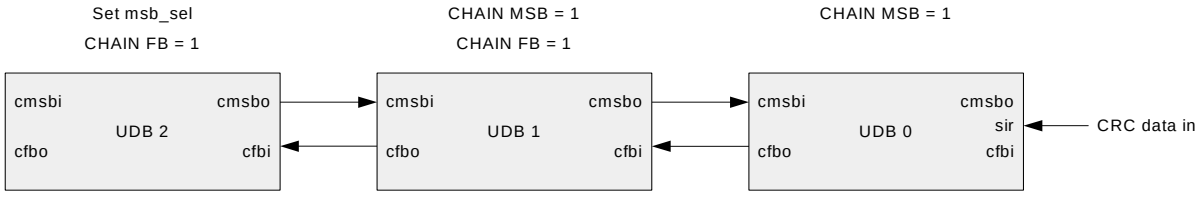
Figure 23-18. CRC Functional Structure



CRC/PRS Chaining

Figure 23-19 illustrates an example of CRC/PRS chaining across three UDBs. This scenario can support a 17- to 24-bit operation. The chaining control bits are set according to the position of the datapath in the chain as shown.

Figure 23-19. CRC/PRS Chaining Configuration



How the CRC/PRS feedback signal (cfbo, cfbi) is chained:

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

How the CRC/PRS MSB signal (cmsbo, cmsbi) is chained:

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

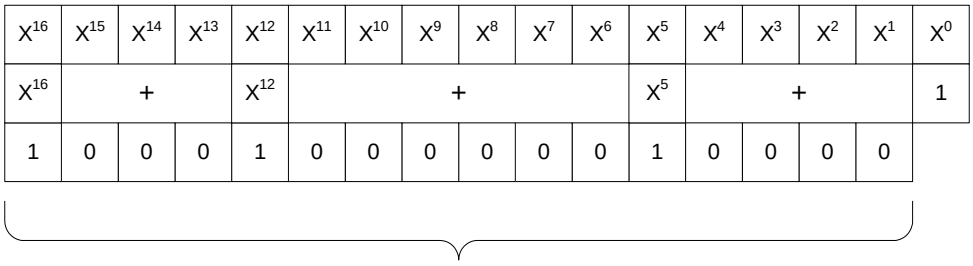
CRC/PRS Polynomial Specification

As an example of how to configure the polynomial for programming into the associated D0/D1 register, consider the CCITT CRC-16 polynomial, which is defined as $x^{16} + x^{12} + x^5 + 1$. The method for deriving the data format from the polynomial is shown in Figure 23-20.

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

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

Figure 23-20. CCITT CRC16 Polynomial Format



Example CRC/PRS Configuration

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

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

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

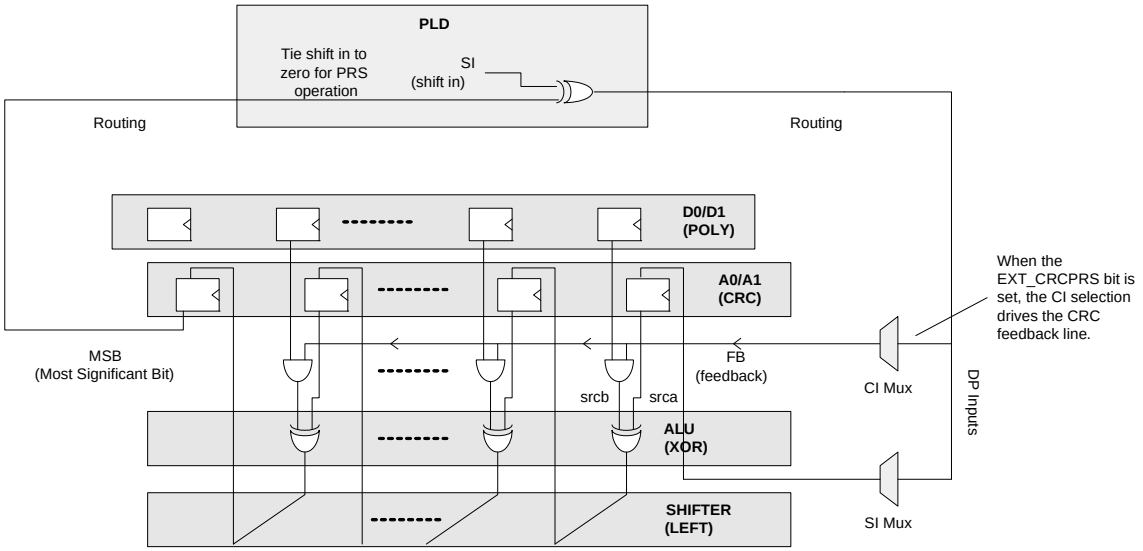
Clocking the UDB with this configuration generates the required CRC or outputs the MSB, which may be output to the routing for the PRS sequence.

External CRC/PRS Mode

A static configuration bit can be set (EXT CRCPRS) to enable support for external computation of a CRC or PRS. As shown in Figure 23-21, computation of the CRC feedback is done in a PLD block. When the bit is set, the CRC feedback signal is driven directly from the CI (Carry In) datapath input selection mux, bypassing the internal computation. The figure shows a simple configuration that supports up to an 8-bit CRC or PRS. Normally the built-in circuitry is used, but this feature gives the capability for more elaborate configurations, such as up to a 16-bit CRC/PRS function in one UDB using time division multiplexing.

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

Figure 23-21. External CRC/PRS Mode



23.3.2.7 Datapath Outputs and Multiplexing

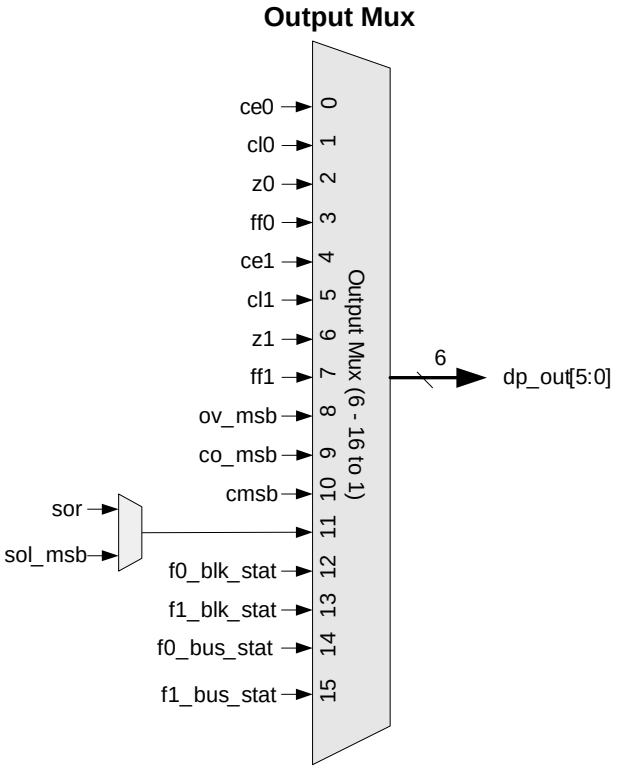
Conditions are generated from the registered accumulator values, ALU outputs, and FIFO status. These conditions can be driven to the digital routing for use in other UDB blocks, for use as interrupts or DMA requests, or to I/O pins. The 16 possible conditions are shown in the following table:

Table 23-15. Datapath Condition Generation

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

There are a total of six datapath outputs. As shown in Figure 23-22, each output has a 16-1 multiplexer that allows any of these 16 signals to be routed to any of the datapath outputs.

Figure 23-22. Output Mux Connections



Compares

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

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

Table 23-16. Compare Configuration

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

Compare 0 and Compare 1 are independently chainable to the conditions generated in the previous datapath (in addressing order). Whether to chain compares or not is statically specified in UDB configuration registers. Figure 23-23 illustrates compare equal chaining, which is just an ANDing of the compare equal in this block with the chained input from the previous block.

Figure 23-23. Compare Equal Chaining

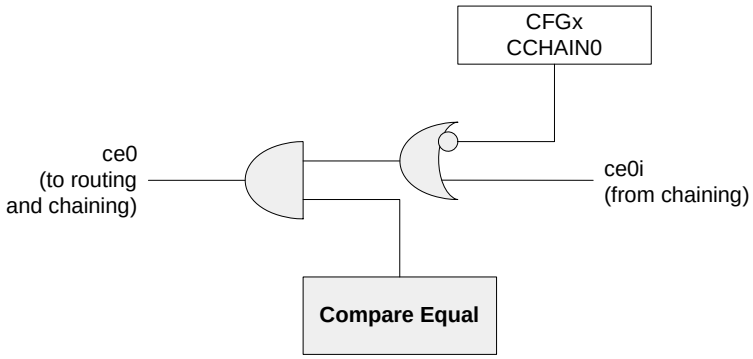
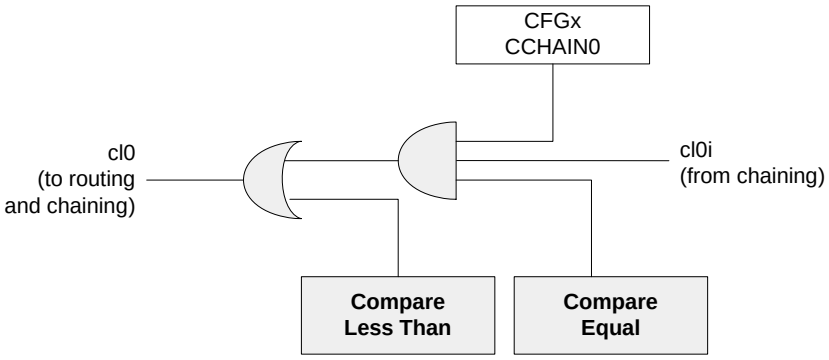


Figure 23-24 illustrates compare less than chaining. In this case, the “less than” is formed by the compare less than output in this block, which is unconditional. This is ORed with the condition where this block is equal, and the chained input from the previous block is asserted as less than.

Figure 23-24. Compare Less Than Chaining



All Zeros and All Ones Detect

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

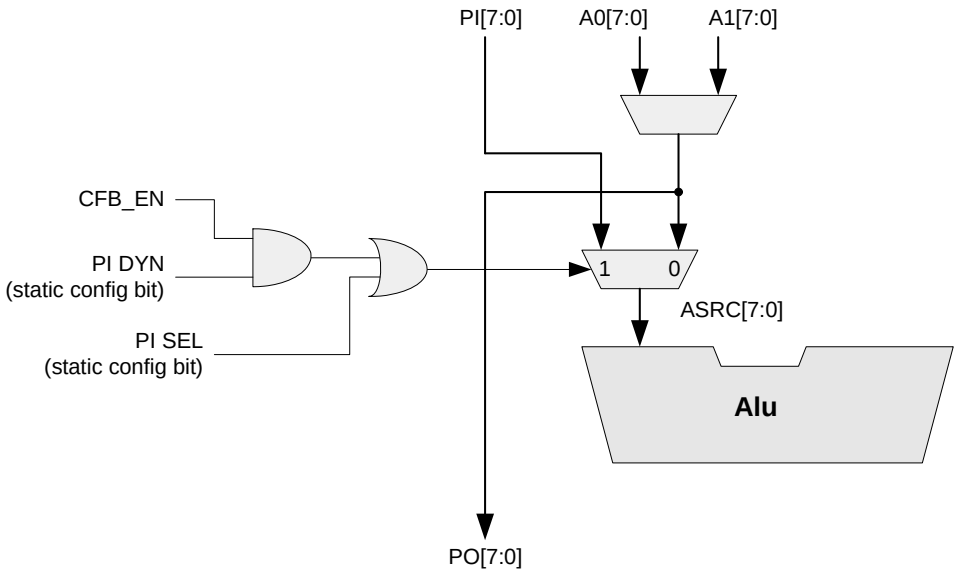
Overflow

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

23.3.2.8 Datapath Parallel Inputs and Outputs

As shown in Figure 23-25, the datapath Parallel In (PI) and Parallel Out (PO) signals give limited capability to bring routed data into and out of the Datapath. Parallel Out signals are always available for routing as the ALU asrc selection between A0 and A1.

Figure 23-25. Datapath Parallel In/Out

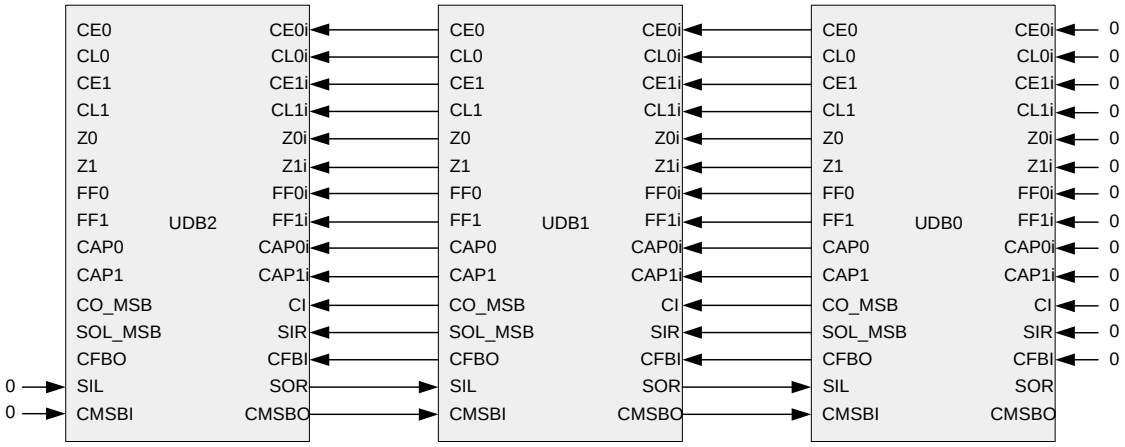


Parallel In needs to be selected for input to the ALU. There are two options, static operation or dynamic operation. For static operation, the PI SEL bit forces the ALU asrc to be PI. The PI DYN bit is used to enable the PI dynamic operation. When it is enabled, and assuming the PI SEL is 0, the PI multiplexer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

23.3.2.9 Datapath Chaining

Each datapath block contains an 8-bit ALU, which is designed to chain carries, shifted data, capture triggers, and conditional signals to the nearest neighbor datapaths, to create higher precision arithmetic functions and shifters. These chaining signals, which are dedicated signals, allow single-cycle 16-, 24- and 32-bit functions to be efficiently implemented without the timing uncertainty of channel routing resources. In addition, the capture chaining supports the ability to perform an atomic read of the accumulators in chained blocks. As shown in Figure 23-21, all generated conditional and capture signals chain in the direction of least significant to most significant blocks. Shift left also chains from least to most significant. Shift right chains from most to least significant. The CRC/PRS chaining signal for feedback chains least to most significant; the MSB output chains from most to least significant.

Figure 23-26. Datapath Chaining Flow

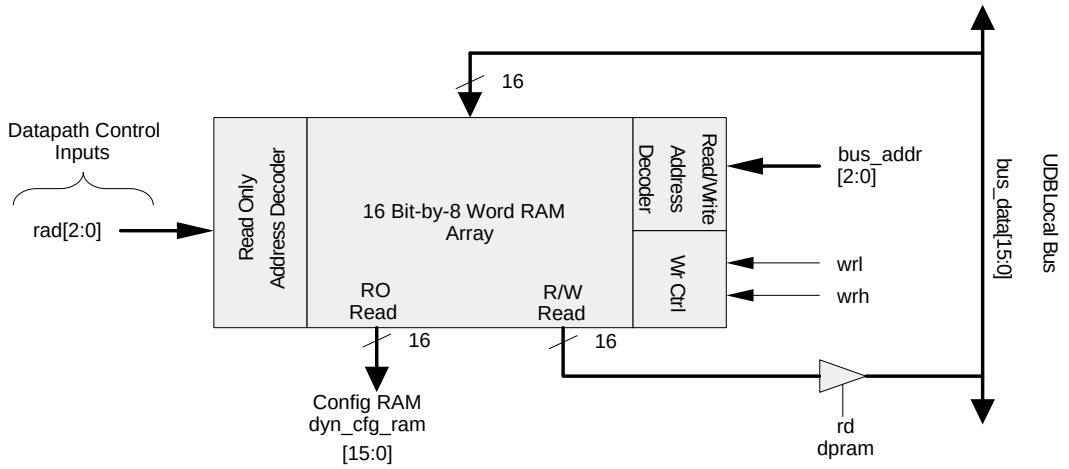


23.3.2.10 Dynamic Configuration RAM

Each datapath contains a 16 bit-by-8 word dynamic configuration RAM, which is shown in Figure 23-27. The purpose of this RAM is to control the datapath configuration bits on a cycle-by-cycle basis, based on the clock selected for that datapath. This RAM has synchronous read and write ports for purposes of loading the configuration via the system bus.

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

Figure 23-27. Configuration RAM I/O



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

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

Register	Address	7	6	5	4	3	2	1	0
CFGRAM	60h - 6Eh (Even)	A0 WRSRC[1:0]		A1 WRSRC[1:0]		CFB EN	CI SEL	SI SEL	CMPSEL

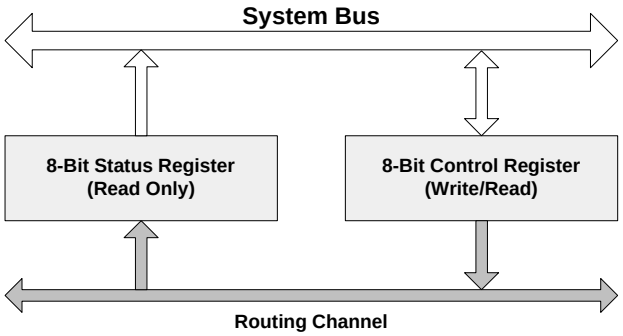
Table 23-17. Dynamic Configuration Quick Reference

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

23.3.3 Status and Control Module

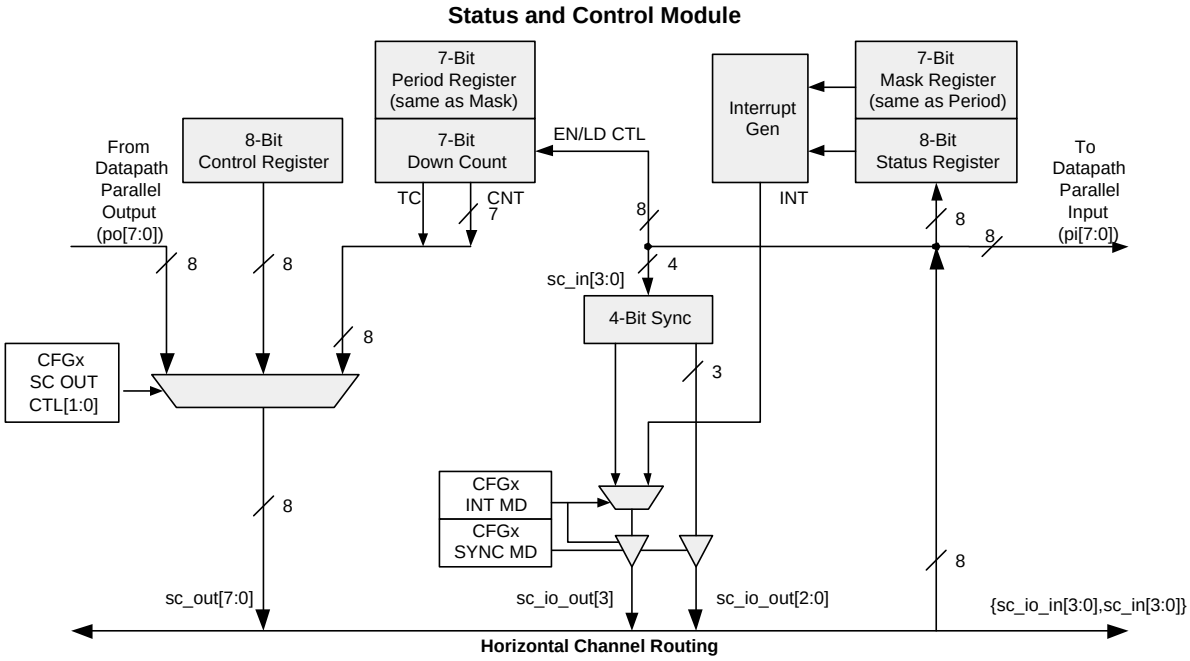
A high level view of the Status and Control module is shown in Figure 23-28. The Control register drives into the routing to provide firmware control inputs to UDB operation. The Status register read from routing provides firmware a method of monitoring the state of UDB operation.

Figure 23-28. Status and Control Registers



A more detailed view of the Status and Control module is shown in Figure 23-29. The primary purpose of this block is to coordinate CPU firmware interaction with internal UDB operation. However, due to its rich connectivity to the routing matrix, this block may be configured to perform other functions.

Figure 23-29. Status and Control Module



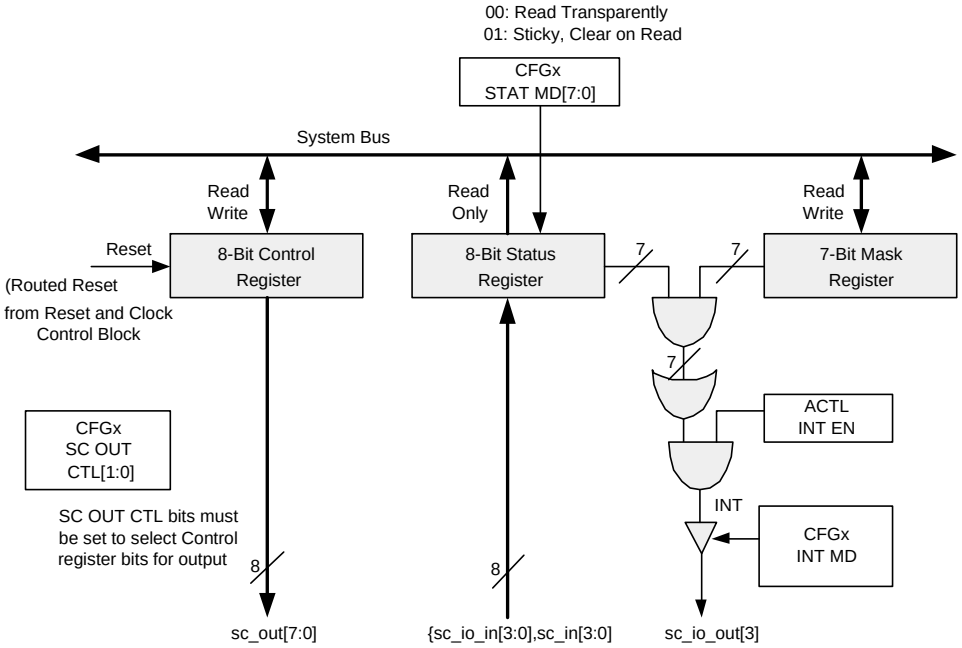
Modes of operation include:

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

23.3.3.1 Status and Control Mode

When operating in status and control mode, this module functions as a status register, interrupt mask register, and control register in the configuration shown in Figure 23-30 on page 174.

Figure 23-30. Status and Control Operation



Status Register Operation

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

Table 23-18. Status Register

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

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

Transparent Status Read

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

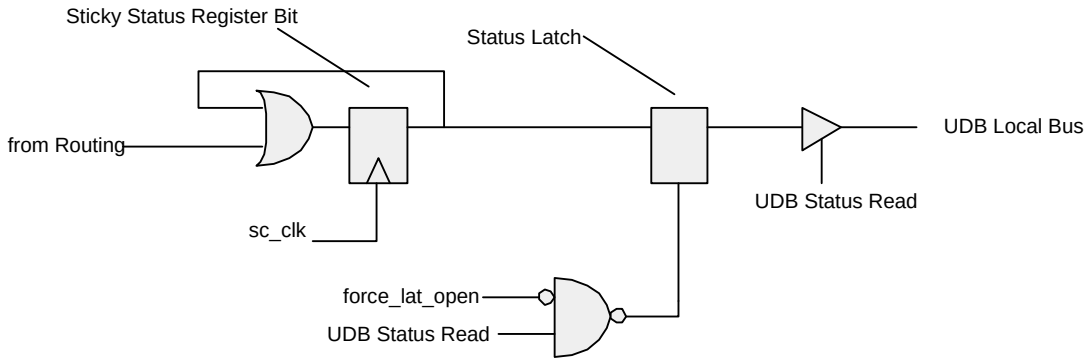
Sticky Status, with Clear on Read

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

Status Latching During Read

Figure 23-31 on page 175 shows the structure of the status read logic. The sticky status register is followed by a latch, which latches the status register data and holds it stable during the duration of the read cycle, regardless of the number of wait states in a given read.

Figure 23-31. Status Read Logic



Interrupt Generation

In most functions, interrupt generation is tied to the setting of status bits. As shown in Figure 23-31, this feature is built into the status register logic as the masking and OR reduction of status. Only the lower 7 bits of status input can be used with the built-in interrupt generation circuitry. The most significant bit is typically used as the interrupt output and may be routed to the interrupt controller through the digital routing. In this configuration, the MSB of the status register is read as the state of the interrupt bit.

23.3.3.2 Control Register Operation

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

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

Control Register Operating Modes

There are three available modes that may be configured on a bit-by-bit basis. The configuration is controlled by the concatenation of the bits of the two 8-bit registers CTL_MD1[7:0] and CTL_MD0[7:0]. For example, {CTL_MD1[0],CTL_MD0[0]} controls the mode for Control Register bit 0, as shown in Figure 23-19.

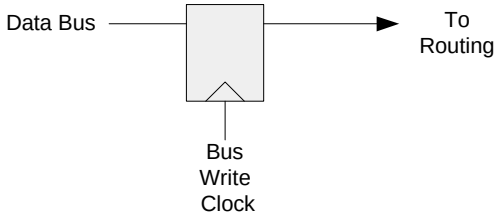
Table 23-19. Mode for Control Register Bit 0

CTL MD	Description
00	Direct mode
01	Sync mode
10	(reserved)
11	Pulse mode

Control Register Direct Mode

The default mode is Direct mode. As shown in Figure 23-32, when the Control Register is written by the CPU or DMA the output of the control register is driven directly to the routing on that write cycle.

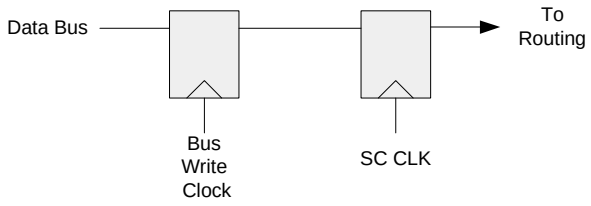
Figure 23-32. Control Register Direct Mode



Control Register Sync Mode

In Sync mode, as shown in Figure 23-33, the control register output is driven by a re-sampling register clocked by the currently selected Status and Control (SC) clock. This allows the timing of the output to be controlled by the selected SC clock, rather than the bus clock.

Figure 23-33. Control Register Sync Mode



Control Register Pulse Mode

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

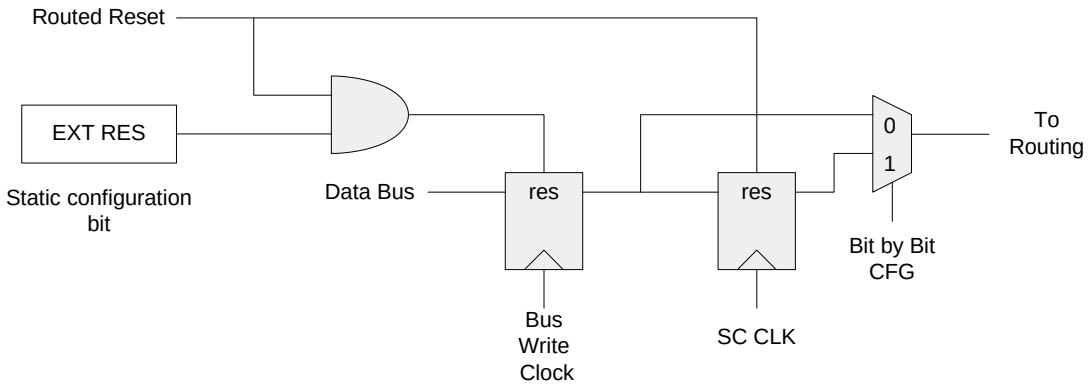
With this mode of operation, firmware can write a ‘1’ to a control register bit to generate a pulse. It is then read back

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

Control Register Reset

The control register has two reset modes, controlled by the EXT RES configuration bit, as shown in Figure 23-34. When EXT RES is 0 (the default) then in sync or pulse mode the routed reset input resets the synced output but not the actual control bit. When EXT RES is 1 then the routed reset input resets both the control bit and the synced output.

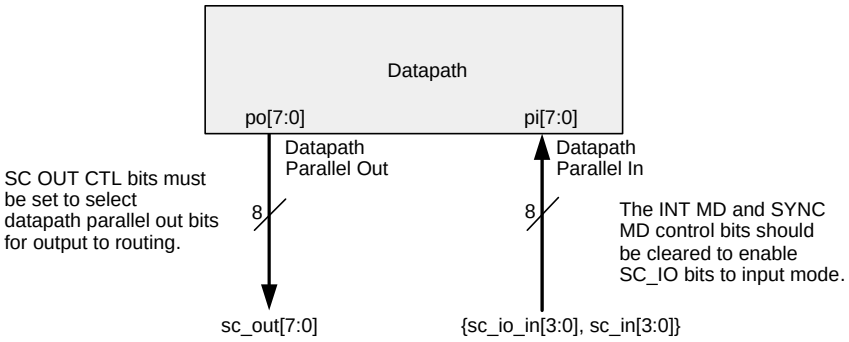
Figure 23-34. Control Register Reset



23.3.3.3 Parallel Input/Output Mode

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

Figure 23-35. Parallel Input/Output Mode



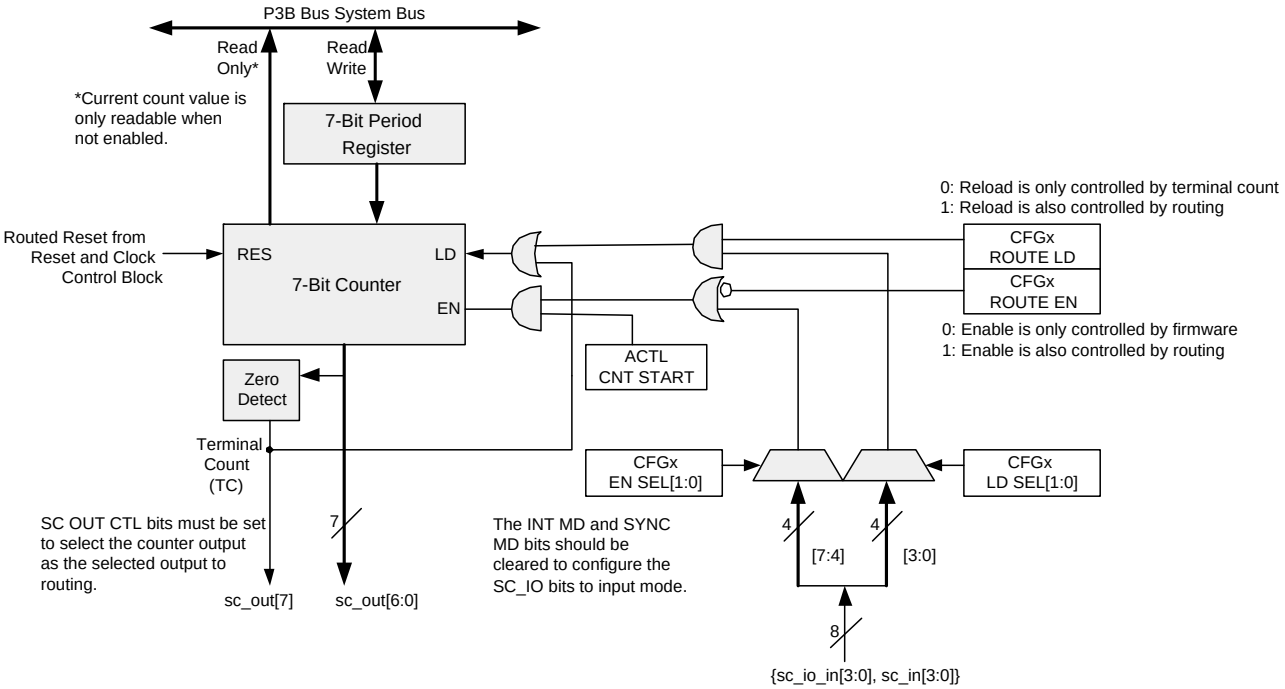
23.3.3.4 Counter Mode

As shown in Figure 23-36, when the block is in counter mode, a 7-bit down counter is exposed for use by UDB internal operation or firmware applications. This counter has the following features:

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

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

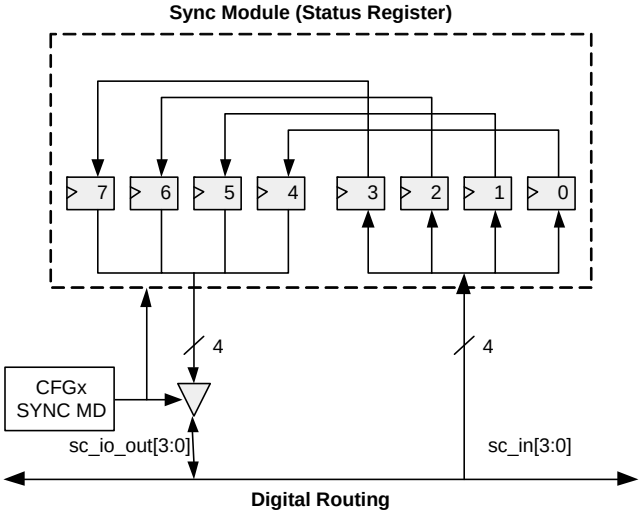
Figure 23-36. Counter Mode



23.3.3.5 Sync Mode

As shown in Figure 23-37, the status register can operate as a 4-bit double synchronizer, clocked by the current SC_CLK, when the SYNC MD bit is set. This mode may be used to implement local synchronization of asynchronous signals, such as GPIO inputs. When enabled, the signals to be synchronized are selected from SC_IN[3:0], the outputs are driven to the SC_IO_OUT[3:0] pins, and SYNC MD automatically puts the SC_IO pins into output mode. When in this mode, the normal operation of the status register is not available, and the status sticky bit mode is forced off, regardless of the control settings for this mode. The control register is not affected by the mode. The counter can still be used with limitations. No dynamic inputs (LD/EN) to the counter can be enabled in this mode.

Figure 23-37. Sync Mode



23.3.3.6 Status and Control Clocking

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

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

The clock for this is allocated in the reset and clock control module. See 23.3.4 Reset and Clock Control Module on page 179.

23.3.3.7 Auxiliary Control Register

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

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

FIFO0 Clear, FIFO1 Clear

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

FIFO0 Level, FIFO1 Level

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

Table 23-20. FIFO Level Control Bits

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

Interrupt Enable

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

Count Start

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

23.3.3.8 Status and Control Register Summary

The following table summarizes the function of the status and control registers. Note that the control and mask registers are shared with the count and period registers and the meaning of these registers is mode dependent.

Table 23-21. Status, Control Register Function Summary

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

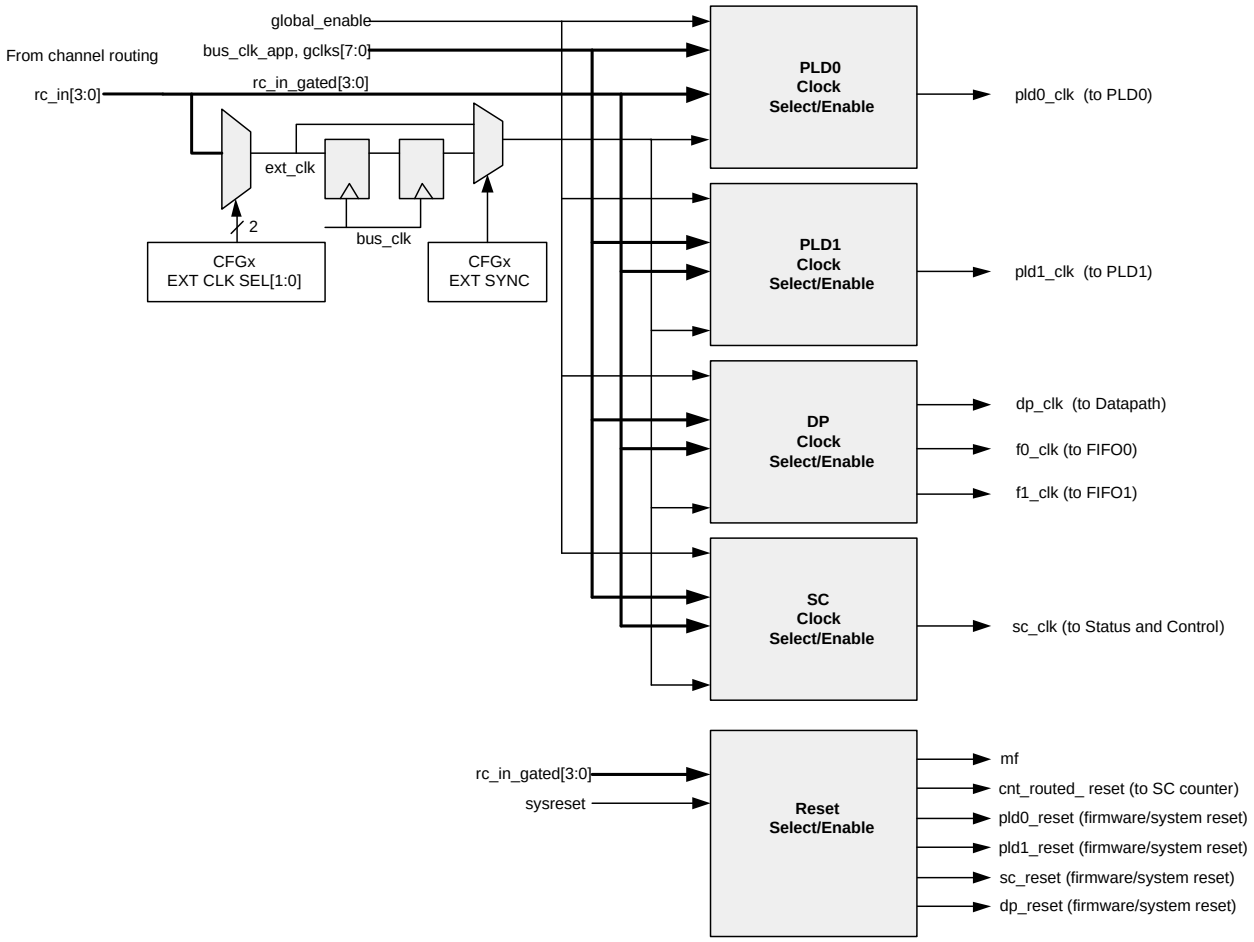
a. Note that in counter mode, the mask register is operating as a period register and cannot function as a mask register. Therefore, interrupt output is not available when counter mode is enabled.
b. Note that in SYNC mode, the status register function is not available, and therefore, the mask register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

23.3.4 Reset and Clock Control Module

The primary function of the reset and clock block is to select a clock from the available global system clocks or bus clock for each of the PLDs, the datapath, and the status and control block. It also supplies dynamic and firmware-based resets to the UDB blocks. As shown in Figure 23-38, there are four clock control blocks, and one reset block. Four inputs are available for use from the routing matrix (RC_IN[3:0]). Each clock control block can select a clock enable source from these routing inputs, and there is also a multiplexer to select one of the routing inputs to be used as an external clock source. As shown, the external clock source selection can be optionally synchronized. There are a total of 10 clocks that can be selected for each UDB component: eight global digital clocks, bus clock, and the selected external clock (ext clk). Any of the routed input signals (rc_in) can be used as either a level sensitive or edge sensitive enable. The reset function of this block provides a routed reset for the PLD blocks and SC counter, and a firmware reset capability to each block to support reconfiguration.

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

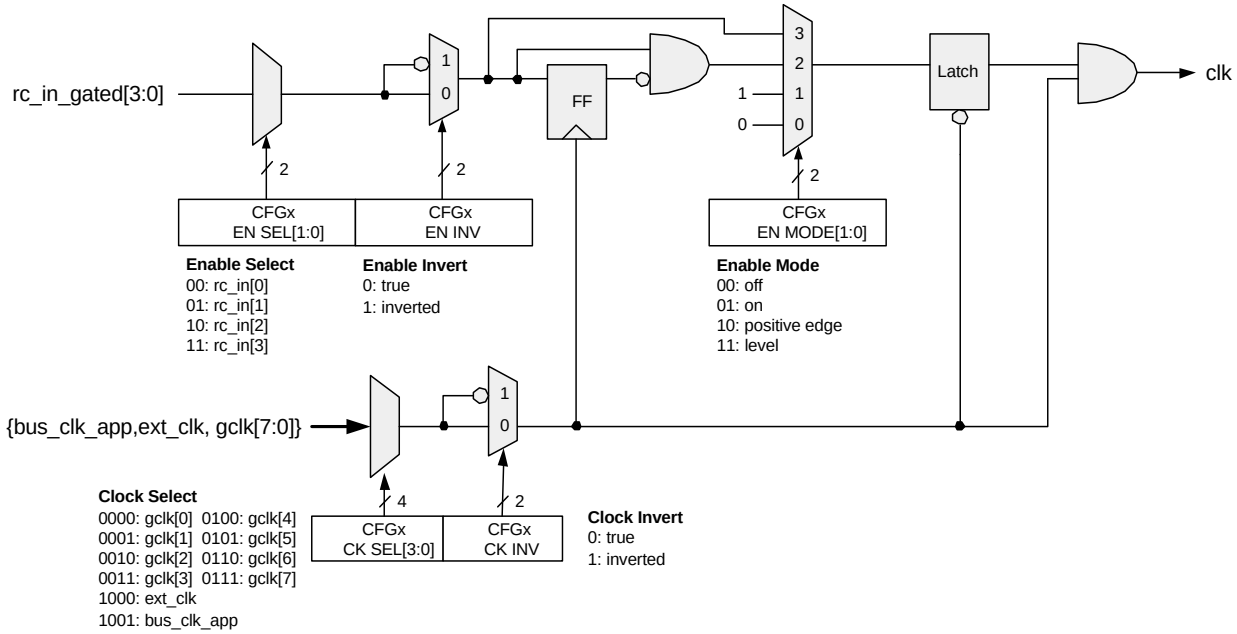
Figure 23-38. Reset and Clock Control



23.3.4.1 Clock Control

Figure 23-39 illustrates one instance of the clock selection and enable circuit. There are four of these circuits in each UDB: one for each of the PLD blocks, one for the datapath, and one for the status and control block. The main components of this circuit are a global clock selection multiplexer, clock inversion, clock enable selection multiplexer, clock enable inversion, and edge detect logic.

Figure 23-39. Clock Select/Enable Control



Clock Selection

There are eight global digital clocks routed to all UDBs; any of these clocks may be selected. Global digital clocks are the output of user selectable clock dividers. See the Clocking System chapter on page 91. Another selection is bus clock, which is the highest frequency in the system. Called “bus_clk_app,” this signal is routed separately from the system bus clock. In addition, an external routing signal can be selected as a clock input to support direct-clocked functions such as SPI. Because application functions are mapped to arbitrary boundaries across UDBs, individual clock selection for each UDB subcomponent block supports a fine granularity of programming.

Clock Inversion

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

Clock Enable Selection

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

Clock Enable Inversion

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

Clock Enable Mode

By default, the clock enable is OFF. After configuring the target block operation, software can set the mode to one of the following using the CFGxEN MODE[1:0] register shown in Figure 23-39.

Table 23-22. Clock Enable Mode

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

Clock Enable Usage

There are two general usage scenarios for the clock enable.

Firmware Enable – It is assumed that most functions require a firmware clock enable to start and stop the function. The boundary of a function mapped into the UDB array is arbitrary - it can span multiple UDBs and/or portions of UDBs. Therefore, there must be a way to enable a given function atomically. This is typically implemented from a bit

in a control register routed to one or more clock enable inputs. This scenario also supports the case where applications require multiple, unrelated blocks to be enabled simultaneously.

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

Special FIFO Clocking

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

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

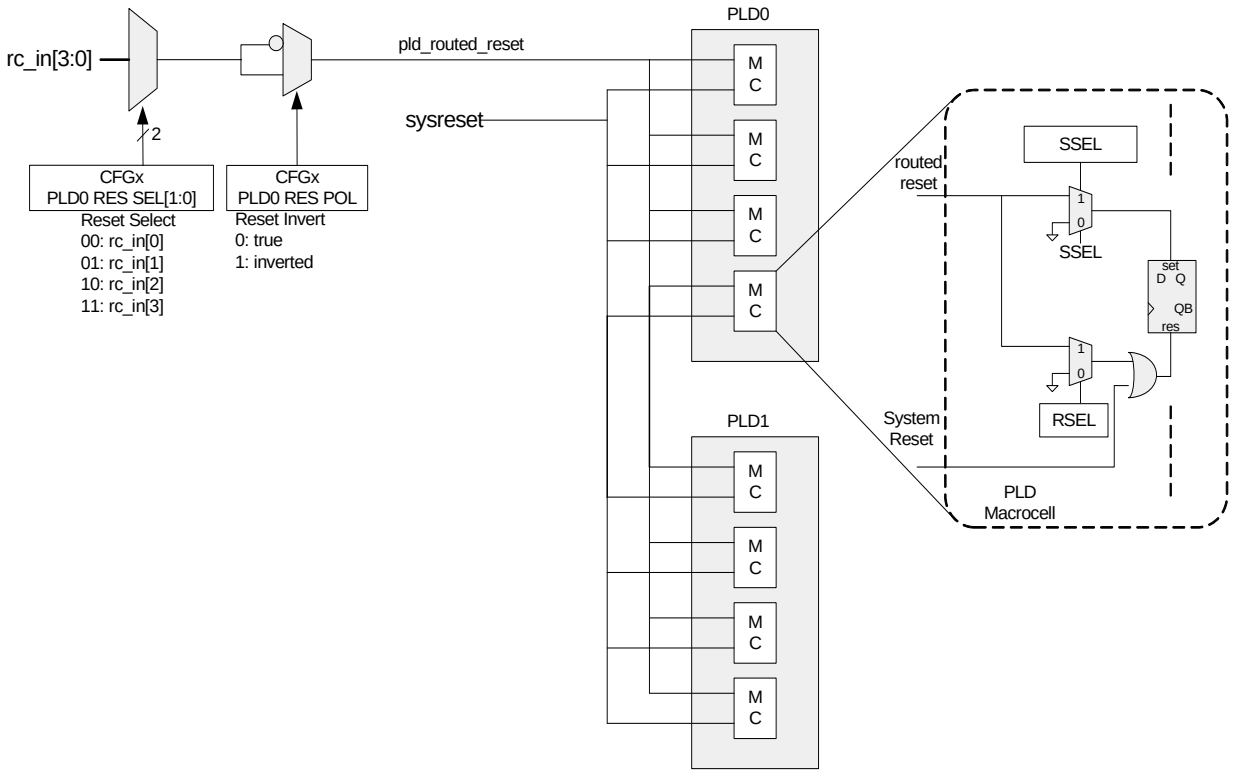
23.3.4.2 Reset Control

There are two modes of reset control: legacy mode and standard mode. The modes are controlled by the ALT RES bit in each UDB configuration register CFG31. The default for this bit is 0 (legacy mode); it is recommended that it be set to '1' for standard mode. Standard mode has greater granularity - routed resets can be used by individual blocks within the UDB. Contact Cypress for information about legacy mode reset.

PLD Reset Control

Figure 23-40 shows the PLD reset system.

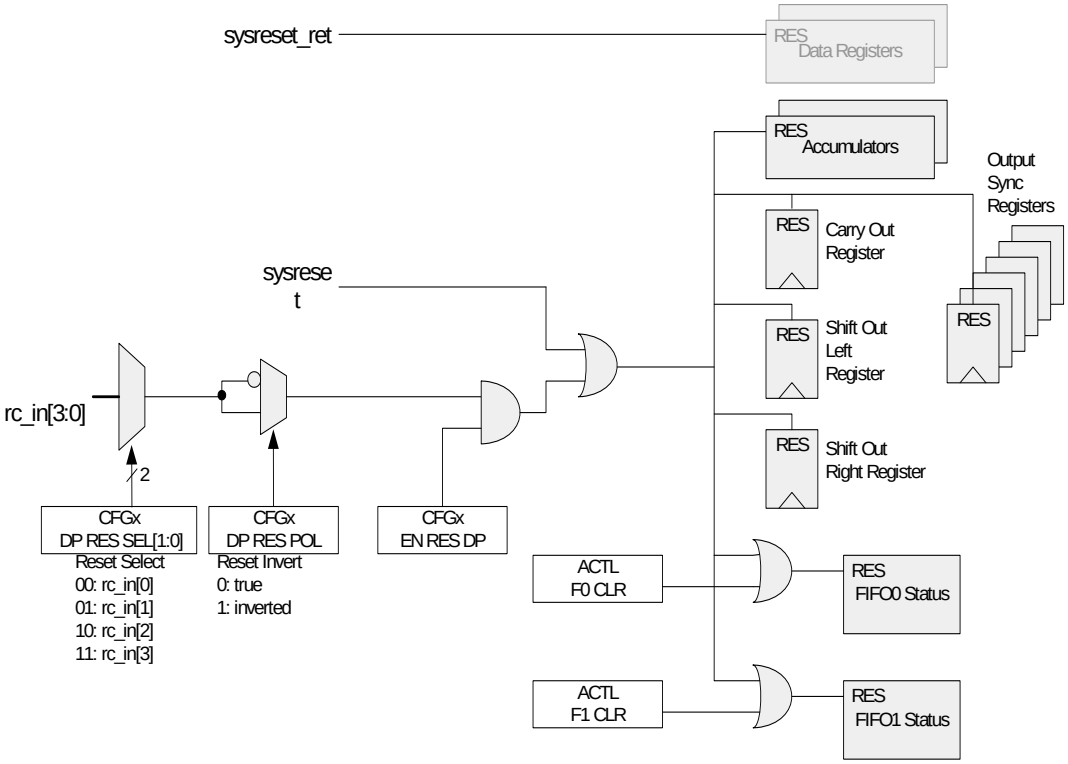
Figure 23-40. PLD Reset Structure



Datapath Reset Control

Figure 23-41 shows the datapath reset system. The routed reset is applied to all datapath registers and states. The FIFO data is unknown after reset because it is RAM based.

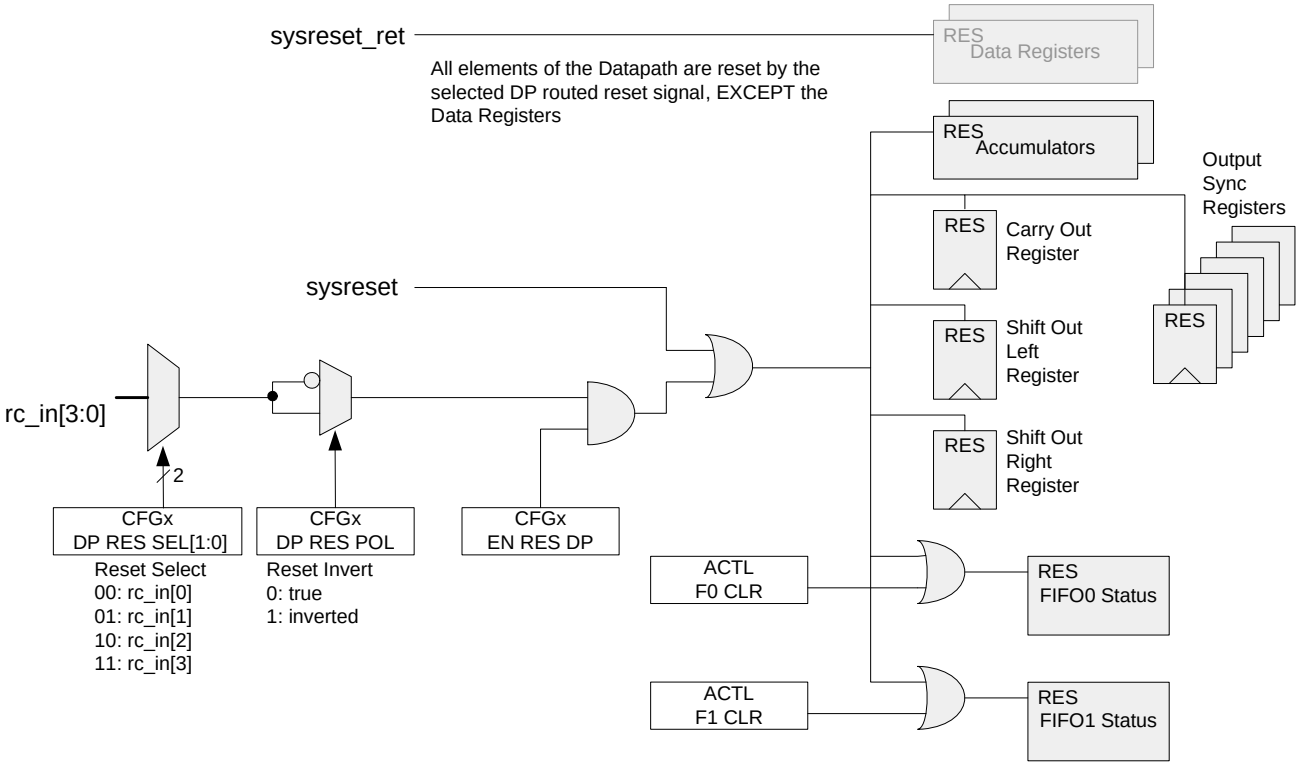
Figure 23-41. Datapath Reset Structure



Status and Control Reset Control

Figure 23-42 shows the status and control block reset system. The status and control/count registers share the routed reset; however, they are individually enabled.

Figure 23-42. Status and Control Reset Control



23.3.4.3 UDB POR Initialization

Register and State Initialization

Table 23-23. UDB POR State Initialization

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

Routing Initialization

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

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

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

23.3.5 UDB Addressing

There are three unique address spaces in a UDB pair:

- 8-Bit Working Registers – A bus master that can only access eight bits of data per bus cycle can use this address space to read or write any UDB working register. These are the registers with which the CPU and DMA interact during normal operation.
- 16-Bit Working Registers – A bus master with 16-bit capability, such as the DMA or the PSoC 5 Cortex-M3, can access 16 bits per bus cycle to facilitate the data transfer of functions that are inherently 16 bits or greater. Although this address space is mapped into a different area than the 8-bit space, the same registers are accessed, two registers at a time.
- 8- or 16-Bit Configuration Registers – These registers configure the UDB to perform a function. When configured, they are normally left in a static state during operation. These registers maintain their state through sleep.

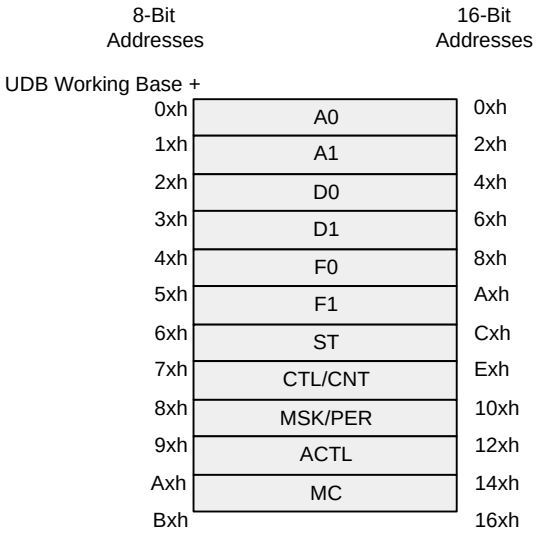
23.3.5.1 Working Register Address Space

Working registers are accessed during normal operation and include accumulators, data registers, FIFOs, status and control registers, mask register, and the auxiliary control register.

Figure 23-43 shows the register map for one UDB.

On the right in Figure 23-43 is the 16-bit address, which is always even aligned. The UDB number is 5 bits instead of 4, due to the even address alignment. The upper 4 bits is still the register number.

Figure 23-43. UDB Working Registers

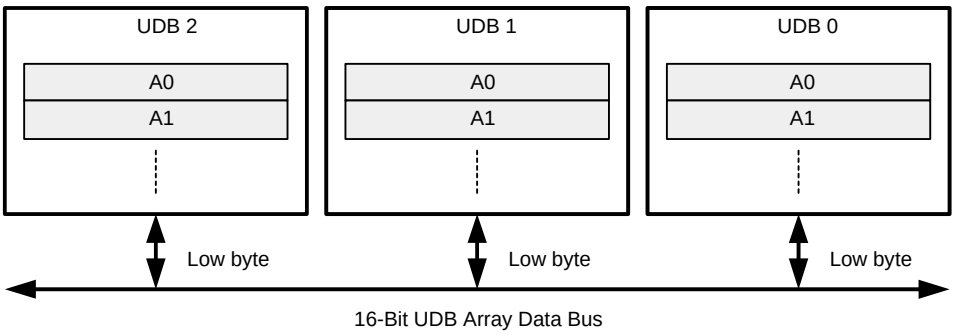


8-Bit Working Register Access

In this mode, all UDB registers are accessed on byte-aligned addresses. In 8-bit register access mode, as shown in Figure 23-44, all data bytes written to the UDBs are aligned with the low byte of the 16-bit UDB bus.

Only one byte at a time can be accessed in this mode.

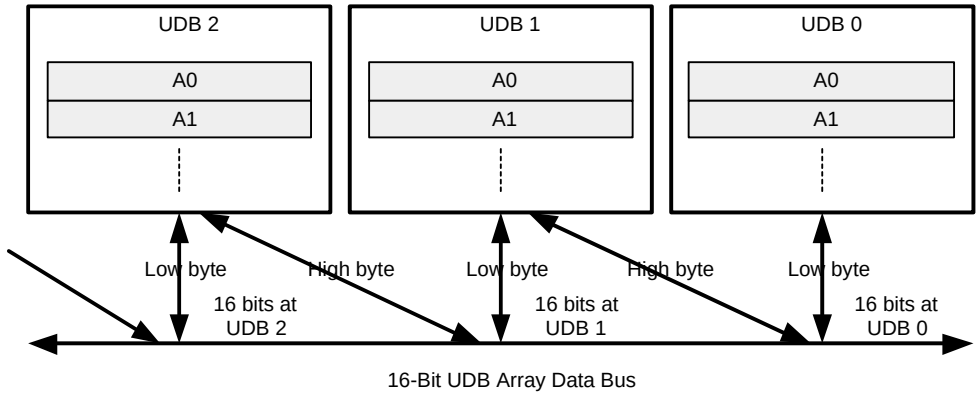
Figure 23-44. 8-Bit Working Register Access



16-Bit Working Register Address Space

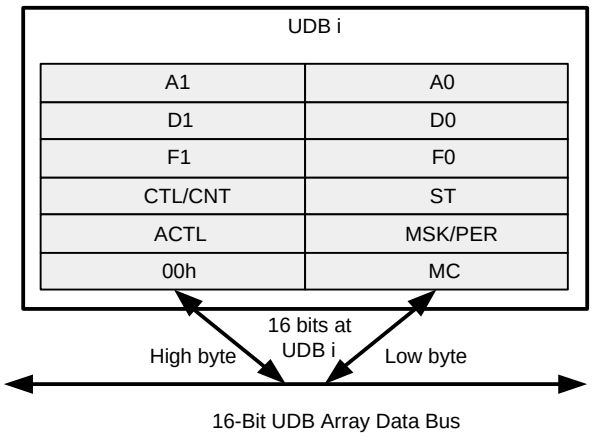
The 16-bit address space is designed for efficient DMA access and to provide support for CPU firmware access in processors that can support it, such as the Cortex-M3 in PSoC 5. There are two modes of 16-bit register access, the “default” mode and the “concat” mode. As shown in Figure 23-45, the default mode accesses a given register in UDB 'i' in the lower byte and the same register in UDB 'i+1' in the upper byte. This makes 16-bit data handling efficient in neighboring UDBs (address order) that are configured as a 16-bit function.

Figure 23-45. 16-Bit Working Register Default Access Mode



In concat mode, the registers of a single UDB are concatenated to form 16-bit registers as shown in Figure 23-46. In this mode, the 16-bit UDB array data bus has access to pairs of registers in the UDB in the format shown in the figure. For example, an access at A0 accesses A0 in the low byte and A1 in the high byte.

Figure 23-46. 16-Bit Working Register Concat Access Mode



There is a limitation in the use of DMA with respect to the 16-bit working register address space. It is inefficient for use when the function is greater than 16 bits. This is because the addressing overlaps, as shown in Table 23-24.

Table 23-24. Optimized Address Space for 16-Bit UDB Function

Address	Upper Byte Goes	Lower Byte Goes
0	UDB1	UDB0
2	UDB2	UDB1
4	UDB3	UDB2

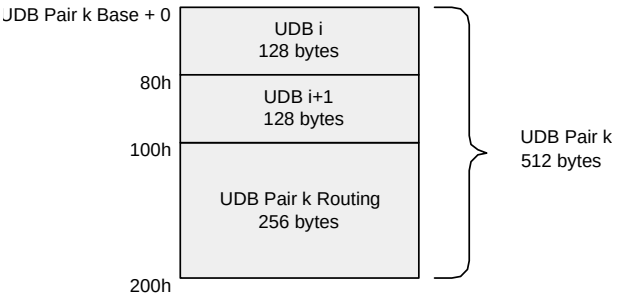
When the DMA transfers 16 bits to address 0, the lower and upper bytes are written to UDB0 and UDB1, respectively. On the next 16 bit DMA transfer at address 2, you overwrite the value in UDB1 with the lower byte of that transfer.

To avoid having to provide redundant data organization in memory buffers to support this addressing, it is recommended that 8-bit DMA transfers in the 8-bit working space be used for functions over 16 bits.

23.3.5.2 Configuration Register Address Space

Configuration is done at the UDB pair level. A UDB pair consists of two UDBs and an associated routing channel, as shown in Figure 23-47.

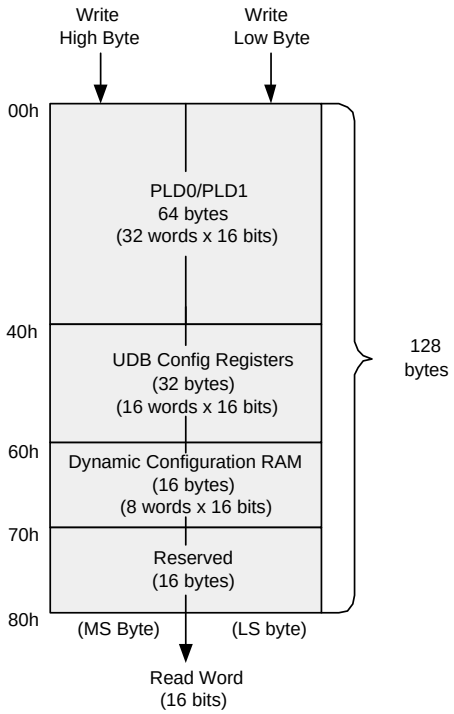
Figure 23-47. UDB Pair Configuration Address Map



23.3.5.3 UDB Configuration Address Space

Figure 23-48 shows the address map for configuration of a given UDB. As shown, this UDB configuration space is replicated for the two UDBs in the UDB pair. There are 128 bytes (7 bits of address) reserved for each UDB configuration, which is organized in 16-bit width. There are individual byte write enables for this address space to support both 16- and 8-bit access. Note that 16-bit access on odd boundaries is not supported. Reads always return 16 bits in configuration space, and the byte not required can be ignored.

Figure 23-48. UDB Configuration Address Space



23.3.5.4 Routing Configuration Address Space

UDB routing configuration consists of embedded RAM bits to control the state of transmission gate switches, segmentation, and input/output buffers. For more information, see the UDB Array and Digital System Interconnect chapter on page 191.

23.3.6 System Bus Access Coherency

UDB registers have dual access modes:

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

23.3.6.1 Simultaneous System Bus Access

The following table lists the possible simultaneous access events and required behavior:

Table 23-25. Simultaneous System Bus Access

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

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

23.3.6.2 Coherent Accumulator Access (Atomic Reads and Writes)

The UDB accumulators are the primary target of data computation. Therefore, reading these registers directly during normal operation gives an undefined result, as indicated in the table above). However, there is built-in support for atomic reads in the form of software capture, which is implemented across chained blocks. In this usage model, a read of the least significant accumulator transfers the data from all chained blocks to their associated FIFOs. This operation is explained in FIFO Software Capture Mode on page 160. Atomic writes to the accumulator can be implemented programmatically. Individual writes can be performed to the input FIFOs, and then the status signal of the last FIFO written can be routed to all associated blocks and simultaneously transfer the FIFO data into the Dx or Ax registers.

23.4 UDB Working Register Reference

All registers except the FIFO are cleared upon any system reset. The FIFO status is cleared, but FIFO data is random. These registers are not retention registers so they must be reset upon wakeup from a power cut-off sequence.

Register	8-Bit Address	16-Bit Address	7	6	5	4	3	2	1	0
Datapath Registers										
A0	0xh	00xh	A0[7:0] (Accumulator 0 Value)							
A1	1xh	02xh	A1[7:0] (Accumulator 1 Value)							
D0	2xh	04xh	D0[7:0] (Data Register 0)							
D1	3xh	06xh	D1[7:0] (Data Register 1)							
F0	4xh	08xh	F0[7:0] (FIFO 0)							
F1	5xh	0Axh	F1[7:0] (FIFO 1)							
Status and Control Registers										
ST	6xh	0Cxh	ST[7:0] (Status Register)							
CTL/CNT	7xh	0Exh	CTL[7:0] / CNT[6:0] (Control / Count Register)							
MSK/PER	8xh	10xh		MSK[6:0] / PER[6:0] (Interrupt Mask / Period Register)						
Auxiliary Control Register										
ACTL	9xh	12xh			CNT START	INT EN	FIFO1 LVL	FIFO0 LVL	FIFO1 CLR	FIFO0 CLR
PLD Macrocell Register										
MC	Axh	14xh	PLD1 MC[3:0]				PLD0 MC[3:0]			

24. UDB Array and Digital System Interconnect

This chapter describes the structure of the UDB Array and Digital System Interconnect (DSI). Universal Digital Blocks (UDBs) are organized in the form of a two-dimensional array with programmable interconnect provided by the DSI. In addition to connecting UDB components, the DSI routing also provides connection between other hardware resources on the device, such as I/O pins, interrupts, and fixed function blocks.

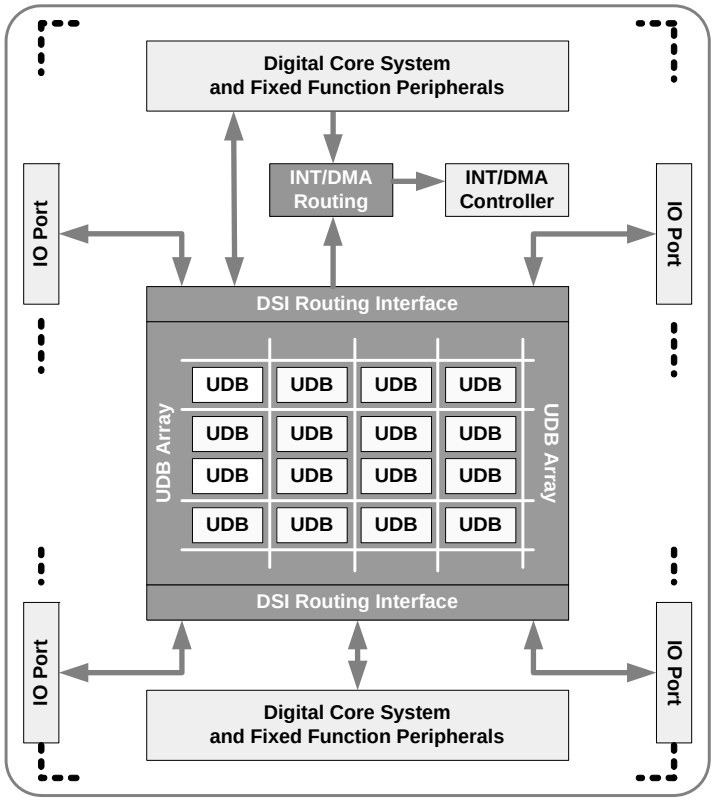
24.1 Features

- Offers a homogeneous array of UDBs which provide flexible function mapping
- Provides array level interconnect routing between the components of the UDB hardware
- Provides device level interconnect routing between UDBs, device peripherals, and I/O pins

24.2 Block Diagram

Figure 24-1 illustrates the programmable digital architecture for PSoC 5.

Figure 24-1. Programmable Digital Architecture



The main components of this system are:

- UDB Array:- UDB blocks are arrayed within a matrix of programmable interconnect. UDB pairs consisting of 2 UDBs are the basic building blocks of the UDB array. UDB pairs are tiled to create an array. UDB pairs can connect with neighboring UDB pairs in seamless fashion
- DSI- Routing interface tiled at top and bottom of UDB array core. Provides general purpose programmable routing between device peripherals, including UDBs, I/Os and fixed function blocks.
- System Interface (not shown)- Built in 8/16-bit bus interface with parallel access to all registers to support fast configuration. Also provides clock distribution and clock gating functionality.

The following section explain in detail the DSI routing and System Interface.

24.3 How It Works

The purpose of the DSI is to provide general purpose programmable connectivity across the device. Peripherals and system blocks that require connectivity are routed to this interface at the UDB array, which allows connections into the core of the array or directly between device peripherals.

Signals in this category include:

- Interrupt requests from all digital peripherals in the system
- DMA requests from all digital peripherals in the system
- Digital peripheral data signals that need flexible routing to I/Os
- Digital peripheral data signals that need connections to UDBs
- Connections to the interrupt and DMA controllers
- Connection to I/O pins
- Connection to analog system digital signals

Figure 24-2 and Figure 24-4 show some examples of the device peripherals that are connected to this interface, including UDBs, I/Os, analog peripherals, interrupts, DMA, and fixed function peripherals.

Figure 24-2. DSI Example Connections to the Interrupt and DMA Controller

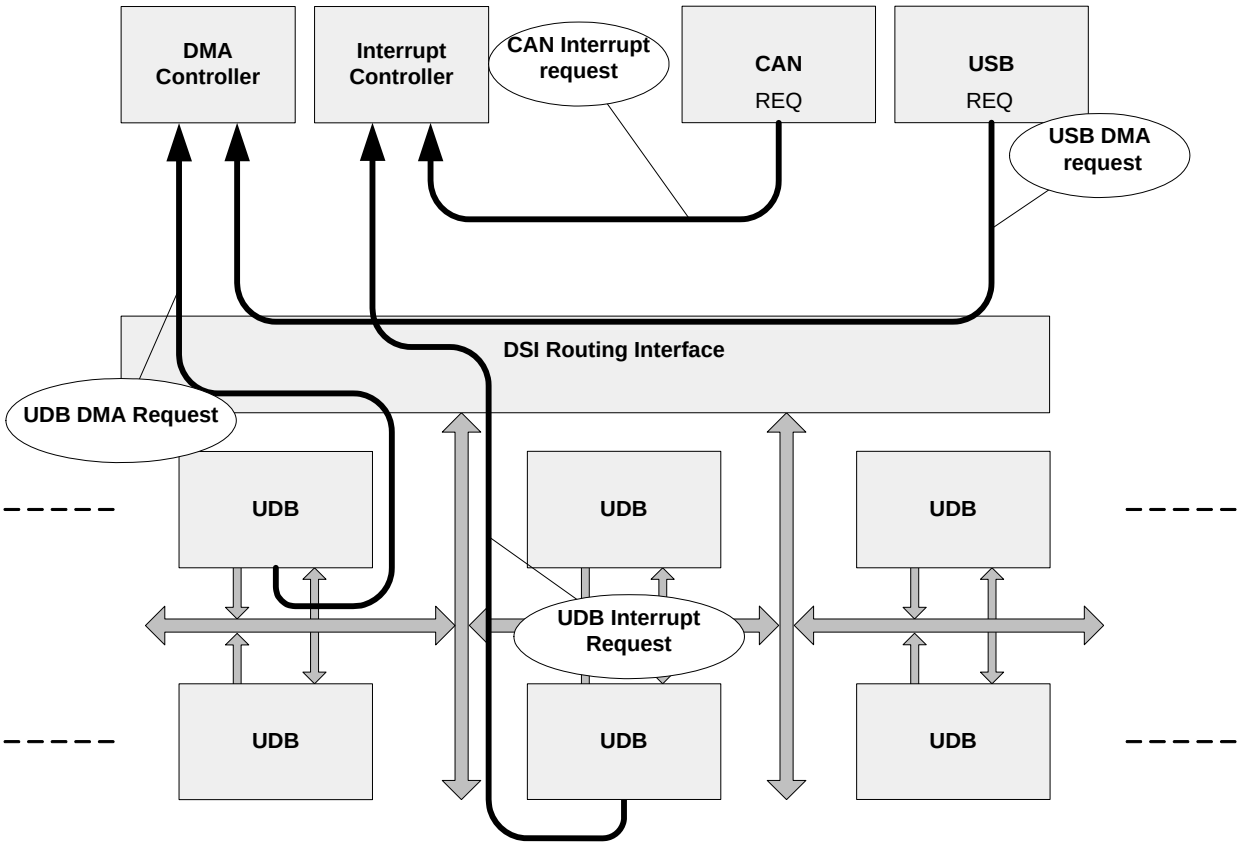
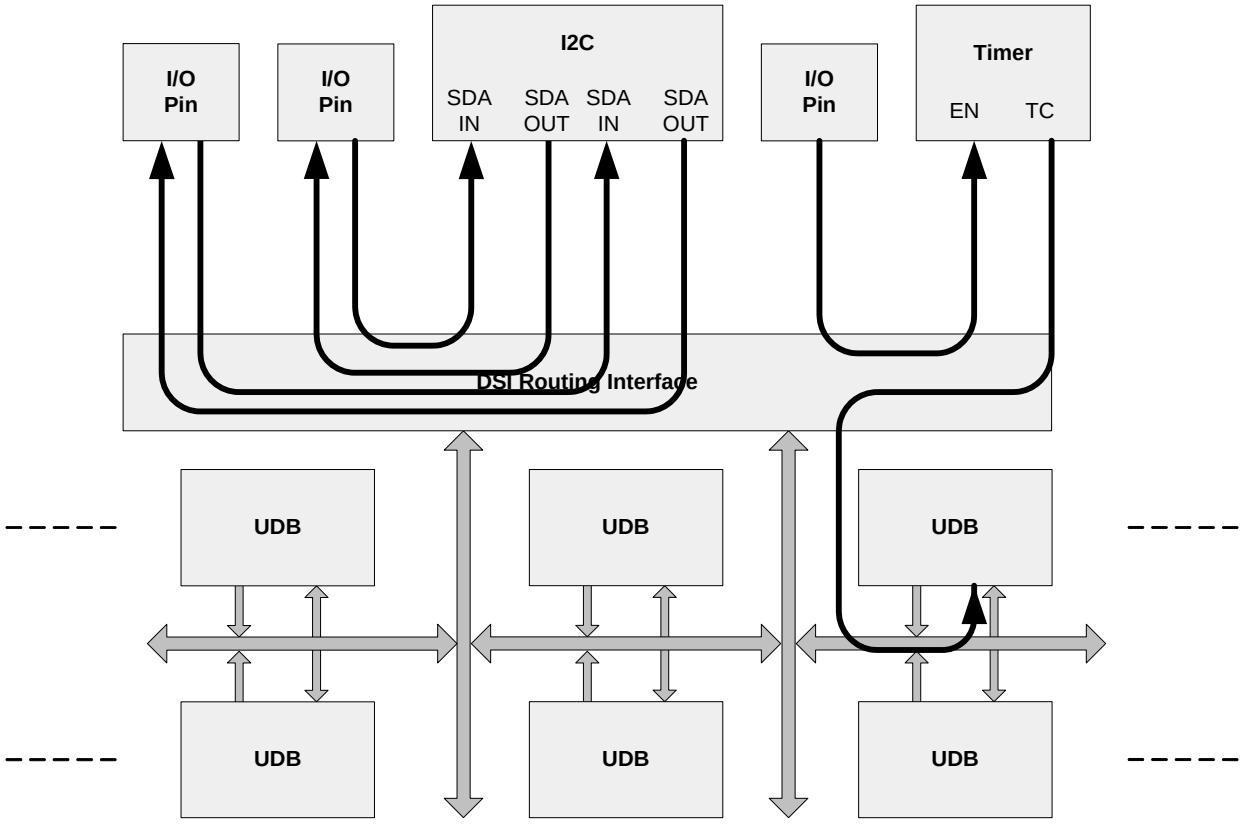


Figure 24-3. DSI Example Connections between Peripherals, I/O Pins, and UDBs



24.4 UDB Array System Interface

The system interface consists of infrastructure blocks that distribute and interface the device system bus to the UDB array bus and to the UDB blocks, the DSI channel routing, and the UDB pair channel routing. Depending on the configuration of the array, there is one or more AHB interfaces that connect to PHUB spokes providing an interface to the UDB array system bus. Both 8-bit and 16-bit bus access is supported. The system interface also provides support for clock distribution and gating for the digital global clocks and bus clock. A gated clock tree distribution is implemented to allow only those clocks that are in use to be activated.

Following are the system interface components:

- **AHB Interface** – Connects to a standard PHUB spoke and provides support for up to 1 bank of UDBs (16). Controls array wait states and translates AHB signaling into array register and routing configuration access control.
- **DSI Channel IF** – Interfaces the UDB array bus to the DSI routing channel for writing and reading configuration.
- **UDB Local IF** – Interfaces the UDB array bus to the UDB blocks for registers and RAM access, and provides local clock gating.
- **UDB Pair Channel** – Interfaces the UDB array bus to the pair routing channel for writing and reading configuration.
- **Bank IF** – Contains the master clock gating and bank wide configuration interface signals.
- **8-Bit WAIT_CFG Register** – Sets the read and write wait states for working and configuration registers.
- **4-Bit BANK_CTL Register** – Contains global bank control bits.
  - One bit to globally enable all DSI inputs. On POR, all DSI inputs are gated off until the DSI channel is configured. This bit globally enables DSI inputs to drive the routing.
  - One to disable all UDB status register clear-on-read function for debug support.
  - One to put the embedded DP RAM into test mode for DFT support.
  - One to put the bank into global write mode, also for DFT support.

There are eight digital global clocks, plus the application bus clock, routed to each bank of UDBs. The UDB local interface blocks contain clock gating control registers, which must be set by configuration firmware to enable clock distribution. There are four registers in each block:

- **8-Bit MDCLK_EN** (Master Digital Clock Enable) – This register individually enables the digital global clocks at the input to the UDB array.
- **1-Bit MBCLK_EN** (Master Bus Clock Enable) – This register individually enables the application bus clock at the input to the UDB array.
- **8-Bit DCLK_ENx** (Quadrant Digital Clock Enable) – This register individually enables the digital global clocks to the associated quadrant (4 UDBs) of the UDB array.
- **1-Bit BCLK_ENx** (Quadrant Bus Clock Enable) – This register individually enables the bus clock to the associated quadrant (4 UDBs) of the UDB array. It also contains bits to put the associated routing channel RAM into global write mode.

24.4.1 UDB Array POR Initialization

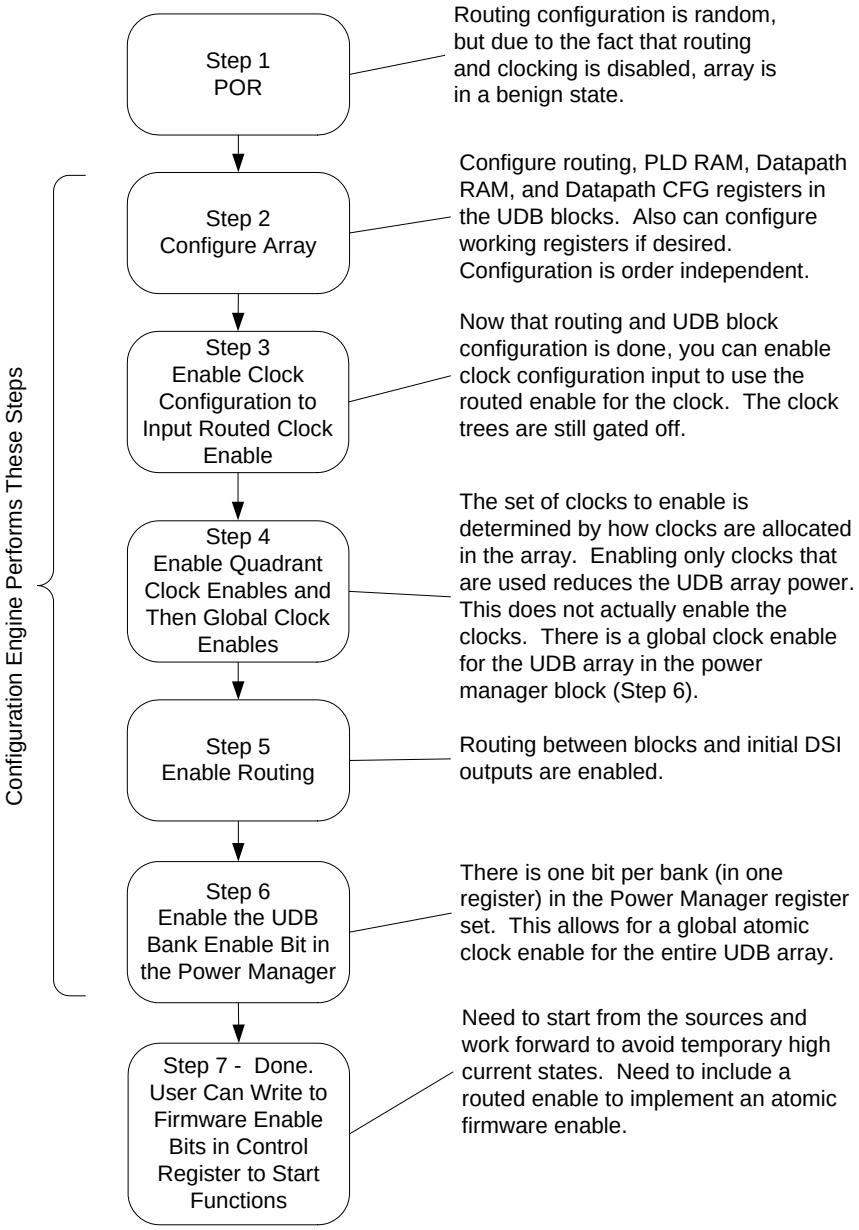
The key aspects of POR initialization are summarized as follows.

- All UDB clocks are gated off. There are three levels of clock gating configuration: one at the UDB level for each individual block clock control and a set of registers at the array level that controls master and quadrant clock gating.
- The state of all drivers into the routing matrix is gated to ‘0’ with a global routing enable control. This includes UDB block outputs, DSI inputs, and segmentation buffers. Because the routing is initialized to a random state, the state of routing nets will be either ‘0’ or ‘Z’.
- The inputs of all routing output buffers, including segmentation buffers, are gated to ‘0’ with a global routing enable control. This prevents floating routes from causing high power states. This also drives the buffer outputs to ‘0’ and that is the state for all DSI outputs.
- Configuration can occur in an order-independent way. When configuration is complete, each bank of UDBs has a global routing enable which is asserted to activate the connections (forced gating is disabled).
- After routing is enabled, a global clock enable bit (bank enable) can be set (residing in the power manager) which then enables clocking in the array. The bank enable bit prevents any spurious operation until the array is completely configured.

24.4.2 UDB POR Configuration Sequence

The previous section documented the POR state for the UDB array. From this initial state, configuration will proceed in the order shown in Figure 24-4.

Figure 24-4. POR Configuration Sequence



24.4.2.1 Quadrant Route Disable

To support fast bring up of initial functionality, the Quadrant Bus Clock Enable register contains a bit called Route Disable to disable the routing for the associated UDB quadrant (2 UDB pairs). By default, this bit is cleared and is not disabling the routing. If this bit is set to '1' during initial configuration, the associated channel routing RAM does not need to be configured. The global route enable bit can be set and this routing will remain in a benign state. Routing configuration for this quadrant can occur at a future time when this bit can be cleared to '0' to enable the routing (assuming that the global route enable bit is set).

24.4.3 UDB Sleep and Power Control

The UDB array has support for low power operation in the form of a sleep control input and power switch control inputs. All static configurations are on the "keep-alive" domain which retains state during a sleep/power down period. However, all application level working registers, including the accumulators, the data registers, the FIFO, control and status registers, etc., lose their state and must be reinitialized on power up. Nonretention registers and FIFO state are reset after a sleep period to insure a good initial state.

24.4.4 UDB Register References and Address Mapping

UDB registers are classified as shown in the Figure 24-5. There are five address spaces: one for 8-bit working registers (registers that are accessed during normal operation), one for 16-bit working registers, and three for configuration.

Each bank of UDBs is on a separate spoke, so a total of 6 select lines are generated from the PHUB to support the UDB array. The working registers are on the main 64K page (Page 0). The configuration registers have their own page (Page 1). Details of these registers are located in the *PSoC® 5 Registers TRM (Technical Reference Manual)*.

Figure 24-5. UDB Register Mapping

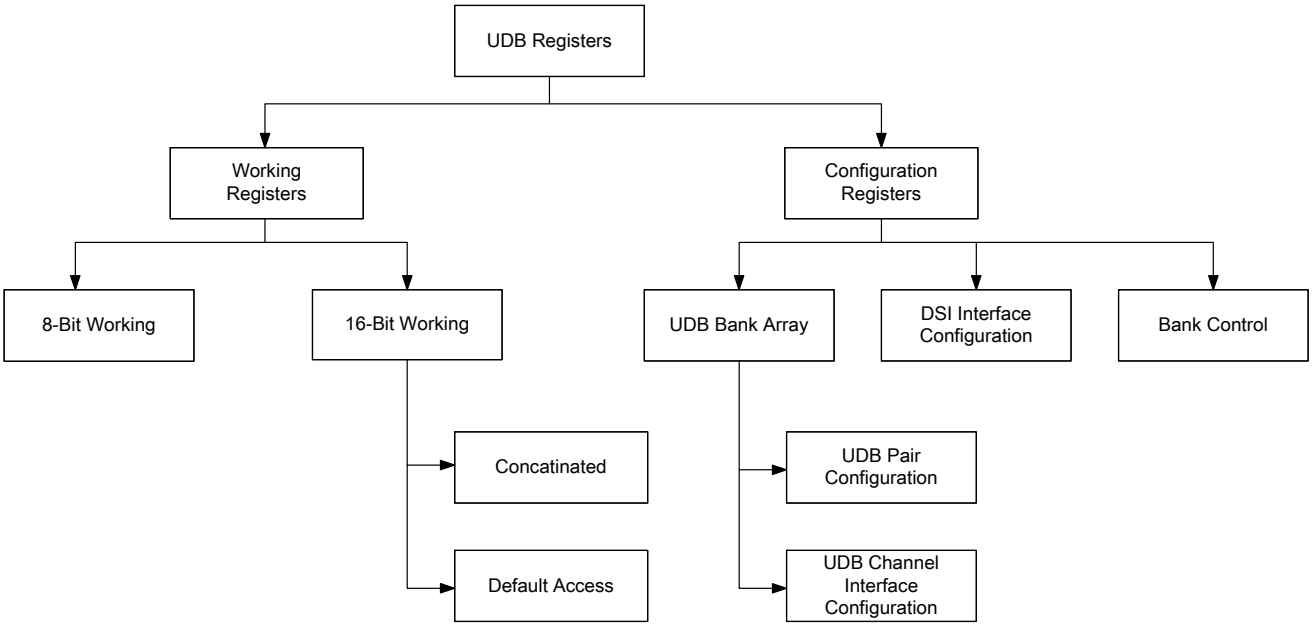
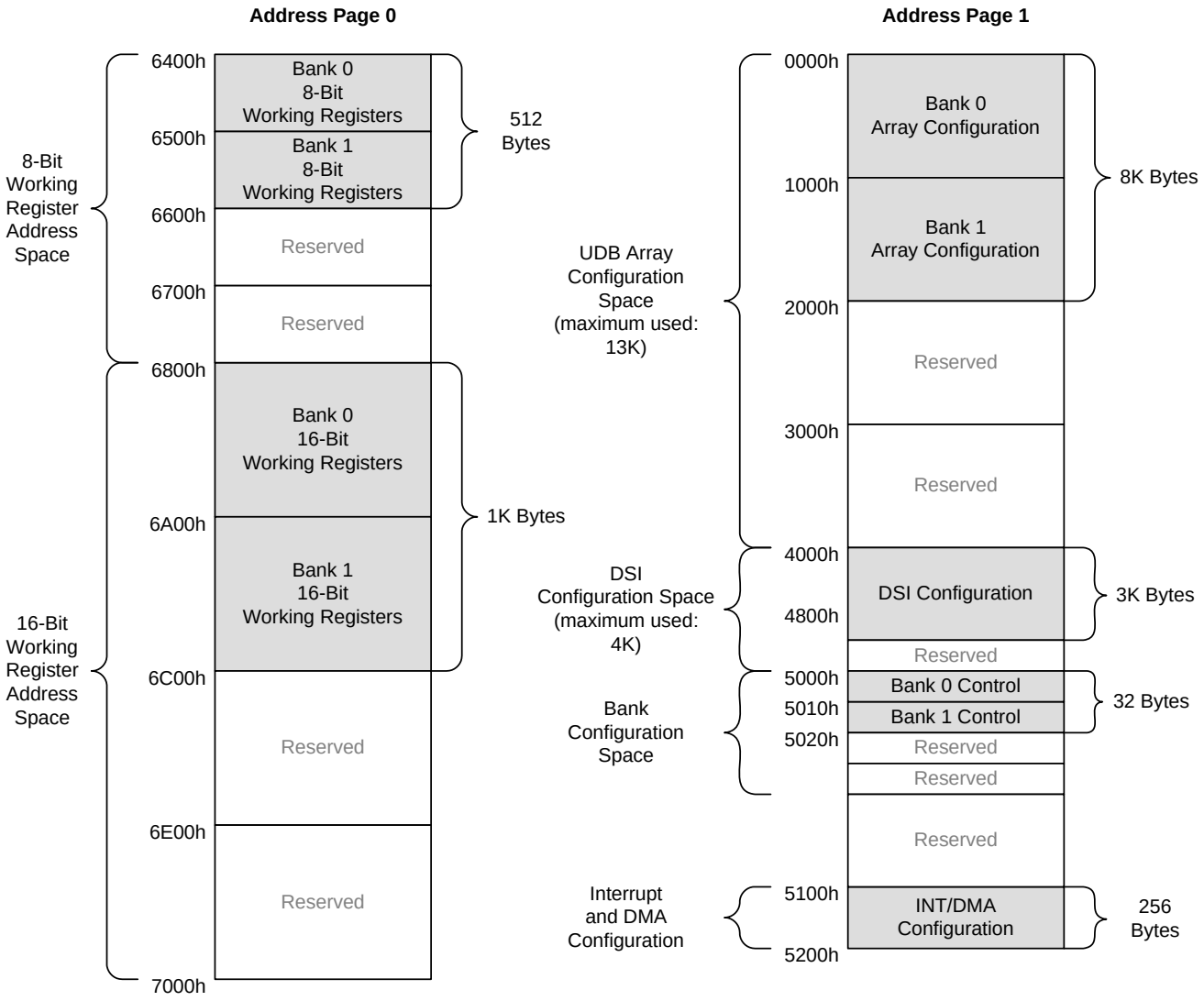


Figure 24-6 shows the register mapping for working and configuration registers of UDB and DSI.

Figure 24-6. UDB Array Base Addresses



25. USB

The PSoC® USB block acts as a USB device that communicates with a USB host. The USB block is available as a fixed function digital block in the PSoC device. It supports full speed communication (12 Mbps) and is designed to be compliant with the USB Specification Revision.2.0. USB devices can be designed for plug and play applications with the host and also support hot swapping. This chapter details the PSoC USB block and transfer modes. For details about the USB specification, see the USB Implementers Forum web site.

25.1 Features

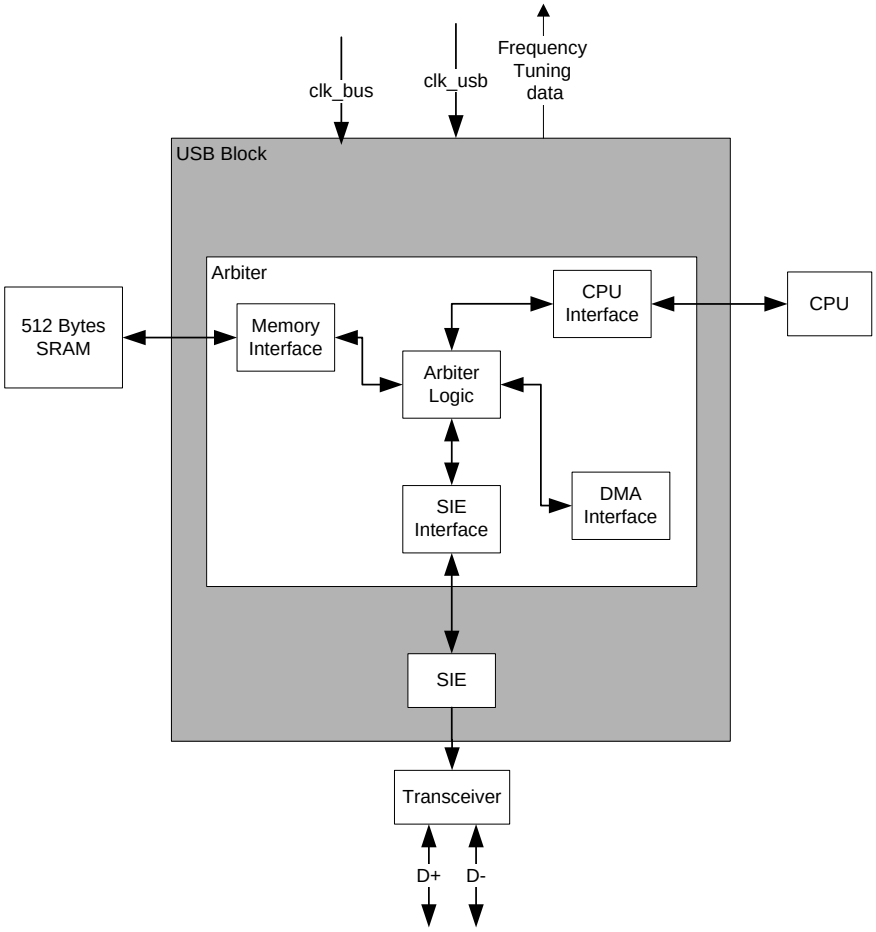
The PSoC USB has these features:

- Compatible with USB Specification 2.0
- Supports full speed peripherals device operation with a signaling bit rate of 12 Mbps
- Supports 8 unidirectional data endpoints and 1 control endpoint
- Supports four types of transfers – bulk, interrupt, isochronous, and control
- Supports Plug and Play
- Supports a Store and Forward logical transfer mode with a maximum packet size of 64 bytes
- Differential signal (D+ and D-) output
- Capable of supplying PS/2 and CMOS signals
- Supports two Vccd voltage ranges, with a nominal voltage of 3.3 V

25.2 Block Diagram

Figure 25-1 illustrates the architecture of the USB block. It consists of the Serial Interface Engine (SIE) and Arbiter.

Figure 25-1. USB Block Diagram



25.2.1 Serial Interface Engine (SIE)

The Serial Interface Engine (SIE) is responsible for handling the decoding and creating of data and control packets during transmit and receive. It decodes the USB bit streams into USB packets during receive and creates USB bit streams during transmit. The following are the features of the SIE block:

- Compatible with the USB 2.0 Specification
- Supports 1 device address
- Supports 8 data endpoints and 1 control endpoint
- Supports interrupt for each endpoint
- Operates at Full Speed with a 48 MHz Clock (maximum permitted tolerance is ±0.25%)
- Integrates an 8-byte buffer in the Control endpoint

The registers for this block are mainly used to configure the data endpoint operations and the Control Endpoint Data buffers. The register also controls the interrupt available for each endpoint.

The SIE generates an interrupt at the end of a transfer. The interrupt enabling and disabling for an endpoint can be done using the USB_SIE_INT_EN register. The status of the interrupt for an endpoint is obtained from the USB_SIE_INT_SR register.

The SIE registers CNT0 and CNT1 hold the count value for each endpoint which reports the number of data bytes in a USB transfer. In the case of an OUT endpoint, the firmware programs the maximum number of bytes that can be received for the endpoint. The SIE updates the register with the number of bytes received. In the case of an IN endpoint, it holds the number of bytes that will be transmitted.

The SIE Control register for each endpoint, USB_SIE_EPx_CR0, holds the mode value. The mode value determines the response of the USB block to the host. Refer to Table 25-1 for the different mode values. The table describes the mode values corresponding to each type token: the SETUP, IN and OUT tokens.

Transition error is also reported by the SIE. The bit “err_in_txn” in the USB_SIE_EPx_CR0 register indicates the occurrence of an error. When this bit is set, the hardware automatically retransmits the same data when it receives another IN token from the host.

Table 25-1. Mode Values in the MODE bits of the SIE_EPx_CR0 Register

Mode	Encoding	SETUP	IN	OUT	Comments
Disable	0000	Ignore	Ignore	Ignore	Ignore all USB traffic to this endpoint
NAK IN/OUT	0001	Accept	NAK	NAK	NAK IN and OUT token
Status OUT Only	0010	Accept	STALL	Check	When this mode is set, it accepts a SETUP token, STALLs in case of IN token and ACKs with a zero length packet in case of OUT token. Used for control endpoint
STALL IN/OUT	0011	Accept	STALL	STALL	When this mode is set, it accepts a SETUP token, STALLs in case of IN and OUT token. Used for control endpoint
Reserved	0100	Ignore	Ignore	Ignore	
ISO OUT	0101	Ignore	Ignore	Always	Isochronous OUT
Status IN only	0110	Accept	TX 0 byte	STATLL	When this mode is set, it accepts a SETUP token, STALLs in case of OUT token and ACKs with a zero length packet in case of IN token. Used for control endpoint
ISO IN	0111	Ignore	TX Count	Ignore	Isochronous IN
NAK OUT	1000	Ignore	Ignore	NAK	Send NAK handshake to OUT token
ACK OUT (STALL = 0)	1001	Ignore	Ignore	ACK	This mode is changed by the SIE to mode 1000 on issuance of ACK handshake to an OUT
ACK OUT (STALL = 1)	1001	Ignore	Ignore	STALL	STALL the OUT transfer
Reserved	1010	Ignore	Ignore	Ignore	
ACK OUT – STATUS IN	1011	Accept	TX0 byte	ACK	ACK the OUT token or send zero length data packet for IN token.
NAK IN	1100	Ignore	NAK	Ignore	Send NAK handshake for IN token
ACK IN (STALL = 0)	1101	Ignore	TX Count	Ignore	This mode is changed by the SIE to mode 1100 after receiving ACK handshake to an IN data
ACK IN (STALL = 1)	1101	Ignore	STALL	Ignore	STALL the IN transfer
Reserved	1110	Ignore	Ignore	Ignore	
ACK IN – Status OUT	1111	Accept	TX Count	Check	Respond to IN data or Status OUT

25.2.2 Arbiter

The Arbiter is the block which handles access of the SRAM memory by the endpoints. The SRAM memory can be accessed by the CPU or the SIE. The Arbiter handles the arbitration between the CPU and the SIE. The Arbiter consists of the following blocks:

- SIE Interface Module
- CPU Interface Module
- Memory Interface
- DMA Engine
- Arbiter Logic
- Synchronization Module

The Arbiter registers are used to handle the endpoint configurations, the Read address, and the Write address for the endpoints. It also configures the logical transfer type required for each endpoint. The types of logical transfers are discussed below. Also, each endpoint supports interrupt. The Arbiter has only one interrupt line for the Interrupt Controller. The Arbiter registers handle the enabling/disabling of the interrupts for the endpoints and hold the status of the interrupts. The Arbiter is also responsible for the memory management (i.e., sharing the available 512 bytes of SRAM among the data endpoints).

25.2.2.1 SIE Interface Module

This module handles all the transactions with the SIE block. The SIE reads data from the SRAM memory and transmits to the host. Similarly, it writes the data received from the host to the SRAM memory. These requests are registered in the SIE Interface module and are handled by this block.

25.2.2.2 CPU Interface Block

This module handles all the transactions with the CPU. The CPU makes requests for the reads and writes to the SRAM memory for each endpoint. These requests are registered in the CPU Interface block and are handled by the block.

25.2.2.3 Memory Interface

The memory interface is used to control the interface between the USB block and the SRAM memory unit. The maximum memory size supported is 512 bytes organized as 256 x a 16-bit memory unit. This is a dedicated memory for the USB. All the control and data lines, including the Data In lines, Data Out lines, Enable line, Address lines, and Direction Control line between the USB and the memory unit, are handled by the memory interface. The memory access can be requested by the SIE or by the CPU. The SIE Interface block and the CPU Interface block handle these requests.

25.2.2.4 DMA Interface

When Direct Memory Access (DMA) is configured, the DMA interface is responsible for all transactions back and forth between the DMA and USB. The block supports the DMA request line for each data endpoint. The behavior of the DMA depends on the type of logical transfer mode configured in the Configuration register. Note that DMA transfers from UDBs to the USB block must first go through SRAM to ensure that proper timing is kept. An additional transaction descriptor should be used to transfer from UDBs to SRAM, and then from SRAM to USB. Other applicable DMA transfers from sources besides UDBs are not constrained to this path.

25.2.2.5 Arbiter Logic

This is the main block of the Arbiter. It is responsible for arbitrations for all the transactions that happen in the Arbiter. It arbitrates the CPU, DMA, and SIE access to the memory unit and the registers. This block also handles the memory management. The read and write address manipulations are done by the firmware. This block takes care of the buffer size allocation, depending on the programmed buffer size (using the USB_BUF_SIZE). It also does the handling of common memory area.

This block also handles the interrupt requests for each endpoint. Each endpoint can have interrupts due to:

- DMA Grants
- IN Buffer Full
- Buffer Overflow
- Buffer Underflow

These arbiter interrupt requests are routed to only one interrupt line which acts as a signal to the interrupt controller.

25.2.2.6 Synchronization Block

The USB block uses two main clocks: the System Clock and the USB Clock. The System Clock is used by the Arbiter. The USB Clock is used by the SIE and the OsClock module. Because these two are different clocks, synchronization is required between the blocks. The handling of the synchronization is done by this block.

25.3 How it Works

The USB Block operates at a certain frequency and voltage range. For proper operation of the USB block, the user must ensure that the operating ranges are within tolerances. The following sections discuss the operating ranges required for the PSoC USB.

25.3.1 Operating Frequency

The USB block needs three different clocks to work: the System Clock which controls the Arbiter, memory and the register block, the USB clock which controls the SIE and the OsClock, and the ILO which is used to detect a USB RESET.

- Minimum system clock – 33 MHz
- USB Clock for Full Speed operation – 48 MHz (+0.25% tolerance)
- Internal Local Oscillator (ILO) – 100 kHz

The USB needs a 48 MHz clock to function. The clock to the USB is called the clk_usb. The clk_usb can be derived from either IMOCLK, doubler clock (IMOCLK * 2), PLL, or the DSI clock. For further details about the clock for this block, refer to the Clocking System chapter on page 91. The OsClock block of the USB trims the USB clock to lock to the frequency of the USB packets. The USB clock is clocked to the USB token according to the USB 2.0 Specification. When the frequency is locked with other USB bit streams, the block will locate a particular edge in the USB packet. The number of clock periods between these edges is measured to lock the internal oscillator frequency with the frequency of the USB packet. The frequency tuning value is sent to the Clocking system by the USB Block to lock the frequency.

The locking of the frequency is done by the hardware and needs no user intervention. The Synchronization Block of the Arbiter handles the synchronization of the USB Clock and System Clock.

The USB uses the 100 kHz Fast Time Wheel (FTW) in the detection of a USB reset event. The counts from this clock are stored in the USB_BUS_RST_CNT register. A reset condition will be triggered upon three counts of the 100 kHz clock when a reset condition on the bus is detected.

25.3.2 Operating Voltage

The USB block can operate in two voltage ranges:

- Standard voltage range – 4.35 V to 5.25 V
- Lower voltage range – 3.15 V to 3.45 V

The USB needs a nominal voltage of 3.3 V for its operation. The block uses the regulated digital voltage Vccd. It supports an internal regulator which is used for voltage regulation. While in the Standard Voltage Range, the voltage is regulated to 3.3 V by the internal regulator. While in the Lower Voltage Range, the internal regulator should be bypassed. The “reg_enable” bit in the USB_USB_CR1 register is used to control the regulator usage.

In all other voltage ranges (that is, 1.7 V to 3.15 V, 3.45 V to 4.35 V, and 5.25 V to 5.5 V, the “suspend,” “pull up,” and “high impedance drive” modes will work properly because the current specification is met. The Drive modes can be selected using the registers USB_USBIO_CR1 and USB_USBIO_CR2.

25.3.3 Transceiver

The USB block includes the transmitter and the receiver. The signal between the USB device and the host is a differential signal. The receiver receives the differential signal and converts it to a single ended signal. The single ended input is given to the USB block at a nominal voltage range of 1.55 V to 1.95 V. The transmitter converts the single ended signal to the differential signal and transmits it to the host. The differential signal is given to the upstream devices at a nominal voltage range of 0 V to 3.3 V.

The transceiver also supports the PS/2 signals. It can receive and transmit PS/2 signals at a nominal voltage of 0 V to 5 V. The transceiver has the pull up resistors to support the PS/2 signals.

Apart from the PS/2 signals, the transceiver also supports the CMOS signal levels. The PS/2 and the CMOS modes can be selected using the registers USB_USBIO_CR1 and USB_USBIO_CR2.

The Transmitter can be manually forced to transmit signals. The register USB_USBIO_CR0 is used to manually transmit the signals. Examples are as follows:

- When the manual transmission is enabled, the register can be configured to transmit Single Ended Zero signal (that is, D+ and D- are low).
- Configurable to transmit the USB signals. The USB signals can be two types:
  - D+ low and D- high = J
  - D+ high and D- low = K
- The register also has a bit which is used to read the received signal levels. The bit can show if D+ < D- or D+ > D-.

25.3.4 Endpoints

The SIE and Arbiter support 8 unidirectional (supports IN or OUT) data endpoints (EP1 to EP8) and one control endpoint (EP0). The data endpoints share the SRAM memory area of 512 bytes. The endpoints are configured for direction and other configuration using the SIE and arbiter registers. The endpoint “read address” and “write address” registers are accessed through the Arbiter. Each endpoint supports a set of interrupts. The interrupts can be enabled or disabled for an individual endpoint. The interrupts for each endpoint can also be collectively enabled or disabled.

The endpoints can be individually made active. The firmware decides the memory allocation, so it is not required to specify the active endpoints. The USB_EP_TYPE register is used to control the transfer direction (IN, OUT) for the endpoints. Endpoint parameters such as activation, transfer type, and direction cannot be dynamically changed during runtime. The control endpoint has a separate 8 bytes for its data.

25.3.5 Transfer Types

The PSoC USB supports Full Speed transfers and is compatible with the USB 2.0 Specification. It supports four types of transfers:

- Interrupt Transfer
- Bulk Transfer
- Isonchronous Transfer
- Control Transfer

For further details about these transfers, refer to the USB Specification 2.0.

25.3.6 Interrupts

The interrupts are generated by SIE and Arbiter. The following interrupt lines are available for the interrupt controller:

- Nine SIE interrupt lines (one for each endpoint and control endpoint)
- Arbiter interrupt line
- SIE interrupt line for SOF
- SIE data endpoints interrupt line
- Reset interrupt line

Nine SIE Interrupts

- Generated after the completion of packet transmission.
  - Automatic for acknowledged transfer
  - Can be enabled for non-acknowledged transfer
- The register USB_SIE_EP_INT_EN is used to enable the SIE interrupt for each endpoint. Each bit in the register corresponds to each endpoint.
- The status of the SIE interrupt can be read using the USB_SIE_EP_INT_SR register. These bits are sticky bits and need firmware to clear the status.
- Separate interrupt line for each data endpoint and control endpoint.
- The register SIE_EP_INT_EN and SIE_EP_INT_SR control/show the status of both SIE and Data Endpoint interrupts.

Arbiter Interrupt Line

The arbiter generates interrupts for the endpoints during these events:

- Buffer overflow
- Buffer underflow
- DMA grant
- IN endpoint local buffer full

This information applies to the arbiter interrupts.

- These interrupts can be generated by every endpoint. The register USB_ARB_EPx_INT_EN (where x = 1 to 8 for each endpoint) is used to enable or disable each interrupt for the endpoint.
- The Status of each interrupt for every endpoint can be read using the USB_ARB_EPx_INT_SR (where x = 1 to 8 for each endpoint) register.
- The interrupt for an endpoint can be collectively enabled or disabled using the USB_ARB_INT_EN register_ Each bit in this register corresponds to each endpoint.
- The status of the Arbiter interrupt for an endpoint can be read using the USB_ARB_INT_SR register.

There is only one arbiter line common for all the endpoints.

SIE Interrupt for SOF

- Generated whenever the SOF is received.

SIE Data Interrupt

- Interrupt generated for the data valid or error in transaction.
- One interrupt line common for all endpoints.
- The sticky bit “data_valid,” in the USB_SIE_EPx_CNT0 register, indicates the data valid state.
- The sticky bit “err_in_txn” in the USB_SIE_EPx_CR0 register indicates the error in transaction state.

Reset Interrupt

- Generated after detecting a reset event.

25.4 Logical Transfer Modes

The USB block in PSoC devices supports a single type of logical transfer and three types of data transfers (Interrupt, Bulk, and Isochronous) mentioned in the USB 2.0 Specification. The Control transfer is mandatory in any USB device. The logical transfers can be configured using the register setting for each endpoint.

The logical transfer mode is a combination of memory management and DMA configurations. The Logical Transfer modes are related to the data transfer within the USB block (to and from the SRAM memory unit for each endpoint). It does not represent the transfer methods between the device and the host (transfer types specified in the USB 2.0 Specification).

The USB block supports a Store and Forward transfer mode; see Table 25-2 on page 205 for details.

Table 25-2. USB Transfer Modes

Feature	Store and Forward Mode
SRAM Memory Management	Manual
SRAM Memory Sharing	512 bytes of SRAM shared between endpoints. Sharing is done by firmware.
IN Command	Entire packet present in SRAM memory before the IN command is received.
OUT Command	Entire packet is written to SRAM memory on OUT command. After entire data is available, it is copied from SRAM memory to the USB device.
Transfer of Data	Data is transferred when all bytes are written to the memory.
Types Based on DMA	No DMA mode (CPU based) Manual DMA mode
Supported Transfer Types	Bulk, Interrupt, and Isochronous transfers

Every endpoint has a set of registers that need to be handled during the modes of operation, as detailed in Table 25-3.

Table 25-3. Endpoint Registers

Register	Comment	Content	Usage
ARB_RWx_WA	Endpoint Write Address register	Address of the SRAM	This register indicates the SRAM location to which the data in the Data register is to be written.
ARB_RWx_RA	Endpoint Read Address register	Address of the SRAM	This register indicates the SRAM location from which the data must be read and stored to the Data register.
ARB_RWx_DR	Endpoint Data Register	8-Bit Data	Data register is read/ written to perform any transaction. IN command: Data written to the Data register is copied to the SRAM location specified by the WA register. After write, the WA value is automatically incremented to point to the next memory location. OUT command: Data available in the SRAM location pointed by the RA register is read and stored to the DR. When the DR is read, the value of RA is automatically incremented to point to the next SRAM memory location that must be read.
SIE_EPx_CNT0 and SIE_EPx_CNT1	Endpoint Byte Count Register	Number of Bytes	Holds the number of bytes that can be transferred. IN command: Holds the number of bytes to be transferred to host. OUT command: Holds the maximum number of bytes that can be received. The firmware programs the maximum number of bytes that can be received for that endpoint. The SIE updates the register with the number of bytes received for the endpoint.
“Mode” bits in SIE_EPx_CR0	Mode Values	Response to the Host	Controls how the USB device responds to the USB traffic and the USB host. Some examples of mode include ACK, NAK, STALL, etc. Refer to Table 25-1 on page 201 for additional details.

The endpoint read and endpoint write address registers are updated by the firmware. So the memory allocation can be done as required by the user and the memory allocation decides which endpoints are active. (i.e., the user can decide to share the 512 bytes for all the 8 endpoints or a lesser number of endpoints).

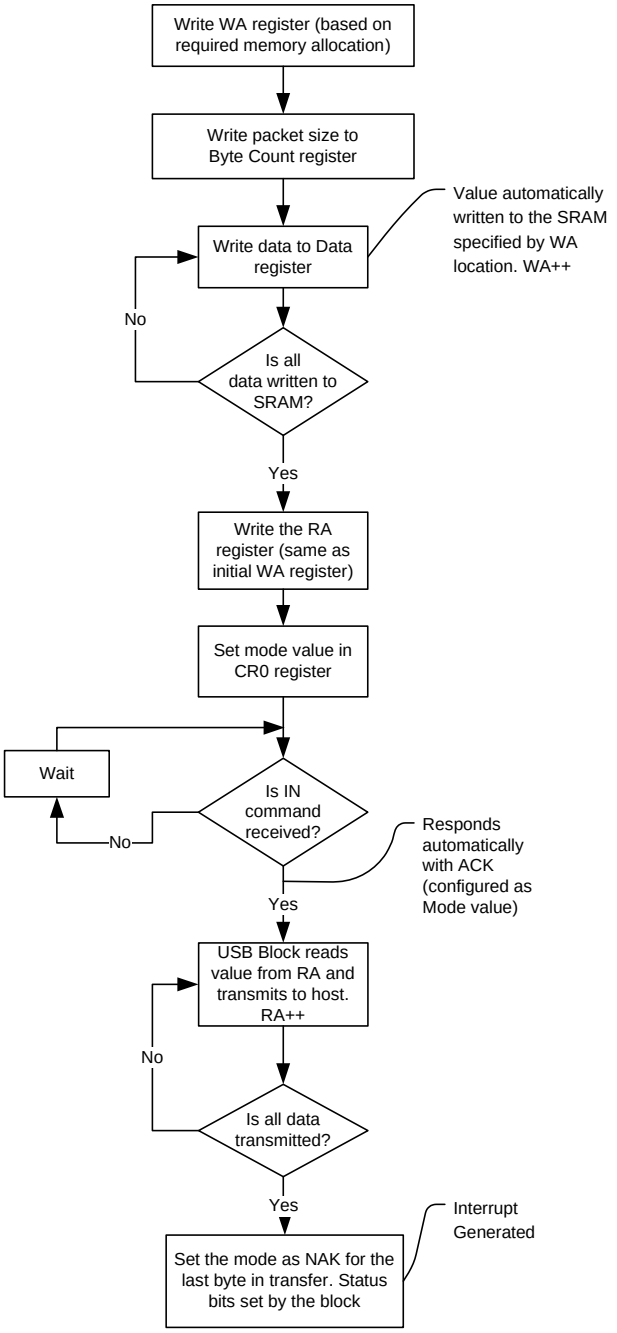
In the following text, the algorithm for the IN and OUT transaction for each mode is discussed. An IN transaction is when the data is read by the USB host (for example, PC). An OUT transaction is when the data is written by the USB host to the USB device (in this case, PSoC 5). The choice of using the DMA and memory management can be configured using the USB_ARB_CFG register and the mode is common to all endpoints.

25.4.1 No DMA Access

This is the Store and Forward mode with no DMA access.

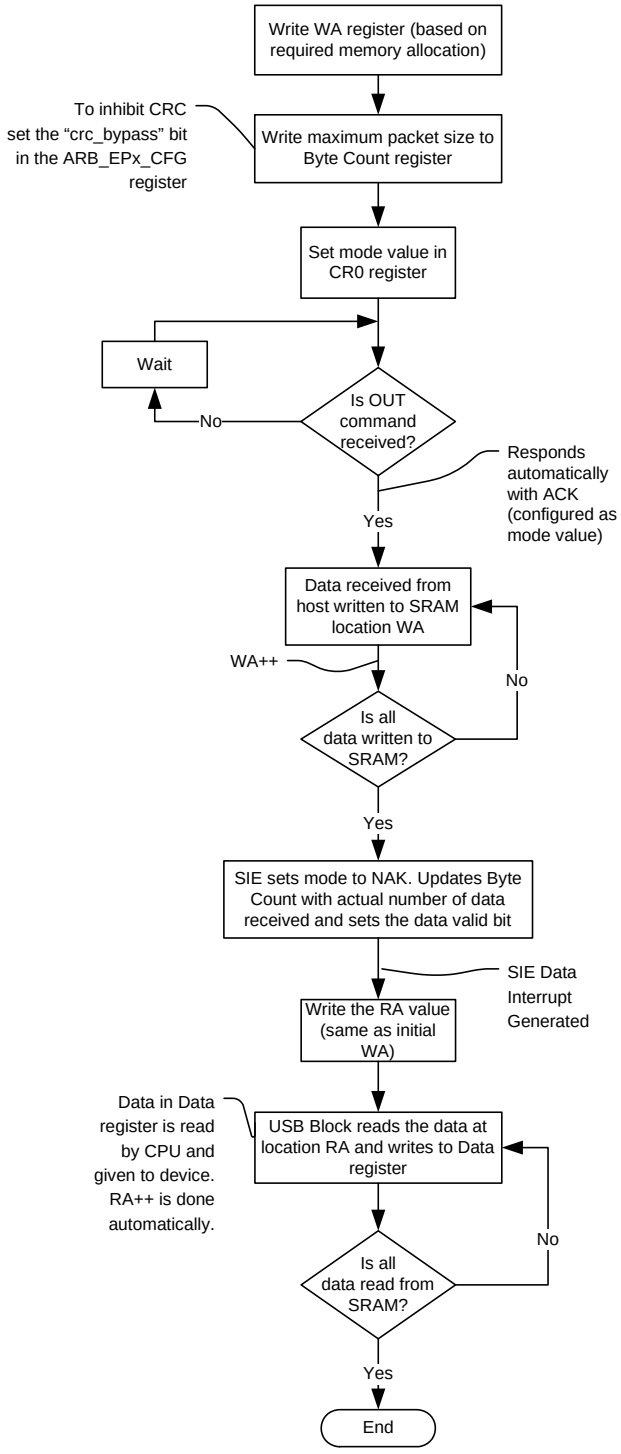
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 25-2.

Figure 25-2. No DMA Access IN Transaction



OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in Figure 25-3.

Figure 25-3. No DMA Access OUT Transaction



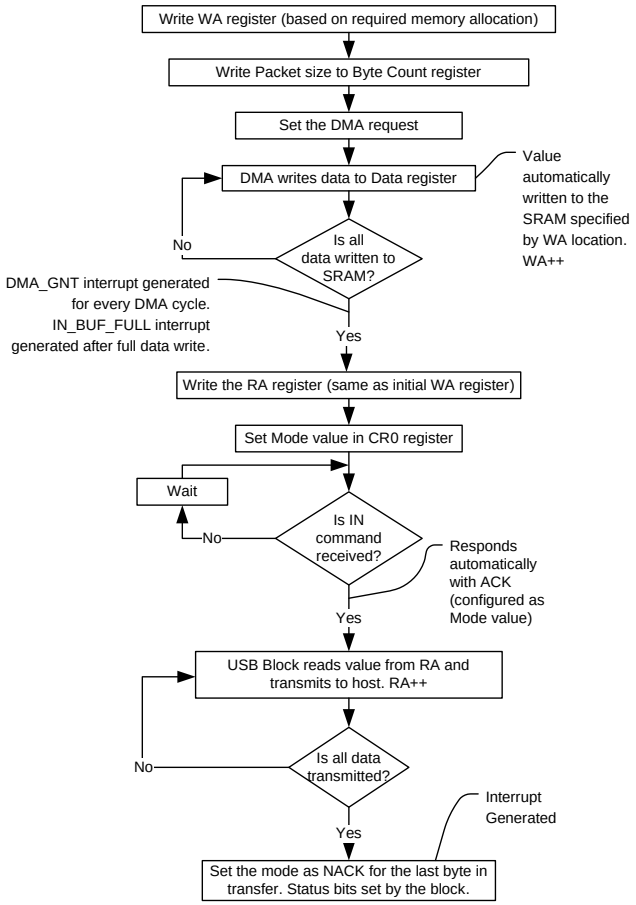
25.4.2 Manual DMA Access

This is the Store and Forward mode with Manual DMA Access. This mode requires the configuration of the DMA controller. See DMA Interface on page 202 for details and constraints about DMA transfers to the USB block.

This mode is similar to the No DMA Access except that the write/read of packets is performed by DMA. A DMA request for an endpoint is generated by setting the DMA_CFG bit in the ARB_EPx_CFG register. When the DMA service is granted and is done (DMA_GNT), an arbiter interrupt can be programmed to occur. The transfer is done using a single DMA cycle or multiple DMA cycles. After completion of every DMA cycle the arbiter interrupt (DMA_GNT) is generated. Similarly, when all the bytes of data (programmed in the byte count) are written to the memory, the arbiter interrupt occurs and the IN_BUF_FULL bit is set.

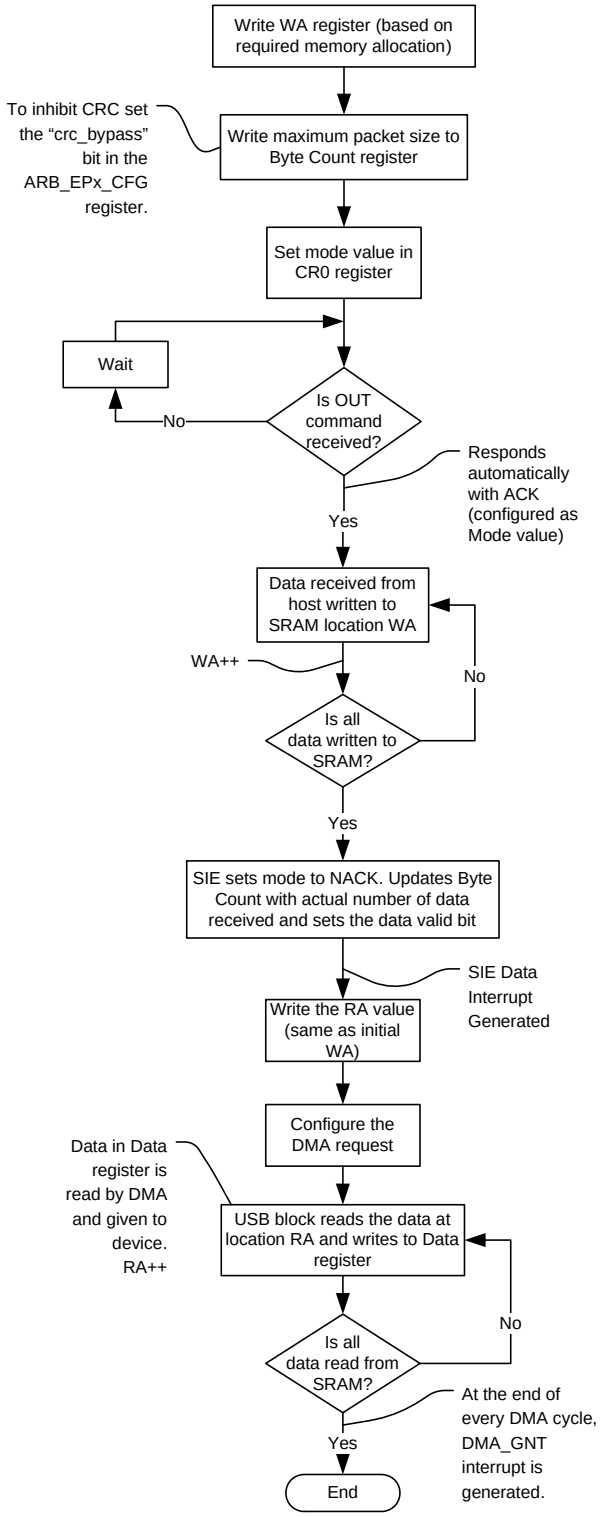
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 25-4.

Figure 25-4. Manual DMA IN Transaction



OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in Figure 25-5.

Figure 25-5. Manual DMA OUT Transaction



25.4.3 Control Endpoint Logical Transfer

The control endpoint has a special logical transfer mode. It does not share the 512 bytes of memory. Instead it has dedicated 8 byte register buffer. The IN and OUT transaction for the control endpoint is detailed in the following figures.

Figure 25-6. IN Transaction

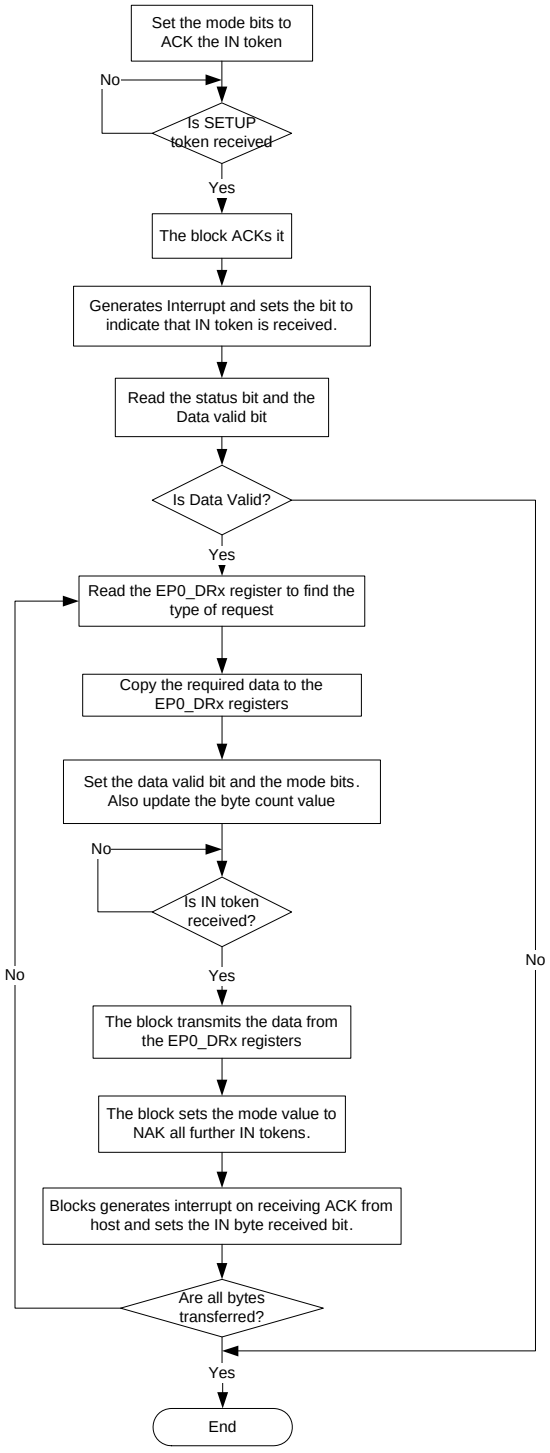
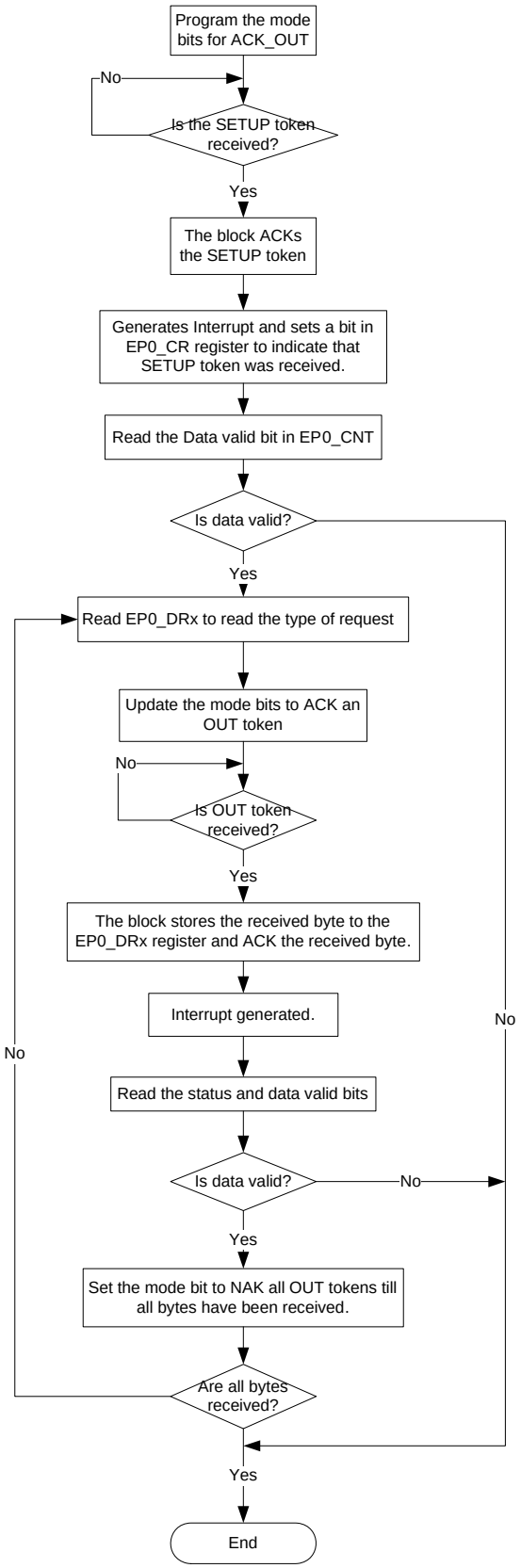


Figure 25-7. OUT Transaction



25.5 PS/2 and CMOS I/O Modes

The USB transceiver is designed in such a way that, apart from the USB signals, it can also transmit other signal levels. The pull up resistors are available at the transmitter end, which enables additional signal levels. The registers USB_USBIO_CR1 and USB_USBIO_CR2 must be configured to get different signal levels.

The D+ and D– USB pins only support the following drive modes:

- Open Drain, Drives Low
- Resistive Pull Up
- Strong Drive

For more information, refer to Figure 21-3 on page 128.

The “test_res” bit in the USBIO_CR2 register puts the transmitter in pull up mode where the pull up resistors are connected.

The I/O mode bit in the USBIO_CR1 register puts the USB in either USB mode or Drive mode. When put in Drive mode, the USB signals are disabled and the bits DMI and DPI are used to drive D- and D+, respectively. There are two different drive modes. In CMOS Drive mode, D+ follows the DPI and D- follows the DMI. In the case of Open Drain mode, the pull up resistors play a role. In this state, when the DPI and DMI bits are set to high, D+ and D- are high impedance.

The pull up resistors can be connected between Vdd and D+ and D-, independent of the Drive modes. The bit “p2puen” is used for this.

An internal pull up of 1.5 kΩ is also supported and can be enabled using the register USBIO_CR1. The USBIO_CR1 register is also used to poll the state of the D+ and D- pins.

25.6 Register List

Table 25-4. USB Register List

Register Name	Comments	Features
General Registers		
USB_CR0	USB Control register 0	To enable the USB and store the USB Device address
USB_CR1	USB Control register 1	To monitor the bus activity and control the regulator operation
USBIO_CR0	USB I/O Control register 0	To control the operation on D+ and D- signals
USBIO_CR1	USB I/O Control register 1	To configure the pull up registers
USBIO_CR2	USB I/O Control register 2	To control in test modes
USB_BUF_SIZE	Dedicated endpoint buffer size register	Stores the dedicated buffer size for each endpoint
USB_EP_ACTIVE	Endpoint active register	Stores the status of active endpoints
USB_EP_TYPE	Endpoint Type register	Stores the type of endpoint either IN/OUT
USB_EP0_DRx x= 0 -7	Control endpoint Data register	The endpoint 0 is the control endpoint
USB_EP0_CR	Endpoint 0 Control register	
USB_EP0_CNT	Endpoint 0 Count register	
SIE Registers		
USB_SIE_EP_INT_EN	Interrupt enable register	To enable the interrupts for each endpoint
USB_SIE_EP_INT_SR	Interrupt status register	To find the status of interrupt for each endpoint
USB_SIE_EPx_CNT0 x= 1- 8	Non control endpoint Count register	Handles the Data toggle state and MSB of the 11 bit counter
USB_SIE_EPx_CNT0 x= 1- 8	Non control endpoint Count register	LSB of the 11 bit counter
USB_SIE_EPx_CR0 x = 1 - 8	Non control endpoint Control register	Controls the mode for the endpoint and stores the state of error, ACK and NACK for the endpoint.
OsClock Registers		
OSCLK_DR0	OsClock Lock register 0	The LSB of the Oscillator locking circuit output
OSCLK_DR1	OsClock Lock register 1	The MSB of the Oscillator locking circuit output
Arbiter Registers		

Table 25-4. USB Register List *(continued)*

Register Name	Comments	Features
USB_ARB_EPx_CFG x = 1 – 8	Endpoint configuration register	Stores the configuration for the transfer modes, reset of pointers and CRC
USB_ARB_Epx_INT_EN x = 1 – 8	Endpoint Interrupt enable register	To enable the required interrupts
USB_ARB_Epx_SR x = 1- 8	Endpoint status register	To indicate status such as overflow, underflow, DMA grant, and Local buffer full
USB_ARB_RWx_WA x = 1 – 8	Endpoint Write address register	Stores the LSB 8 bits of the Write address pointer
USB_ARB_RWx_WA_MSB x = 1 – 8	Endpoint Write address register	Stores the MSB 1 bit of the Write address pointer
USB_ARB_RWx_RA x = 1 – 8	Endpoint Read address register	Stores the LSB 8 bits of the Read address pointer
USB_ARB_RWx_RA_MSB x = 1 – 8	Endpoint Read address register	Stores the MSB 1 bit of the Read address pointer
USB_ARB_CFG	Arbiter Configuration register	
USB_ARB_INT_EN	Arbiter Interrupt Enable register	To enable the interrupt for each endpoint
USB_ARB_INT_SR	Arbiter Interrupt Status register	To store the interrupt status for each endpoint
USB_CWA	Common Area Write Address register	The LSB 8 bits of the Write address pointer
USB_CWA_MSB	Common Area Write Address register	The MSB 1 bit of the Write address pointer
USB_DMA_THRES	DMA Threshold Count register	The LSB 8 bits of the DMA threshold count register
USB_DMA_THRES_MSB	DMA Threshold Count register	The MSB 1 bit of the DMA threshold count register
USB_SOF0	Start of Frame register 0	LSB 8 bits of the Start of Frame counter
USB_SOF1	Start of Frame register 1	MSB 3 bits of the Start of Frame counter
USB_BUS_RST_CNT	Bus reset count register	The reset counter for the USB

26. Timer, Counter, and PWM

Timer blocks in PSoC® devices are 8/16 bits and configurable to act as Timer, Counter, or Pulse Width Modulator (PWM) blocks that play important roles in embedded systems. PSoC devices give a maximum of four instances of the block. If additional blocks are required, they can be configured in the UDBs using PSoC Creator™. Timer blocks have various clock sources and are connected to the General Purpose Input/Output (GPIO) though the Digital System Interconnect (DSI).

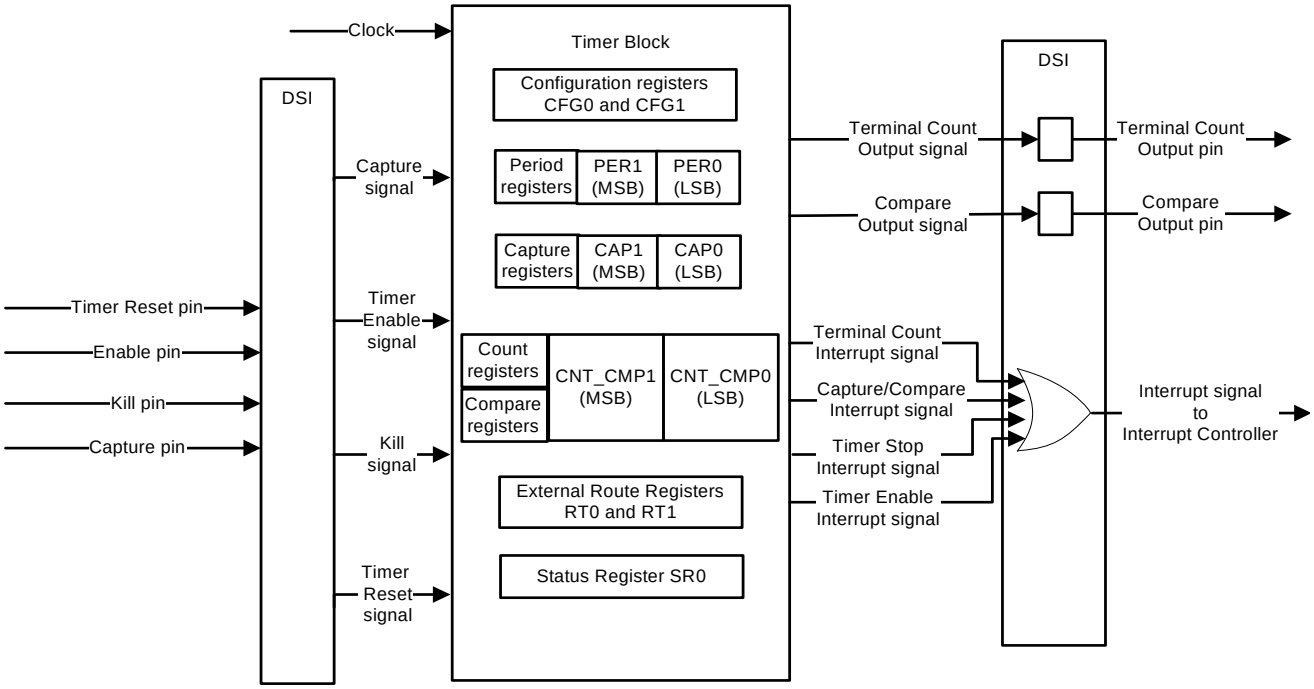
26.1 Features

- 8/16-bit timer/counter/PWM that acts as a down counter
- Supports the following modes:
  - Timer
  - Gated Timer
  - Pulse-width Modulator (PWM)
  - One Shot
- Supports interrupts upon:
  - Terminal count – the final value in the Count register is reached
  - Compare true – the timer value matches with the Compare register
  - Capture – capture of timer value on edge detection in the Capture signal
- Counts when Enable signal is asserted
- Supports the free running timer
- Period reload on start, reset, and terminal count
- Selectable clock source
- Supports kill and dead band features

26.2 Block Diagram

Figure 26-1 on page 212 shows one timer block.

Figure 26-1. Timer Block Diagram



26.3 How It Works

The block receives a clock signal that is selectable from different sources. The block in PSoC devices is a down counter and counts for every rising edge of the input clock. It counts down from the period value to zero. When it reaches zero (terminal count) the period value is reloaded into the count register, and the timer continues to count. If the timer is configured for One Shot mode, the timer stops when it reaches the terminal count.

The timer block can act in various modes, depending on appropriate configuration of the registers:

- Timer
  - Free Run
  - Gated Timer
    - Pulse Width
    - Period
    - Stop on Interrupt
- PWM
- One Shot

The block can be used as a timer to capture time of external event, to measure period and pulse width of the input signal, and to find the time of occurrence of interrupt and as a PWM generation unit.

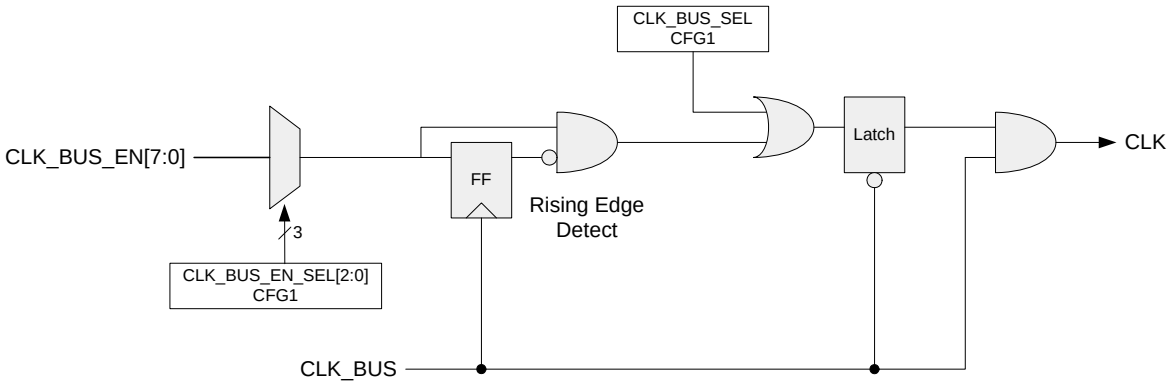
26.3.1 Clock Selection

The block supports the flexibility to select the required clock source. As shown in Figure 26-2 on page 213, the block uses the CLK_BUS frequency, or it is routed through one of the eight selectable clock lines CLK_BUS_EN 0…7, which are synchronous to the clock bus.

Clock selection is done through the Configuration register CFG1. If the BUS_CLK_SEL bit in register CFG1 is set, the block uses the CLK_BUS frequency, instead of the eight selectable digital clock lines.

If the BUS_CLK_SEL bit is set to 0, one of the eight selectable lines is used for the clock. The bits CLK_BUS_EN_SEL in Configuration register CFG1 are set to choose one of eight selectable digital clock lines. The clock for the digital clock lines can be derived from the CLK_BUS or it can be another UDB signal or external clock signal.

Figure 26-2. Clock Selection



26.3.2 Enabling and Disabling Block

The block is enabled or disabled by setting the Enable bit EN in Configuration register TMRx_CFG0. All the required configurations for the block must be done before it is enabled. When the block is enabled, it functions in the configured mode (Timer or PWM). Enabling a block updates the registers with the new configured value. Disabling a block retains the values in the registers until it is enabled again.

- When the EN bit is set, the previous state is cleared and the count register is loaded with the reload value from the period register. The block starts to count.
- When Configuration and Period registers are modified with the EN bit set to ‘1’, the changes go into effect only after the completion of the current running period (at the terminal count).
- When Configuration and Period registers are modified with the EN bit set to ‘0’, the changes go into effect immediately after the EN bit is set to ‘1’.
- When the block is enabled, the count value is loaded with the new reload value, regardless of the state of the register before setting EN = ‘0’.

When the register values are changed after setting EN = ‘0’, the changes go into effect immediately. This is useful during the PWM mode, where the user can change the PWM period or duty cycle immediately.

26.3.3 Input Signal Characteristics

The block has four input signals separate from the clock signal:

- Enable
- Capture
- Timer Reset
- Kill

Input signals are connected to the GPIO through the Digital System Interconnects (DSI). The user maps the input pins to the DSI routing through External Routing register RT0. DSI 1 through DSI 4 within any block can be routed to as any of the above input signals, depending on user mapping. Mapping between DSI routing and the input pins is not fixed. See Figure 26-1 on page 212.

The block has two outputs, terminal count and compare output. They are synchronized to the clock signal. This is done by setting the bits in the external routing register RT1. When the pins are set as asynchronous, the changes go into effect immediately. If synchronous, the changes go into effect during the next clock cycle.

26.3.3.1 Enable Signal

The effect of the enable signal is explained in the timing diagram for each mode. The following characteristics apply:

- Gated timer pulse width mode and period mode take the Enable signal as input.
- Gated timer stop at interrupt mode and PWM mode need an asserted Enable signal to function properly.
- Free run mode is independent of the enable signal.
- Enable signal polarity is reversed by setting the bit INV in configuration register CFG0.
- Use of the capture signal to capture a time instance is valid only when the enable signal is asserted.

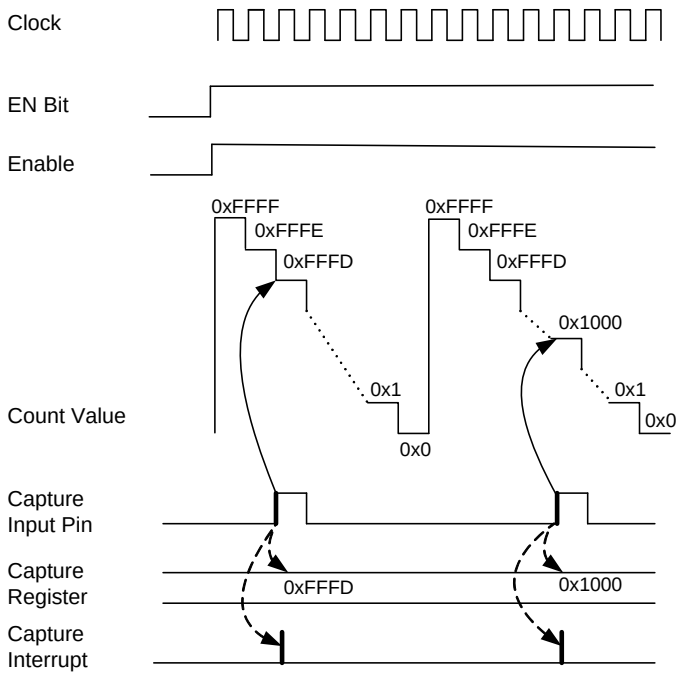
26.3.3.2 Capture Signal

The capture signal is useful to find the time when an event occurs. The capture signal is usually combined with the free run timer mode. For the timer block to respond to the capture signal, the enable signal must be asserted before asserting the capture signal. The following describes the process:

- The time value is captured in the capture register by assertion of the Capture signal for the block.
- Whenever the rising edge of the Capture signal is detected, the count value is captured in the Capture register.
- The capture register is read to find the time when the assertion of Capture signal occurred.
- With every assertion of the Capture signal, a new value is captured to the Capture register.
- An interrupt can be configured to occur at the assertion of the Capture signal. The interrupt bit in the Status register should be unmasked for the capture interrupt to occur. The Capture register value can be read in the capture ISR.
- When using a fixed function timer with interrupt on capture enabled, read the capture register twice. The first reading yields an incorrect value (0xff)”

Figure 26-3 shows the effect of the capture signal (period register = 0xFFFF).

Figure 26-3. Capture Mode Timing Diagram

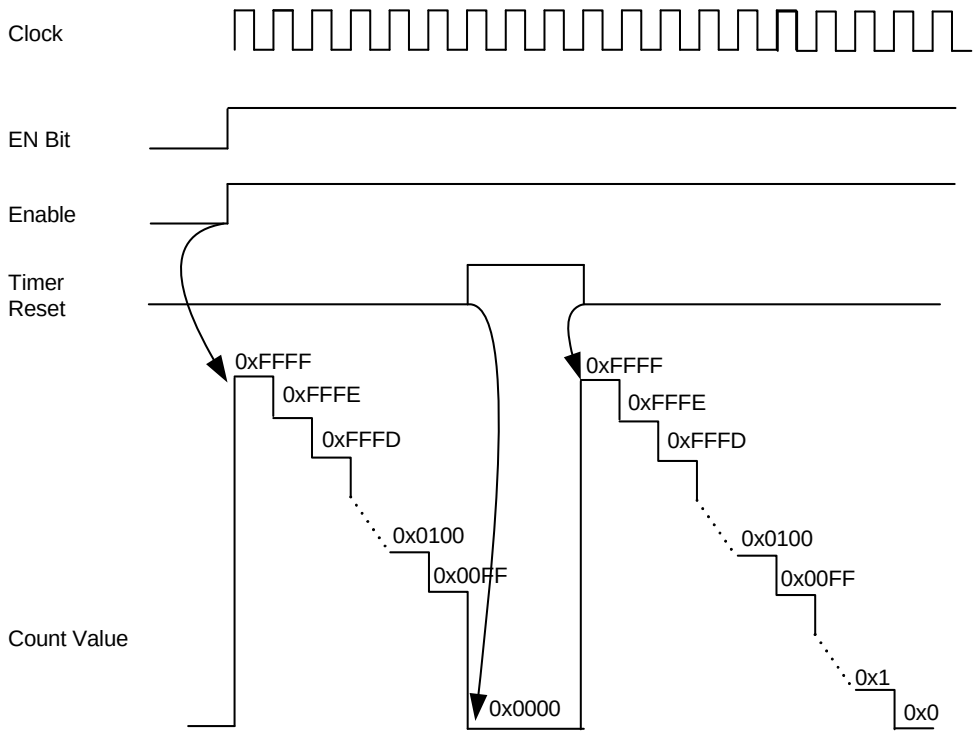


26.3.3.3 Timer Reset Signal

When the timer reset pin is asserted, the count value in the Count register (TMRx_CNT) is set to 0x00. When the timer reset pin is deasserted, the TMRx_CNT register is reloaded with the period value, and it functions in the configured mode. This signal stops the block operation for the time during which the timer reset signal is high and then restarts the operation from the beginning.

Figure 26-4 is a timing diagram for the timer reset signal (Period register = 0xFFFF).

Figure 26-4. Timer Reset Signal Timing Diagram



26.3.3.4 Kill Signal

The Kill signal is valid only during PWM mode. The effect of the kill signal is explained in PWM mode in the sections ahead.

26.3.4 Operating Modes

26.3.4.1 Timer Mode – Free Run Mode

The register configuration for Timer mode is:

- Registers to set – TMRx_CFG0, TMRx_CFG1, TMRx_CFG2
- Bit MODE in TMRx_CFG0 = 0 – Timer mode
- Bits TMR_CFG in TMRx_CFG2 = 0 – Timer runs in continuous mode

The Free Run mode is mainly used to obtain the current system time. Timer operation, automatically forced into the Free Run mode, occurs independent of the state of the Enable pin. This mode is called Free Run because the timer runs even if the state of the Enable pin is low.

The following describes the process:

- The timer is a down counter, and the current time value is stored in the TMRx_CNT registers.
- The reload value for the timer is stored in the Period registers TMRx_PER0 and TMRx_PER1.
- After the count reaches zero (terminal count), the period value is reloaded automatically to the Count registers for the timer to count. The reload value determines the

period for the timer. Two types of output result when the terminal count is reached:

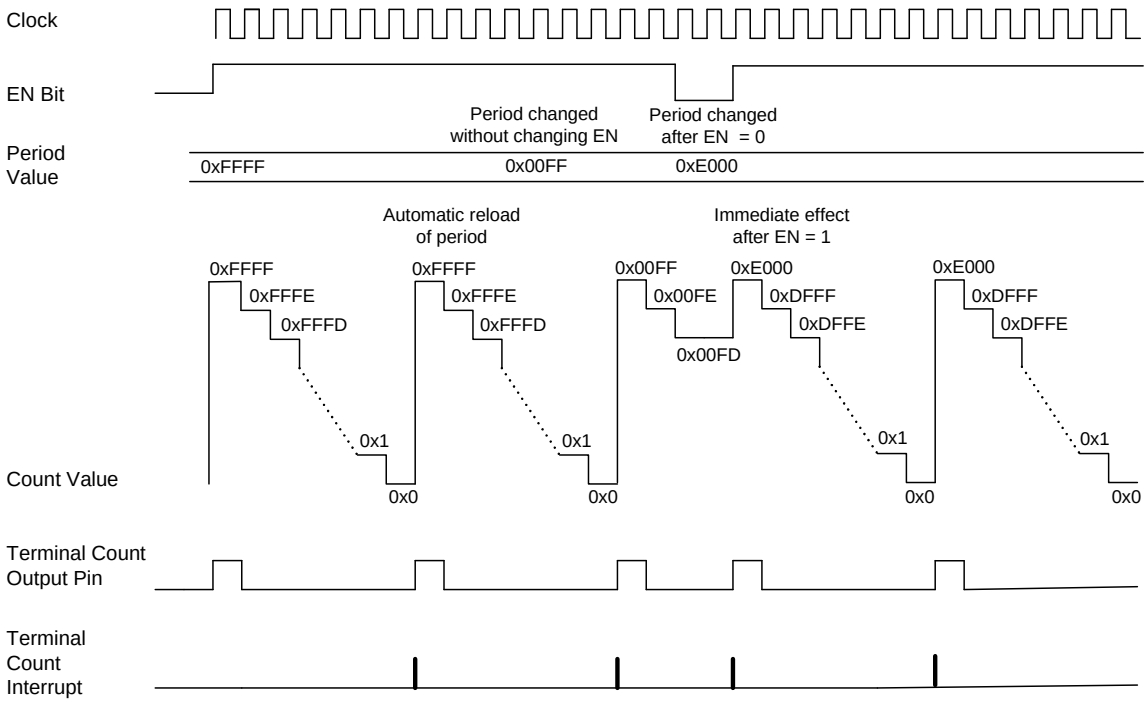
- A terminal count output signal that generates a pulse at the terminal count – The terminal count output signal can be routed to any GPIO through the DSI.
- An interrupt at the terminal count – To initiate an interrupt, the terminal count interrupt in the Status register must be unmasked.

- The current timer value is read from the 8-bit Count registers CNT0 and CNT1. In the case of the 32-bit controller, a 16-bit read of the Capture register can be done.
- In the case of the 8-bit controller, the 8-bit read is done. When an 8-bit read is done for the CNT0 register (LSB) the values of LSB and MSB are automatically captured in the Capture registers. The user can read the Capture register to obtain the 16-bit time value.

Figure 26-5 shows the terminal count output signal and the terminal count interrupt behavior in the Free Run mode (Period register value = 0xFFFF) and illustrates the following behavior.

- Independence of the Timer from the Enable signal for the block
- The effect of changing the Period register with both EN = '1' and EN = '0'
  - When the Period register is changed without setting EN = 1, the effect takes place only after the terminal count.
  - When the Period register is changed with EN = 0, the effect takes place immediately after setting EN = 1.

Figure 26-5. Free Run Mode Timing Diagram



26.3.4.2 Gated Timer Mode

In the Gated Timer mode, the timer does not run continuously; it starts and stops, based on certain criteria. The Gated Timer mode measures some parameters of the input signal, including the period of the input signal, the pulse width of the input signal, and the time after which an interrupt occurs. Depending on the configuration of the register, the following modes are supported:

- Pulse Width
- Period
- Stop on Interrupt

The register configuration for the Counter mode is:

- Registers to set – TMRx_CFG0, TMRx_CFG1, TMRx_CFG2
- Bit MODE in TMRx_CFG0 = 0 – block acts in gated timer mode
- Two bits TMR_CFG in TMRx_CFG2 – gated timer runs in various modes

The modes are achieved by setting the TMR_CFG bits appropriately, as shown in Table 26-1.

Table 26-1. TMR_CFG Bit Settings in Gated Timer Mode

TMR_CFG	Comments
00	Timer runs while EN bit of CFG0 register is set to ‘1’
01	Pulse width count – counts from positive edge to negative edge of TMEREN
10	Period count – counts from one positive edge to the next positive edge of TMEREN
11	Counts from enabled to IRQ

The signal for which the pulse or period is measured is given to the Enable pin.

The following describes the process:

- When the EN bit is set to ‘1’, the Count register is loaded with the period value from the Period register.
- The timer begins counting whenever a rising edge occurs in the enable input. The Count register counts for every clock cycle.
- When the next edge is reached (falling edge in the case of a Pulse Width count and the next rising edge in the case of a Period count), the timer stops to count.
- On reaching the terminal count, the TMRx_CNT register is automatically reloaded with the period value. The timer stop interrupt can be configured to occur when the timer stops to count. The timer stop interrupt enable bit should be unmasked for the interrupt to occur.
- The state of the timer is obtained from the TSTOP bit in the Status register. This sticky bit shows whether the timer has stopped counting; the user must clear the bit.

Pulse Width Mode

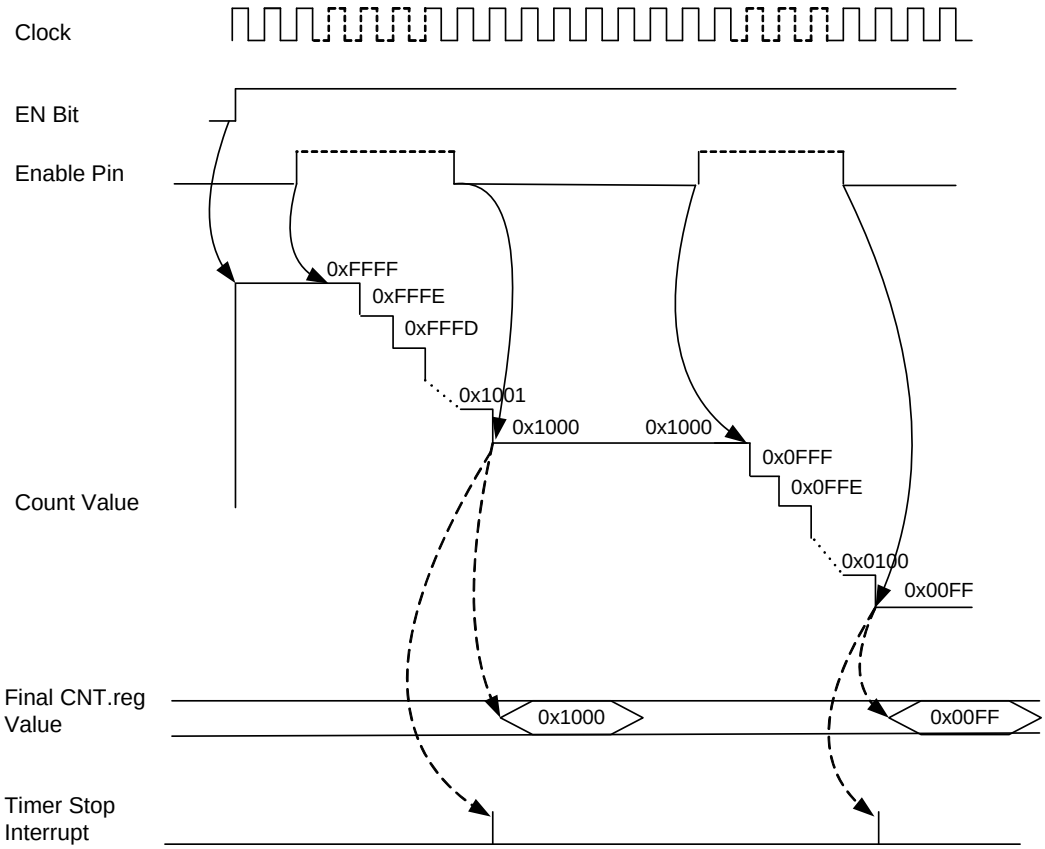
The input signal is given to the Enable pin. The timer begins counting at the rising edge of the Enable signal and stops counting at the falling edge of the Enable signal. There is a latency of one clock cycle for the block to detect the edges.

The difference in the count value before and after the count is equal to the pulse width of the input signal in terms of counts.

The count value is read using 16-bit read in case of a 32-bit controller and 8-bit read in case of a 8-bit controller. During 16-bit read, the count values are read as one 16-bit value and the value is captured in the Capture register. During the 8-bit read, a read of the CNT0 (LSB value) captures the LSB and MSB in the Capture register. The user can read the Capture register to obtain the time value.

Figure 26-6 shows the Gated Timer in Pulse Width mode. In this figure, the One Shot mode is disabled, so the timer will start to count when the next rising edge is encountered. When the One Shot mode is enabled, the timer stops after the falling edge and should be enabled again.

Figure 26-6. Gated Timer in Pulse Width Mode



Period Mode

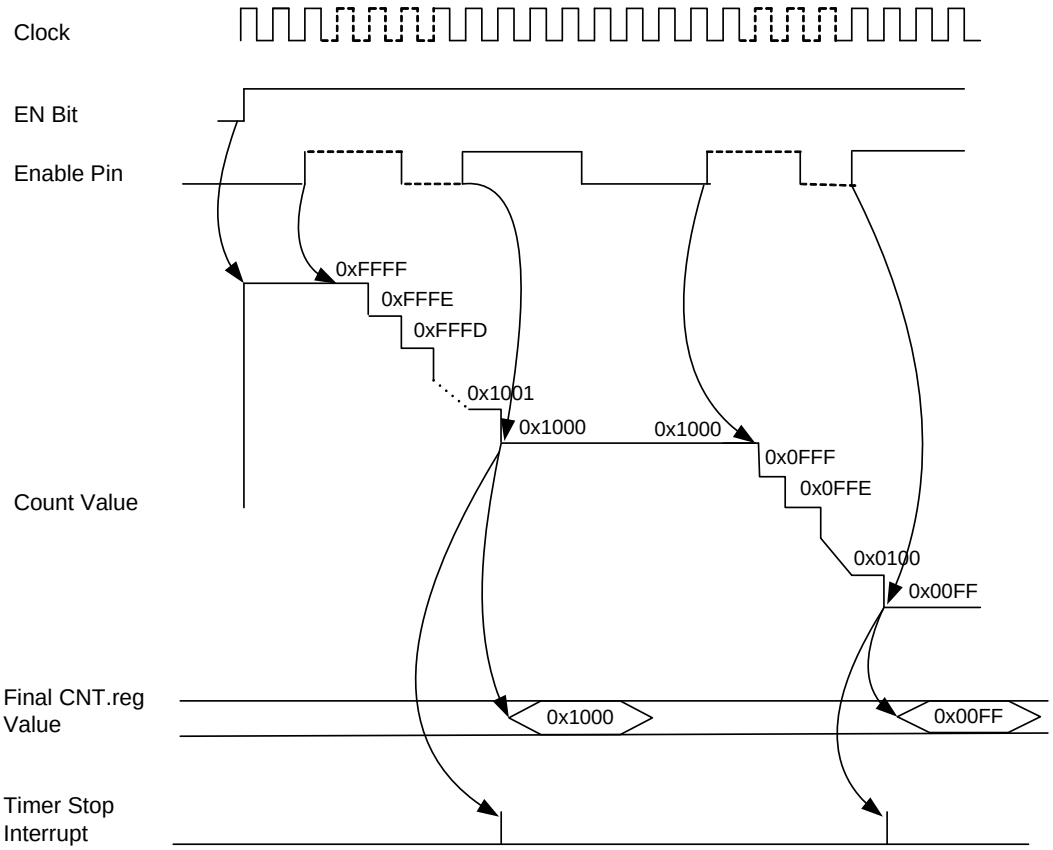
The input signal is given to the Enable pin. In this mode, the timer begins counting at the rising edge of the Enable signal and stops counting at the next rising edge. There is a latency of one clock cycle for the block to detect the edges.

The difference in the count value between the start and the end of the count is equal to the period (in counts) of the input signal.

The count value is read, using a 16-bit read in the case of a 32-bit controller and an 8-bit read in case of an 8-bit controller. During a 16-bit read, the count values are read as one 16-bit value, and the value is captured in the Capture register. During the 8-bit read, a read of the CNT0 register (LSB value) captures the LSB and MSB in the Capture register. The user can read the Capture register to obtain the time value.

Figure 26-7 shows the Gated Timer in Period mode. In this figure, the One Shot mode is disabled; the timer starts to count when encountering the next rising edge after the period calculation. When the One Shot mode is enabled, the timer stops after the second rising edge and should be enabled again.

Figure 26-7. Gated Timer in Period Mode



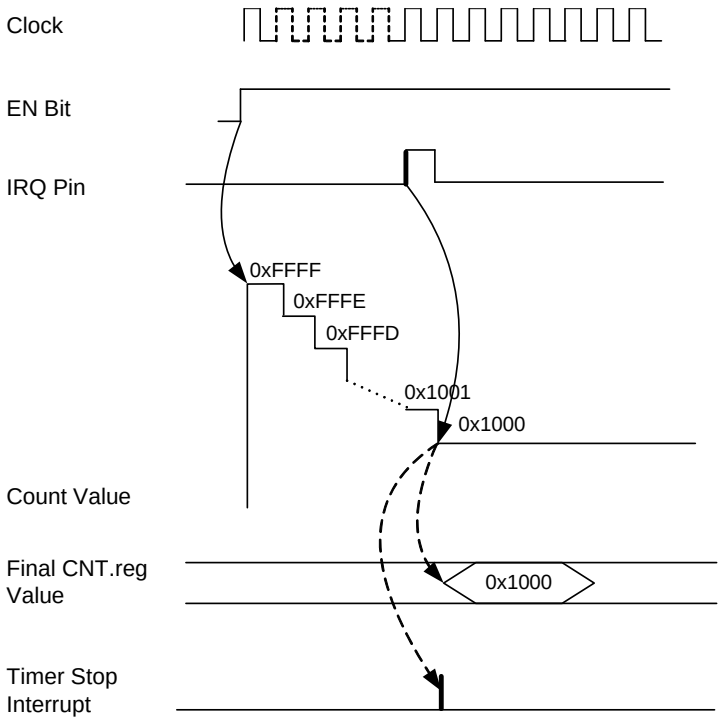
Stop on Interrupt Mode

The Stop on Interrupt mode is useful to stop the timer on occurrence of a specific event for the block. In this mode, the timer starts counting when the EN bit is set to ‘1’ and stops counting when an Interrupt Request (IRQ) is received. The IRQ is any configured interrupt (Terminal Count/Capture, Compare/Timer Stop) of the block. When the IRQ is received, the timer is automatically disabled. The timer should be enabled (EN = ‘1’) to start the timer again.

The timer begins to run only after it is disabled and enabled again. The count value is read using a 16-bit read in case of 32-bit controller and an 8-bit read in case of 8-bit controller. During a 16-bit read, the count values are read as one value and the value is also captured in the Capture register. During the 8-bit read, a read of the CNT0 register (LSB value) captures the LSB and MSB in the Capture register. The user can read the Capture register to obtain the time value.

Figure 26-8 shows the Gated Timer in IRQ mode.

Figure 26-8. Gated Timer in IRQ Mode



26.3.4.3 Pulse-width Modulator Mode

The Pulse-width Modulator (PWM) mode is also called the Comparator mode, because the comparison output is a PWM output with a varying duty cycle and a varying period. The duty cycle depends on the compare type and compare value. The period depends on the Period register. For example, consider a 16-bit PWM block with a clock of 48 MHz. The period value is set to 0x8000 (32768 in decimal). This block gives a PWM period as follows:

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

PWM period for this example = (32768 * 1/48MHz) = 682.7 microsecond

The register configuration for the Comparator mode is:

- Registers to set – TMRx_CFG0, TMRx_CFG1, TMR_CFG2
- Bit MODE in TMRx_CFG0 = 1 – block acts as Comparator
- Three Bits CMP_CFG in TMRx_CFG2 – Comparator runs in various compare modes

The following table lists appropriate register settings.

Table 26-2. Register Settings for Compare Type

CMP_CFG	Comments
000	Timer Value == Comparator Value
001	Timer Value < Comparator Value
010	Timer Value <= Comparator Value
011	Timer Value > Comparator Value
100	Timer Value >= Comparator Value

The Comparator mode compares the timer value and the Compare register value, using either “==”, “<”, “<=”, “>” or “>=” depending on the mode configuration in the CFG2 register.

The following describes the compare process:

1. The timer value begins to count when EN = ‘1’.
2. When the compare is true, the compare output signal is asserted or the compare interrupt signal is asserted. The block continues to count.
3. The CNT register is reloaded with the period value when the terminal count is reached and begins to count compare again.
4. The output of the compare is either the compare output signal or interrupt at the compare.
5. The interrupt occurs when the compare interrupt enable bit is unmasked in the Status register.
6. The compare output signal is routed to the GPIO pin using the DSI.

During the Comparator mode alone, the terminal count output pin acts as the complement to the compare output pin. To use this feature, enable the dead band mode (see Dead Band Feature on page 223). Enable the dead band feature by setting ‘1’ in the DB bit of CFG0. In the Comparator mode, the CNT register cannot be read.

Compare Types

The following is a description of various compare types.

CMP_CFG = 000

The compare output pin generates a pulse when the timer value = the comparator value. In this case, the width of the pulse = one clock cycle. The compare output interrupt signal occurs when the compare value = Timer Value.

CMP_CFG = 001

The compare output pin generates a pulse when the timer value is less than the comparator value. The following describes the event:

- The width of the pulse = one clock cycle x Comparator value.
- The rising edge occurs when the timer value becomes less than the comparator value, such as when the less than condition is met.
- The falling edge of the pulse occurs when the terminal count is reached, such as when the condition changes to false.
- When the comparator is disabled (EN = ‘0’) before the terminal count, the output remains high.
- The Compare output interrupt signal occurs when the timer value is less than the Compare value.

CMP_CFG = 010

The compare output pin generates a pulse when the timer value is less than or equal to the comparator value. The following describes the event:

- The width of the pulse = one clock cycle x (Comparator value + 1).
- The rising edge occurs when the timer value becomes equal to the comparator value, such as when the less than or equal to condition is met.
- The falling edge of the pulse occurs when the terminal count is reached, such as when the condition changes to false.
- When the comparator is disabled (EN = ‘0’) before the terminal count, the output remains high.
- The Compare output interrupt signal occurs when the timer value = Compare value.

CMP_CFG = 011

The compare output pin generates a pulse when the timer value is greater than the comparator value. The following describes the event:

- The width of the pulse = one clock cycle x (Period – Comparator value).
- The rising edge occurs when the Count register is reloaded with the period value, such as when the greater than condition is met.
- The falling edge of the pulse occurs at the end of count value = (Comparator value + 1), such as when the condition changes to false.
- When the comparator is disabled (EN = ‘0’) before the condition changes to false, the output remains high.
- The Compare output interrupt signal occurs after the reload of the period value.

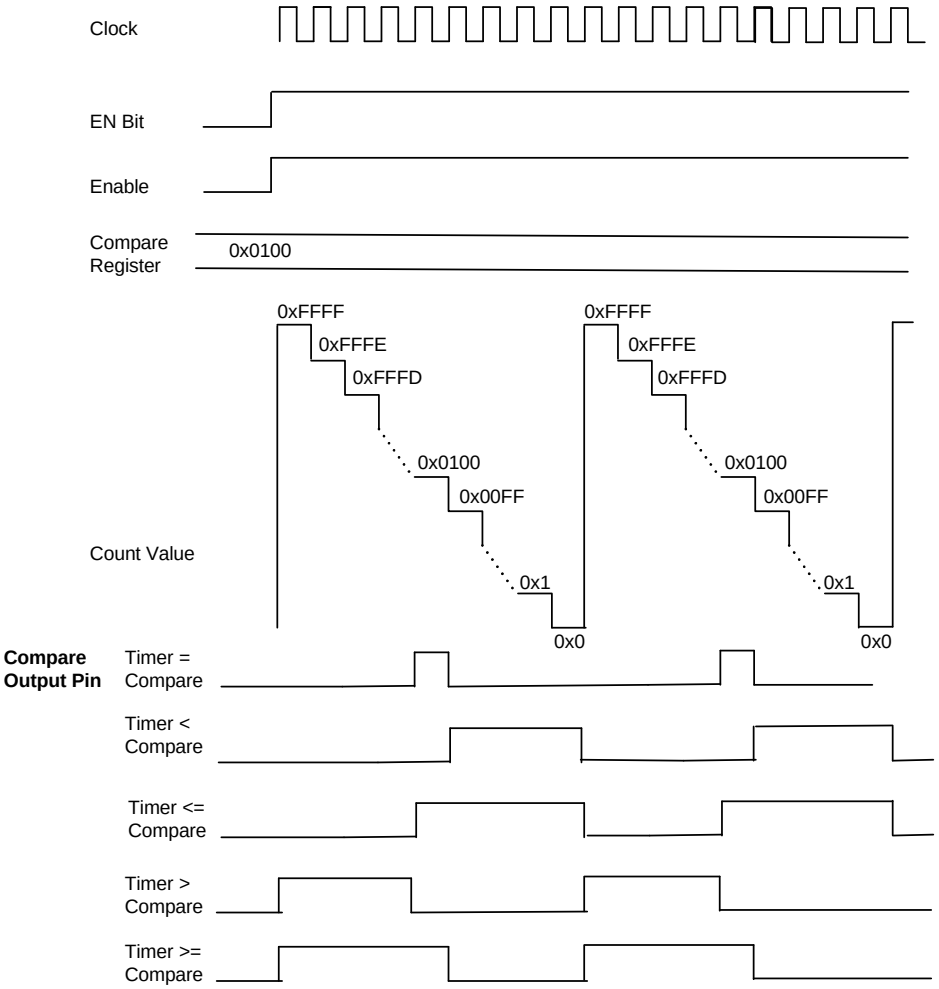
CMP_CFG = 100

The compare output pin generates a pulse when the timer value is greater than or equal to the comparator value. The following describes the event:

- The width of the pulse = one clock cycle x (Period – Comparator value + 1).
- The rising edge occurs when the Count register is reloaded with the period value, such as when the greater than or equal to condition is met.
- The falling edge of the pulse occurs at the end of count value = Comparator value, such as when the condition changes to false.
- When the comparator is disabled (EN = ‘0’) before the condition changes to false, the output remains high.
- The Compare output interrupt signal occurs after the reload of the period value.

Figure 26-9 shows the compare output for various Compare types. The Period register is loaded with 0xFFFF, and the Compare register is loaded with 0x1000.

Figure 26-9. Compare Output for Various Compare Types



On the Fly Duty Cycle Update

Support for multiple comparisons depends on the bit CMP_BUFF in Configuration register CFG0. The following describes the process:

- When the CMP_BUFF is set to ‘1’; the updated comparator value takes effect only after completion of the currently running period. After the terminal count, the new compare value is taken for further comparison. When this mode is used, the PWM block detects only one compare during a period.
- When the CMP_BUFF is set to ‘0’; the updated comparator value takes effect immediately even before the completion of the current running period. This may result in another toggling of the pin even before the completion of current period, thus supporting multiple comparisons.

Dead Band Feature

The dead band feature is used only in Comparator mode. To enable the dead band feature, set the DB bit in Configuration register TMRx_CFG0 to ‘1’. In the dead band mode, the terminal count output pin complements the comparator output pin.

During the dead band period, both compare output and complement compare output are low for a period, determined by the DEADBAND_PERIOD bits in the TMRx_CFG0 register. The dead band feature allows generation of two PWM pulses with non-overlapping outputs. The dead band feature uses a counter. The following describes the process:

- When the comparator asserts the comparator output, it negates the asserted output for the dead band period.
- The dead band period is loaded and counted for the period configured in the DEADBAND_PERIOD bits.
- When the dead band period has completed, the signal is asserted, and the complement is negated.

- A dead band period of zero has no effect.
- When the rate of change in the compare output is less than the dead band period, the immediate change is ignored. Transitions in the compare and complement compare output occur only for the next change in the compare output.
- When the rate of change in the compare output is more than the dead band period, the transitions occur at both compare output changes.

Kill Feature

The Kill signal is mainly used to deactivate the PWM signal in case of fault. Used only in Comparator mode, this signal places the output signals of the block in an unasserted state.

The following describes the process:

- When the Kill signal is asserted, the compare output and the complement of the compare output (if it exists) go to its unasserted state. The terminal count output acts as the complement of the compare output when the dead band feature is enabled.
- When the Kill signal is reasserted, the output signal is restored to its default state. Kill signal duration should be at least one full clock cycle for proper stopping and restoration of the output signal. There is a latency of two clock cycles before the output signal is restored.
- When the Kill signal is asserted, any change in the compare output is ignored, and the deassertion of the Kill signal results only in the previous default state.

26.3.4.4 One Shot Mode

The One Shot mode works in combination with all of the modes specified above. The only difference is that the automatic reload of the Count register with the period does not occur. The block stops working when the required criteria are reached; there is no further reload and running of the block.

The register configuration for the One Shot mode is:

- Bit ONESHOT in Configuration register TMRx_CFG0 = 1 – enabled One Shot mode

The following table shows end criteria (where the block would stop) for each mode. When an end criterion is met, the block stops running and the EN bit is cleared. If the user wants to run the block again, then the block must be enabled (EN = ‘1’):

Table 26-3. Block Stops

Modes	Sub-Types	Criteria
Timer	Free Run Mode	Terminal Count
	Capture Mode	Terminal Count
Counter	Pulse Width Mode	Negative Edge
	Period Mode	Second Positive Edge
	IRQ Mode	IRQ
PWM	CMP_CFG = 000	Terminal Count
	CMP_CFG = 001	Terminal Count
	CMP_CFG = 010	Terminal Count
	CMP_CFG = 011	Terminal Count
	CMP_CFG = 100	Terminal Count

26.3.5 Interrupt Enabling

The block supports four types of interrupt:

- Terminal Count
- Capture/Compare
- Timer Enable
- Timer Stop

These interrupts are enabled by setting the corresponding bits in the Status register; occurrences are stored in the Status registers. Because these Status register bits are sticky, the interrupt request bits must be cleared explicitly by the software on occurrence of the interrupt. See Figure 26-1 on page 212. The process is described as follows:

- Interrupt signals are sent to the Interrupt controller block, where execution is decided and processed.
- The blocks are configured to support any combination of the four interrupts; only one interrupt is supported at a time.
- When another interrupt signal comes during the execution of one interrupt, the new interrupt request is held pending until the previous interrupt execution is completed.
- After the completion of the previous interrupt the new interrupt begins the execution.

Interrupt signals can be of two types:

- Raw Interrupt – Sent whenever the interrupt occurs. These interrupt signals do not wait for the execution of the previous interrupt request; they are continuously sent whenever the interrupt occurs. This type of interrupt signal is called "pulse input" because for every interrupt occurrence, a pulse is sent on the interrupt signal and does not wait for acknowledgement from the CPU.
- Status Interrupt – Sent depending on the status bits in the Status register. When the status bit is set to ‘1’, the interrupt signal is sent. The next interrupt signal is sent

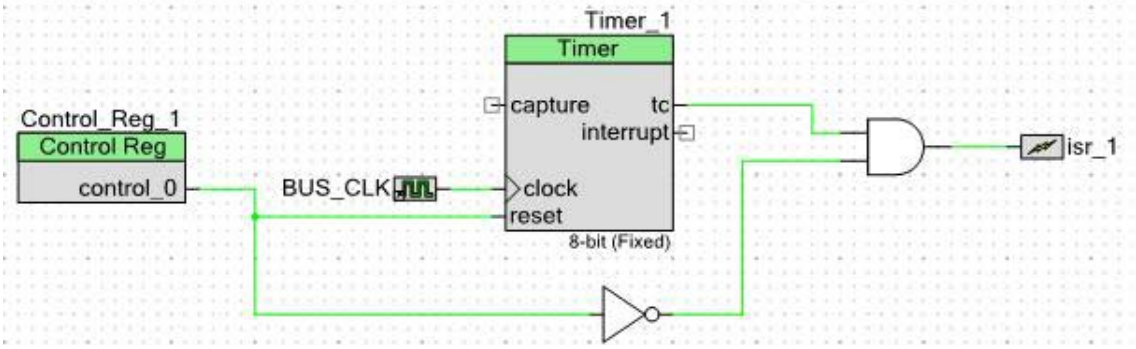
only after the status bit is cleared. The clearing of the status bit is handled by the software inside the interrupt service routine. The interrupt signal is not sent for every interrupt occurrence, but for every new setting of the status bit in the Status register. These types of interrupt allow the user control over the execution of the interrupt. This type of interrupt signal is called “level input” because the signal is asserted and remains asserted until the bit is cleared by the software. The selection of the interrupt signal type is decided using the bit IRQ_SEL, available in the configuration register TMRx_CFG1.

When an external clock is supplied to the fixed-function timer counter with interrupts enabled, the ISR is executed multiple times for a single interrupt event. To avoid this, use a UDB timer component instead.

26.3.6 TC Interrupt Behavior during Reset:

On PSoC 5 devices, the fixed function implementation of the timer differs from the UDB implementation. The TC during reset goes high, whereas in the UDB implementation the TC goes low. The following schematic shows a Fixed Function Timer implementation, which drives the TC low while the reset input is active, thus giving the same functionality as the UDB implementation of the same component.

Figure 26-10. TC Interrupt Behavior



26.3.7 Sleep Mode Behavior

The block supports the following two power saving features:

- When the timer blocks are not accessed by the PHUB, the clock to the AHB interface is gated off, preventing all registers in the block from accessing the clock.
- When the EN bit for a block is not asserted, the clock for that particular block is gated off.

The block retains the values of the Period, Configuration, and Compare registers during the sleep and hibernate states. The Count register value is not retained during the sleep and hibernate states.

26.4 Register Listing

The following table lists the registers.

Table 26-4. Registers

Register Names	Comments	Features
TMRx_CFG0	Configuration Register	Configures Enable of block, One Shot mode, mode of block, Enable pin inversion and dead band features
TMRx_CFG1	Configuration Register	Configures clock, deadband mode, disable on clear, first terminal count, IRQ selection
TMRx_CFG2	Configuration Register	Configures each of the modes, reset on disable, clear on disable, timer enable
TMRx_PER0, TMRx_PER1	Period Register	Retains the reload value
TMRx_CNT_CMP0, TMRx_CNT_CMP1	Count/Comparator Registers	In the Comparator mode, the Count register cannot be read. So the Compare and Count register share the same address space.
TMRx_CAP0, TMRx_CAP1	Capture Register	
TMRx_SR0	Status Register	Hold the status of interrupts and controls the interrupt masking
TMRx_RT0, TMRx_RT1	External Routing Registers	Controls synchronization of the signals and routing of the signals to the DSI

27. I²C

PSoC® 5 devices include a fixed block I²C peripheral designed to interface the PSoC device with an I²C communications bus. Additional I²C interfaces can be created using Universal Digital Blocks (UDBs) and PSoC Creator™. This chapter describes the fixed block I²C interface. For details about the UDB-based interface, see the component datasheet in PSoC Creator. Users not familiar with the I²C interface and the basics of an I²C transaction should refer to 27.3 Background Information.

27.1 Features

The I²C communication block is a serial to parallel processor, designed to connect the PSoC device to a two wire I²C serial communications bus. To eliminate the need for excessive CPU intervention and overhead, this block gives I²C specific support for status detection and framing bit generation.

This block operates as a slave, a master, both, or a multimaster. When active in slave mode, the unit listens for a start condition, or sends or receives data. The master modes works in conjunction with slave mode. The master has the ability to generate a START and STOP condition and determine whether other masters are on the bus. For multimaster mode lock synchronization is supported. To eliminate the need for excessive CPU intervention and overhead, I²C specific support is provided for status detection and generation of framing bits.

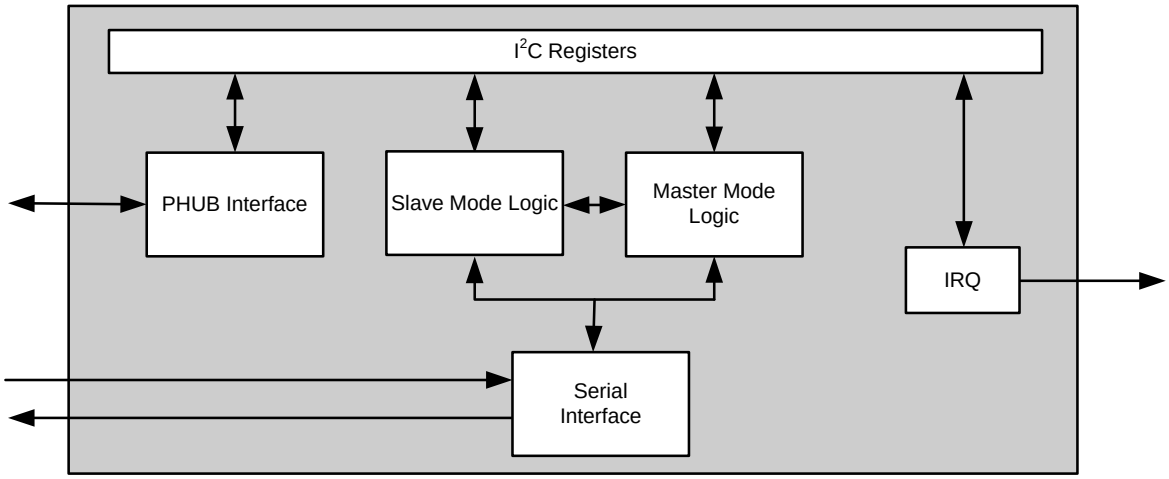
Basic I²C features include:

- Slave/master/multimaster, transmitter and receiver operation
- Byte processing for low CPU overhead
- Provides support for bus status detection and generation of framing bits
- Generates interrupts for a variety of bus events
- Interrupt or polling CPU interface
- Supports bus stalling
- Support for clock rates of up to 1MHz(Fast-mode plus)
- 7 or 10-bit addressing (10-bit addressing requires firmware support)
- SMBus operation (through firmware support - NO SMBus timeout protocol HW support)
- Routes SDA and SCL connection directly to one of two pairs of assigned pins on the SIO port, or through the DSI to any pair of GPIO or SIO pins
- Provides 50 ns glitch filtering

27.2 Block Diagram

Figure 27-1 is a block diagram of the PSoC I²C interface.

Figure 27-1. Block Diagram of the PSoC I²C Interface



27.3 Background Information

The following information is provided to familiarize the user with the I²C bus and the way it transfers data.

27.3.1 I²C Bus Description

The Inter IC, or I²C, bus was developed by Philips Semiconductors (now NXP) to provide a simple means to allow multiple ICs to communicate directly with each other over a common bus. Features of the I²C bus include:

- Only two bus lines are required: (1) serial data (SDA) and (2) serial clock (SCL).
- Serial, 8-bit, bi-directional data transfers can be made at up to 100 kbps in the standard mode, up to 400 kbps in the fast mode and up to 1 Mbps in the fast mode plus. See Figure 27-2 for bus states.
- Devices are connected to the bus using open collector or open-drain output stages, with pull up resistors, for wired AND functions.
- Each slave device connected to the bus is software addressable by a unique address.
- Simple master/slave relationships exist; masters and slaves can operate as either transmitters or receivers.
- Multiple masters are supported, using collision detection and arbitration if two or more masters simultaneously initiate data transfer.

For more information, see the I²C-Bus Specification, and User Manual, Version 03 at http://www.nxp.com/acrobat_download/usermanuals/UM10204_3.pdf.

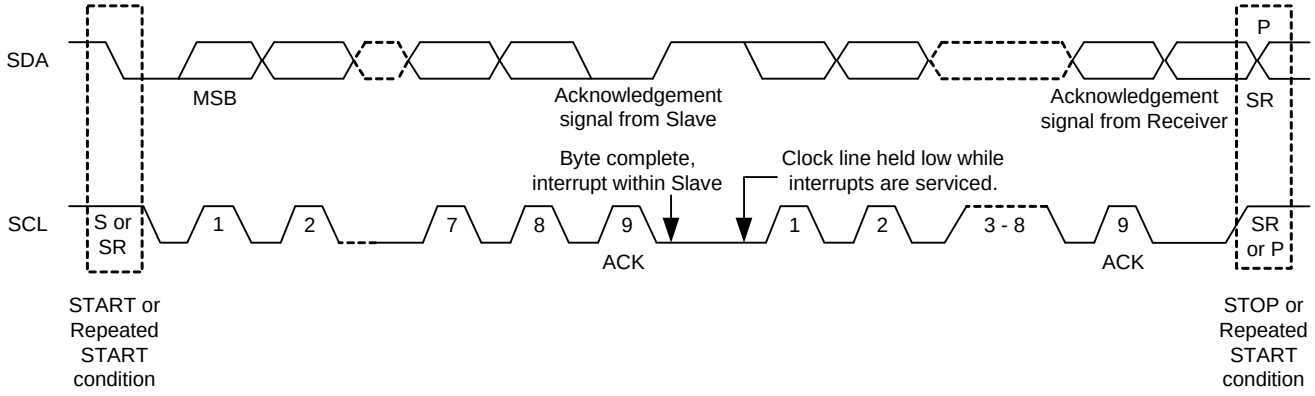
27.3.2 Typical I²C Data Transfer

In a typical I²C transaction, the following sequence takes place:

1. A master device controls the SCL line and generates a Start condition followed by a data byte. The data byte contains a 7-bit slave address and a Read / Write (RW) bit. The bit sets the direction of the data transfer, relative to the master. It is high for read and low for write.
2. The slave device recognizes its address and acknowledges (ACK) the byte by pulling the data line low during the ninth bit time.

   If the slave does not respond to the first data byte with an ACK, a Stop condition is generated by the master to terminate the transfer. A Repeated Start condition may also be generated for a retry attempt.
3. The master transmits or receives an indeterminate number of bytes, depending on the RW direction.
4. When the transfer is complete, the master generates a Stop condition.

Figure 27-2. I²C Transfer of a Single Data Byte, With Clock Stretching by a Non-PSoC Slave



27.4 How It Works

The PSoC 5 I²C interface provides support for bus status detection and generation of framing bits. It can operate at up to fast mode plus speeds, in these modes:

- Slave – The interface listens for Start and Stop conditions to begin and end data transfers.
- Master – The interface generates the Start and Stop conditions and initiates data transfers by transmitting a slave address.
- Multi-Master – The interface provides clock synchronization and arbitration to allow multiple masters on the same bus. Slave mode can be enabled at the same time as master mode.

For details about the operation of these three modes, see 27.4.6 Operating the I²C Interface on page 230 and sections 27.5 Slave Mode Transfer Examples on page 233, 27.6 Master Mode Transfer Examples on page 236, and 27.7 Multi-Master Mode Transfer Examples on page 238.

The I²C interface supports either 7-bit or 10-bit addressing. In slave mode, 7-bit address detection is done by the CPU in firmware. A 10-bit address detection must be done by the CPU in firmware. In master mode, 10-bit address generation must be done by the CPU in firmware.

27.4.1 Bus Stalling (Clock Stretching)

After a byte is transferred on the I²C bus, a slave device may need time to store the received byte or to prepare another byte to be transmitted. In that case, the slave can hold the SCL line low before or after acknowledgment of a byte, which forces the master into a wait state until the slave is ready. This operation is known as stalling the I²C bus. Some devices in master mode may not support bus stalling; the system design should be checked before using bus stalling in slave mode.

The I²C interface can stall the bus on every received address and on every completed byte transfer. After a byte is transferred, the CPU has half of the SCL clock cycle period to write/read the next byte before stalling begins. SCL is released when the next byte is written/read, and the next byte transfer begins.

27.4.2 System Management Bus

The System Management Bus (SMBus) is a bus definition based on the I²C bus. It is similar to, and generally a subset of, the I²C bus. For more information, see the SMBus Specification, Version 1.1. The I²C interface generally supports SMBus, although additional firmware support may be required.

27.4.3 Pin Connections

The I²C block controls the data (SDA) and the clock (SCL) to the external I²C interface, through direction connections to the GPIO/SIO pins. When I²C is enabled, these GPIO/SIO pins are not available for general purpose use.The SDA and SCL connections of the I²C interface can be directly routed to one of two pairs of assigned pins on the SIO port. The connections can also be routed through the DSI to any other pair of GPIO or SIO pins. In all cases, the GPIO or SIO pins must be configured for “Open Drain, Drives Low” mode (see 21.3.2.5 Open Drain, Drives High and Drives Low on page 130).

27.4.4 I²C Interrupts

The I²C interface generates interrupts for these conditions:

- Byte transfer (receive or transmit) complete
- I²C bus Stop condition detected
- I²C bus error detected

The I²C interface cannot generate DMA requests.

27.4.5 Control by Registers

The I²C interface is controlled by reading and writing a set of configuration, control, and status registers listed in the following table. These 8-bit wide registers are used to turn the I²C interface on or off, connect to I/O pins, set the baud rate, provide status and control for the data transfer processes, and monitor for exceptions.

Table 27-1. I²C Registers

Register	Usage
I2C_CFG	Configuration – basic operating modes, oversample rate, and selection of interrupts.
I2C_XCFG	Configuration – configures enhanced features.
I2C_CLK_DIV1 I2C.CLK_DIV2	Clock Divide – sets baud rate (along with oversample rate in I2C_CFG).
I2C_CSR	Control / Status – used to control the flow of data bytes and to keep track of the bus state during a transfer.
I2C_MCSR	Master Mode Control / Status – implements I²C framing controls and provides bus status.
I2C_ADR	Slave Address – for slave address recognition in hardware, holds the 7-bit slave address.
I2C_D	Data – provides read / write access to the data shift register.

27.4.6 Operating the I²C Interface

Operate the I²C interface in this manner:

1. Turn on the I²C interface by setting the I2C_XCFG bit 7, csr_clk_en.
2. To route the SDA and SCL to the desired pin pair, set up I2C_CFG as described in Table 27-2.
3. Select the baud rate (SCL clock frequency) by setting the I2C_CFG register, bit 2and the I2C_CLK_DIV1 and I2C.CLK_DIV2 registers as shown in Table 27-3. The formula to determine the baud rate is:

   Baud Rate = Bus clock frequency / (Clock Division Factor * Oversample Rate)
4. Enable the desired mode of operation, following the instructions in 27.4.6.1 Slave Mode on page 231, 27.4.6.2 Master Mode on page 232, or 27.4.6.3 Multi-Master Mode on page 233.

Table 27-2. Configuration of the I2C_CFG Register, Bit 7

Pin Pair	Port Pins[a]	Register Settings
I2C0	P12[4,5]	I2C_CFG[6] = 1, I2C_CFG[7] = 0
I2C1	P12[0,1]	I2C_CFG[6] = 1, I2C_CFG[7] = 1
Any other GPIO / SIO pin pair	Selectable	I2C_CFG[6] = 0, other DSI and GPIO registers according to pin pair selected

a. The port pins used must be configured to “Open drain, Drives Low” mode (mode 4). The SIO pins are more suited for this purpose than the GPIO pins as the SIO pins have higher current sink capability and over voltage tolerance.

Table 27-3. Configuration For I²C Baud Rate[a]

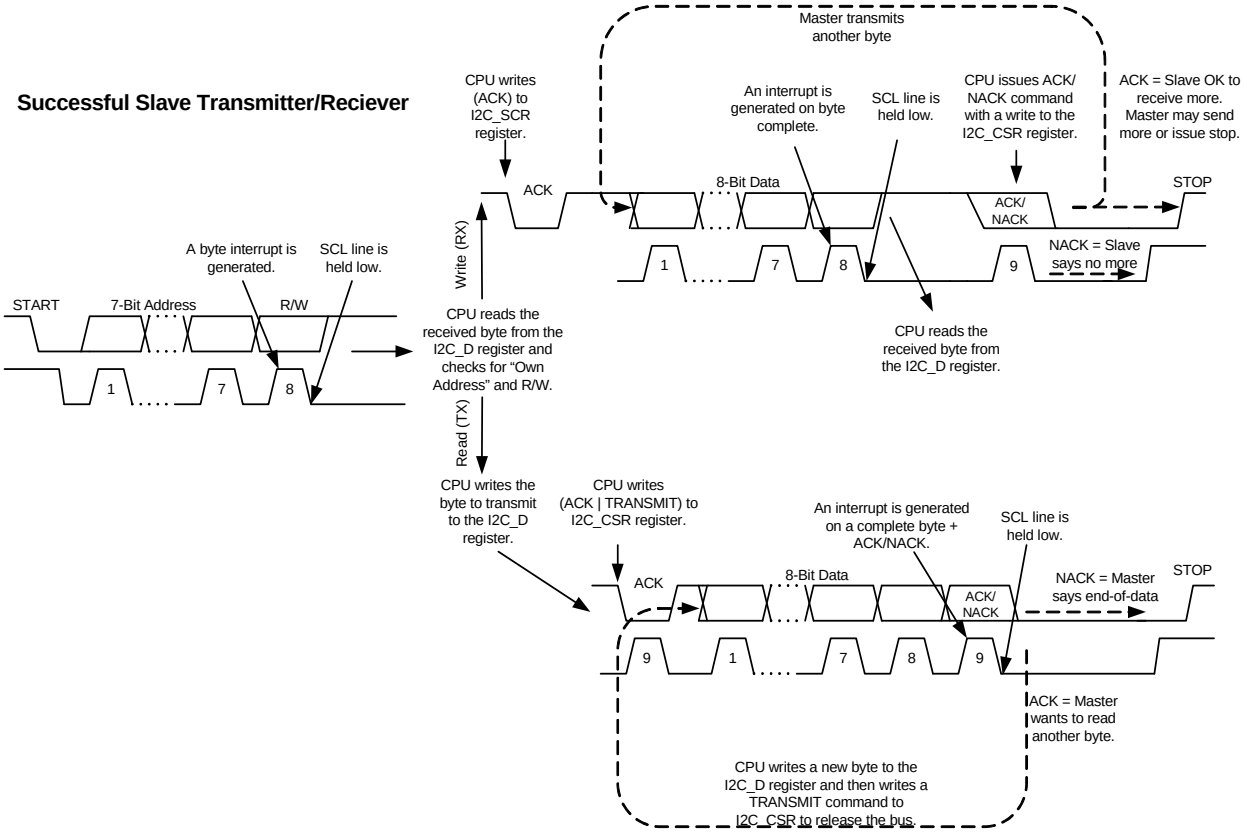
IMO Bus Clock (MHz)	I2C Mode	Oversample Rate	Divide Factor		SCL (kHz)
			I2C_CLK_DIV2[1:0]	I2C.CLK_DIV1[7:0]	
3	Standard	16	(0)2'b00	(2)8'b00000010	93.75
6	Standard	32	(0)2'b01	(2)8'b00000010	93.75
6	Fast	16	(0)2'b02	(1)8b'00000001	375
12	Standard	32	(0)2'b03	(4)8'b00000100	93.75
12	Fast	16	(0)2'b04	(2)8'b00000010	375
24	Standard	32	(0)2'b05	(8)8b'00001000	93.75
24	Fast	16	(0)2'b06	(4)8'b00000100	375
48	Standard	32	(0)2'b07	(16)8b'00010000	93.75
48	Fast	16	(0)2'b08	(8)8b'00001000	375
48	Fast plus	16	(0)2'b09	(3)8b'00000011	1000
67	Standard	32	(0)2'b10	(21)8b'00010101	99.7
67	Fast	16	(0)2'b11	(11)8b'00001011	381
67	Fast plus	16	(0)2'b12	(4)8'b00000100	1046
80	Standard	32	(0)2'b13	(25)8b'00011001	100
80	Fast	16	(0)2'b14	(13)8b'00001101	385
80	Fast plus	16	(0)2'b15	(5)8b'00000101	1000

a. Other values of bus clock, oversample rate and clock divider cause the baud rate to be scaled accordingly.

27.4.6.1 Slave Mode

To enable slave mode operation, set I2C_CFG bit 0, Enable Slave. See Figure 27-3.

Figure 27-3. Slave Mode Operation



In slave mode, the I2C interface continually monitors the bus for a Start condition. When a Start condition is detected, the following ensues.

1. The first byte, which is the Address / RW byte, starts to be shifted in. When all eight bits have been received, a Byte Complete status is generated.
2. On the following low of the clock, the bus is stalled by holding SCL low, until the address byte is read and compared. An ACK or NAK is then issued, based on that comparison.
3. If there is an address match, the RW bit determines the direction of the data transfer, as shown in the two branches of Figure 27-3. After each byte is received, or when a new byte can be transmitted, a Byte Complete status is generated, and SCL is held low to stall the bus until the CPU handles the interrupt and transfers the next byte. If the interrupt must be disabled for some reason (for example, VIDAC update), the interrupt disabled time should be kept as short as possible so that the SCL line can be released after servicing the interrupt.
4. When transmitting bytes, the slave receives an ACK/ NAK from the master for each byte sent.

   ACK is a signal that the master wants another byte.

   NAK or a Stop condition is a signal that the master doesn't want any more bytes – the CPU should let the I2C interface go to an idle state.
5. When receiving bytes, the slave ACKs/NAKs each byte received from the master.

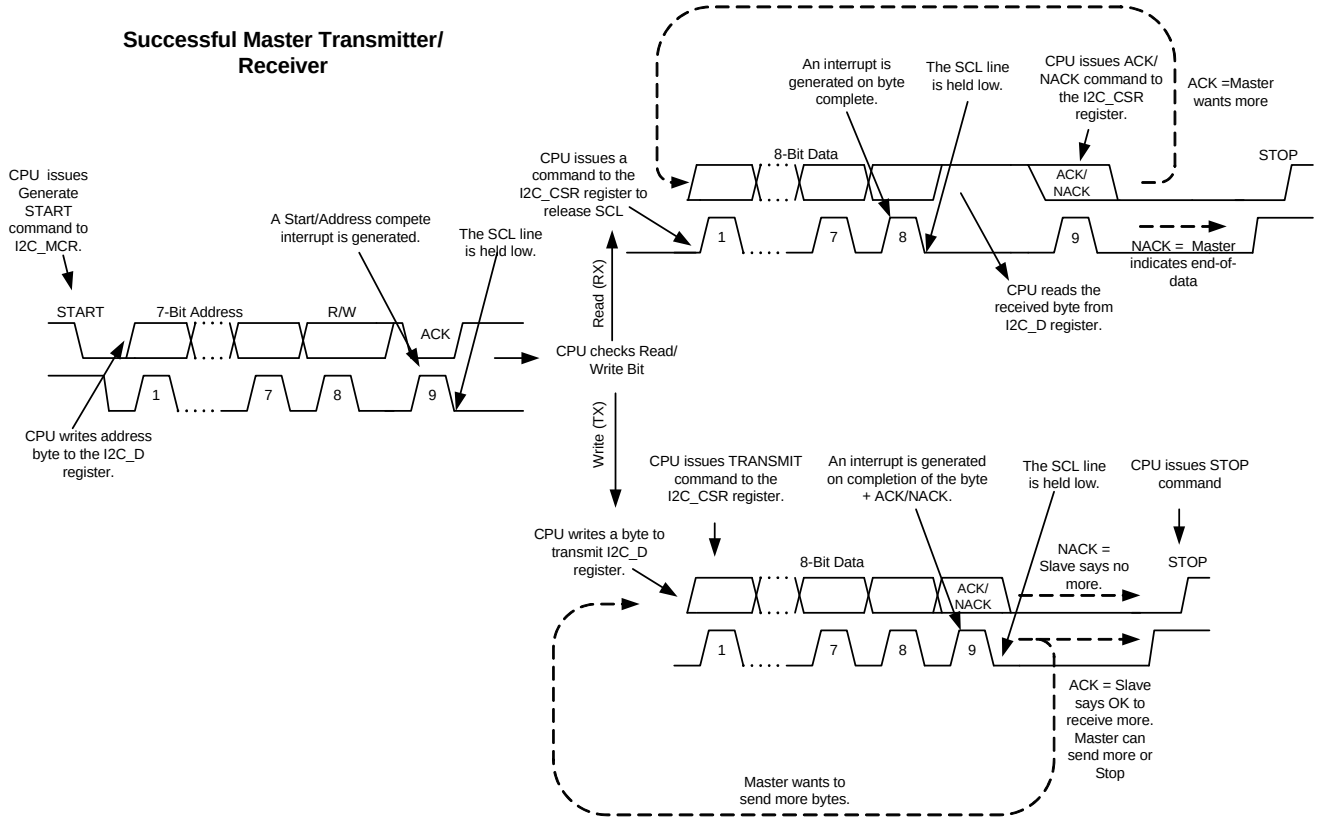
   ACK is a signal that the slave can accept another byte.

   NAK is a signal that no more bytes can be accepted – after generating a NAK the CPU should then let the I2C interface go to an idle state.
6. Data transfer is complete when the master generates a Stop condition.
7. At anytime when a Stop condition or Bus Error is detected, the I2C interface is automatically reset to an idle state. Do not power down or disable the fixed-function I2C block while the bus is active because this could stall the bus. Instead, you should wait until the bus is free, that is, both the SCL and SDA lines are released and pulled up.

27.4.6.2 Master Mode

To enable master mode operation, set the I2C_CFG bit 1, Enable Master. See Figure 27-4.

Figure 27-4. Master Mode Operations



If Enable Slave is not set, the I2C interface is in Master Only mode and ignores all externally generated Start conditions.

Operation in master mode is as follows:

1. To start a transfer, the CPU writes the slave address/ direction byte to I2C_D and sets I2C_MCSR bit 0, Start Gen (or bit 1, Restart Gen).

   In a single-master environment the Start condition is successfully generated, the byte is transmitted, and a Byte Complete is generated. If the byte is ACKed by the slave, data bytes can be sent or received as shown in the two branches of Figure 27-4.
2. When transmitting bytes, the master receives an ACK/ NACK from the slave for each byte sent.

   ACK is a signal that the slave can accept another byte.

   NACK is a signal that no more bytes can be accepted.
3. When receiving bytes, the master ACKs/NACKs each byte received from the slave.

   ACK is a signal that the master wants another byte.

   NACK is a signal that the master is done accepting bytes.
4. When data transfer is complete, the CPU issues a Stop command. The I2C interface generates a Stop condition and goes to an idle state.

   Instead of a Stop condition, the CPU can issue a Restart command, and another transfer is immediately started.

   The fixed-function I2C block does not support zero-byte transfers in master mode. In master mode, if a Stop or Restart condition is set immediately after the Start condition, the module generates the Stop/Restart condition only after the address field is sent (master also sends 0xFF data byte after address for data write). The clock line remains low after generating the Stop/Restart condition. To avoid this condition, do not set the Stop/Restart

condition immediately after Start; transfer at least a byte and set Stop/Restart after NAK or ACK.

Do not power down or disable the fixed-function I2C block while the bus is active because this could stall the bus. Instead, you should wait until the bus is free, that is, both the SCL and SDA lines are released and pulled up.

27.4.6.3 Multi-Master Mode

Multi-master mode becomes enabled when Master mode is enabled by setting the I2C_CFG bit 1, Enable Master.

In Multi-master mode, the CPU starts the transfer in the same manner as in a single-master environment. However, before generating a Start condition, the master must monitor the Bus Free bit in I2C_MCSR, and wait until the I2C bus is free.

After a Start condition is generated other outcomes may result, causing the CPU to delay or abort the transfer:

- Another master in a multimaster environment has generated a valid Start, and the bus is now busy. The Start condition is not generated. The resulting behavior depends upon whether Slave mode is enabled.
  - Slave mode is enabled – A Byte Complete interrupt is generated. When reading I2C_MCSR, the master sees that the Start Gen bit is still set and that I2C_CSR has the Address bit set, indicating that the block has been addressed as a slave. The firmware may then ACK the address to continue the transfer as a slave, or NACK the address.
  - Slave mode is not enabled – The Start Gen bit remains set, and the transfer is delayed until the bus becomes free. A Byte Complete is generated when the Start condition has been generated and the address byte has been transmitted.
- The Start condition is generated, but the master loses arbitration to another master. The resulting behavior depends upon whether Slave mode is enabled.
  - Slave mode is enabled – A byte complete interrupt is generated. When reading I2C_MCSR, the master sees that the Start Gen bit is clear, indicating that the Start condition was generated. However, the Lost Arb bit is set in I2C_CSR. The Address status is also set, indicating that the block has been addressed as a slave. The firmware may then ACK the address to continue the transfer as a slave, or NACK the address.
  - Slave mode is not enabled – A Byte Complete interrupt is generated. The Start Gen bit is clear and the Lost Arb bit is set. The hardware waits for a command from the CPU, stalling the bus if necessary. The master clears I2C_CSR to release the bus and allow the transfer to continue, and the I2C interface goes back to idle mode. The firmware can then retry the transfer when the bus becomes free again.

The fixed-function I2C block does not support zero-byte transfers in master mode. In master mode, if a Stop or Restart condition is set immediately after the Start condition, the module generates the Stop/Restart condition only after the address field is sent (master also sends 0xFF data byte after address for data write). The clock line remains low after generating the Stop/Restart condition. To avoid this condition, do not set the Stop/Restart condition immediately after Start; transfer at least a byte and set Stop/Restart after NAK or ACK.

Do not power down or disable the fixed-function I2C block while the bus is active because this could stall the bus. Instead, you should wait until the bus is free, that is, both the SCL and SDA lines are released and pulled up.

Fixed-block I2C does not support undefined bus conditions. Avoid these conditions or use the UDB-based I2C component instead.

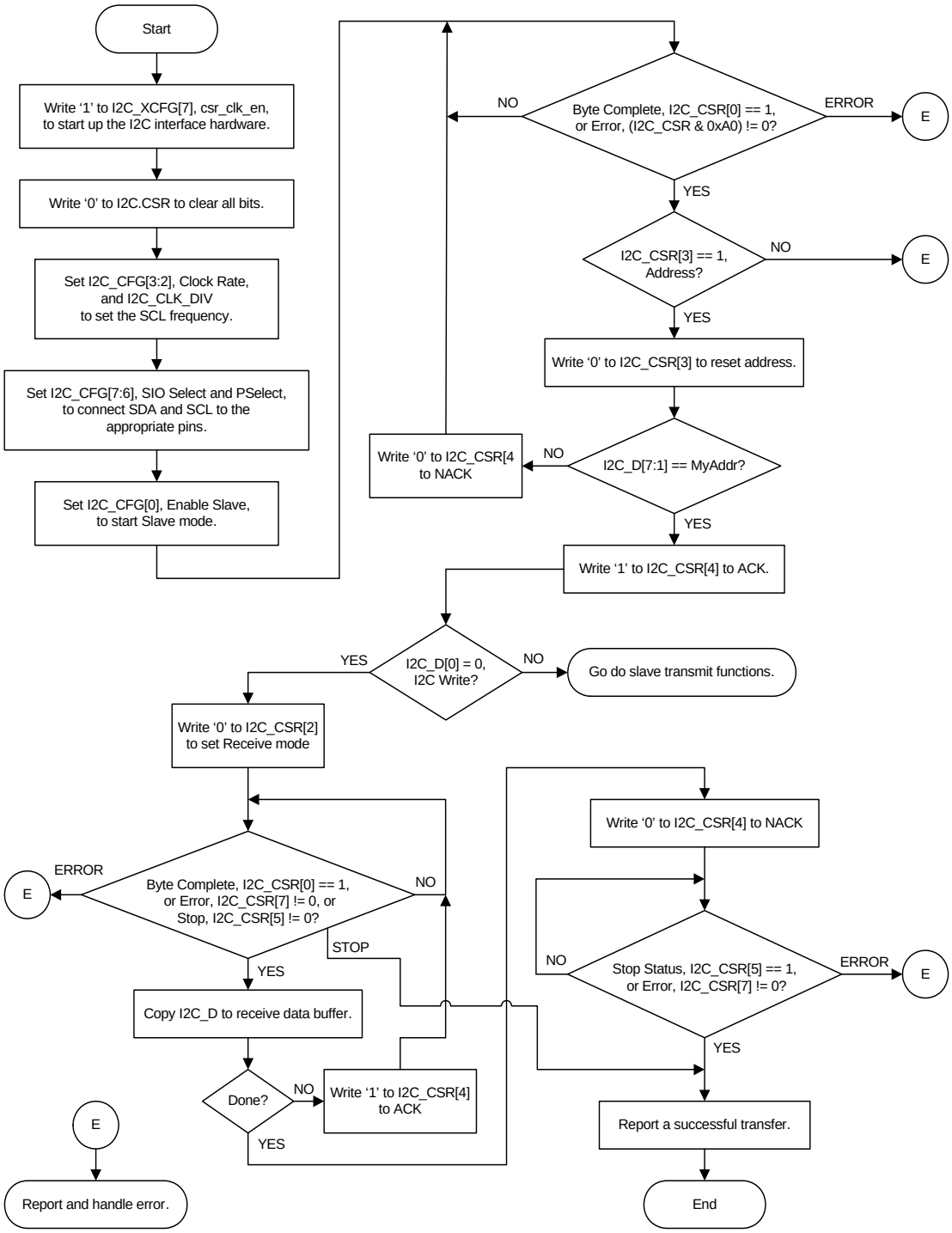
27.5 Slave Mode Transfer Examples

Slave mode receives or transmits data, as described in this section.

27.5.1 Slave Receive

A slave receive operation is accomplished as shown in Figure 27-5.

Figure 27-5. Slave Receive Operation Sequence

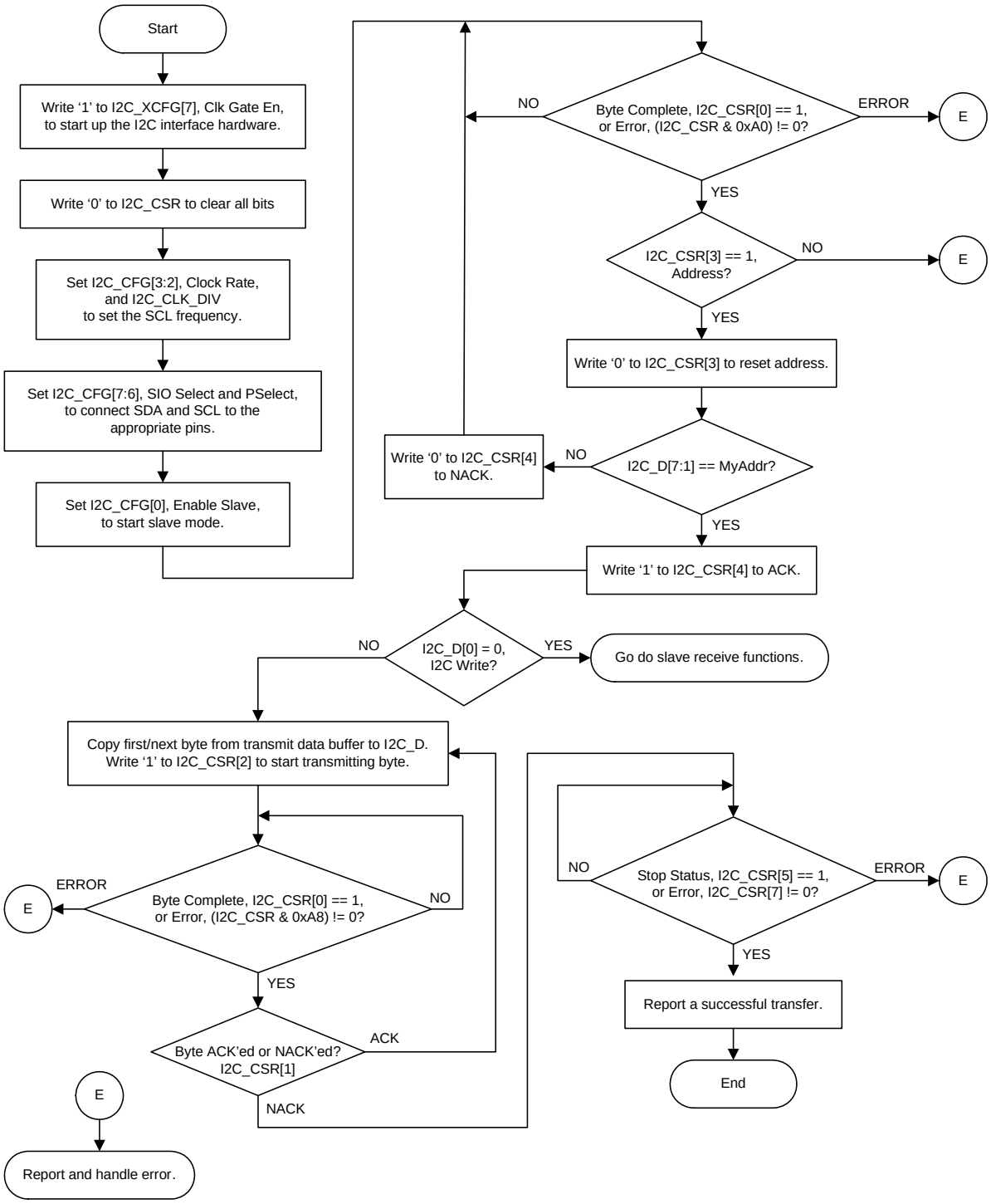


27.5.2 Slave Transmit

A slave transmit operation is accomplished is accomplished as shown in Figure 27-6.

Figure 27-6. Slave Transmit Operation Sequence

Flow Chart for Slave Transmit



Note that, instead of waiting for Byte Complete or Error, an interrupt can be generated for each of these conditions, as well as for the I2C Stop condition. The interrupt handler can then do some or all of the functions shown.

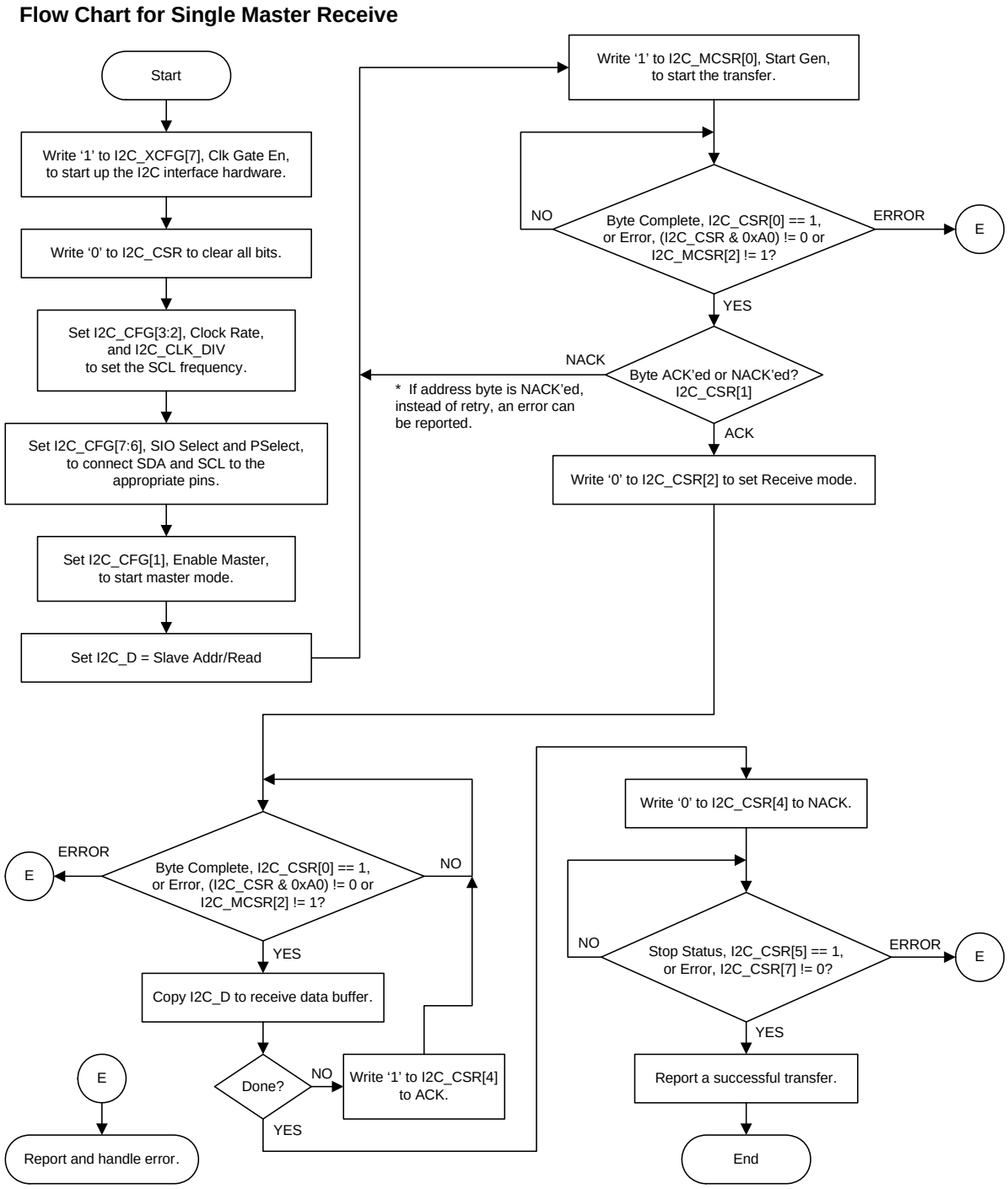
27.6 Master Mode Transfer Examples

Master mode receives or transmits data, as described in this section.

27.6.1 Single Master Receive

A master receive operation in a single-master system is accomplished as shown in Figure 27-7.

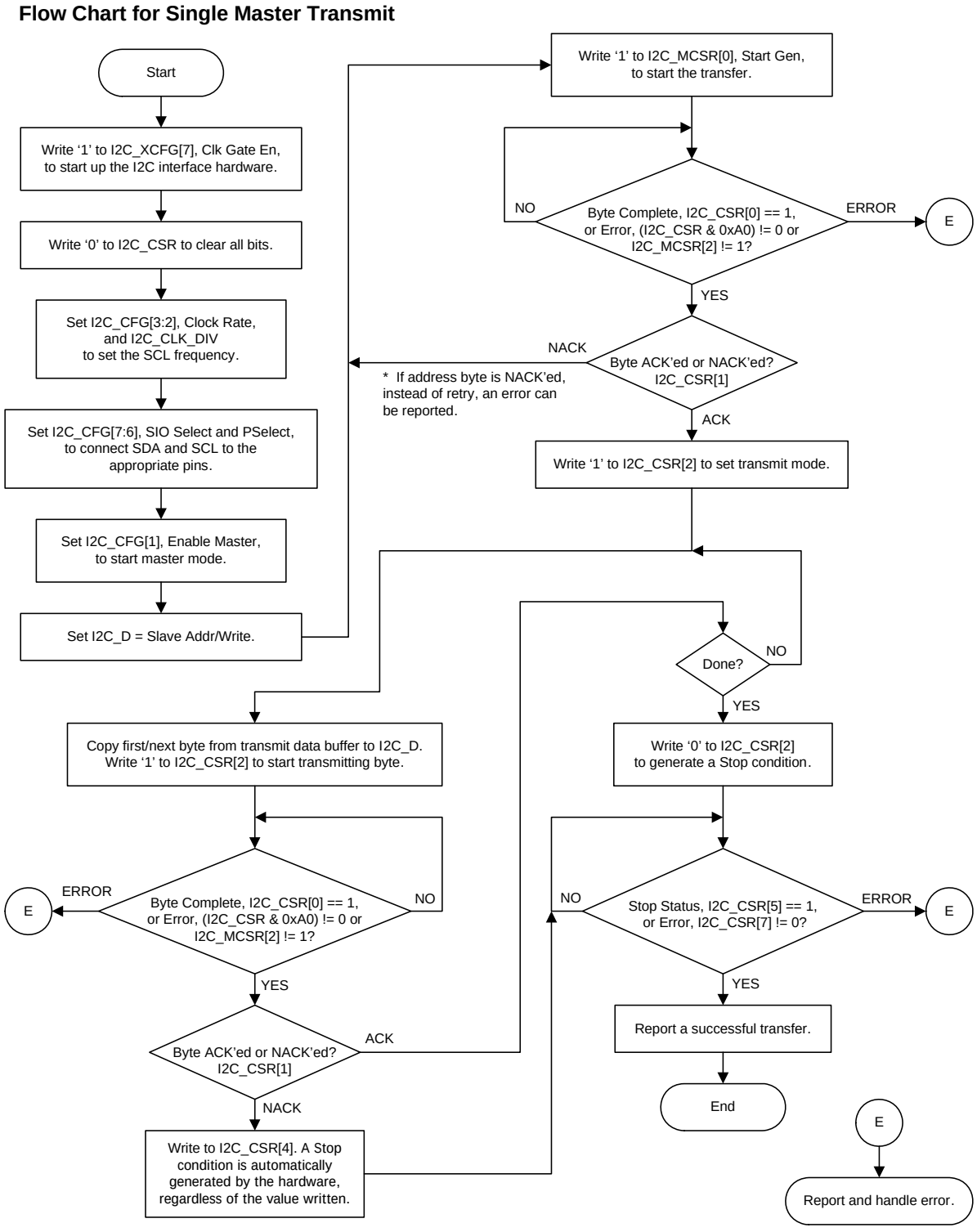
Figure 27-7. Single Master Mode Receive Operation



27.6.2 Single Master Transmit

Figure 27-8 illustrates the process by which you generate a master transmit operation in a single master system.

Figure 27-8. Single Master Mode Transmit Operation



Defining single master operations allows the following assumptions to be made:

- There is no need to check for bus busy (I2C_MCSR[3]) or Lost Arb (I2C_CSR[6]).
- There is no need to Enable Slave (I2C_CFG[0]) when enabling the master mode, as the interface will never be forced into slave mode due to bus busy or lost arbitration.

27.7 Multi-Master Mode Transfer Examples

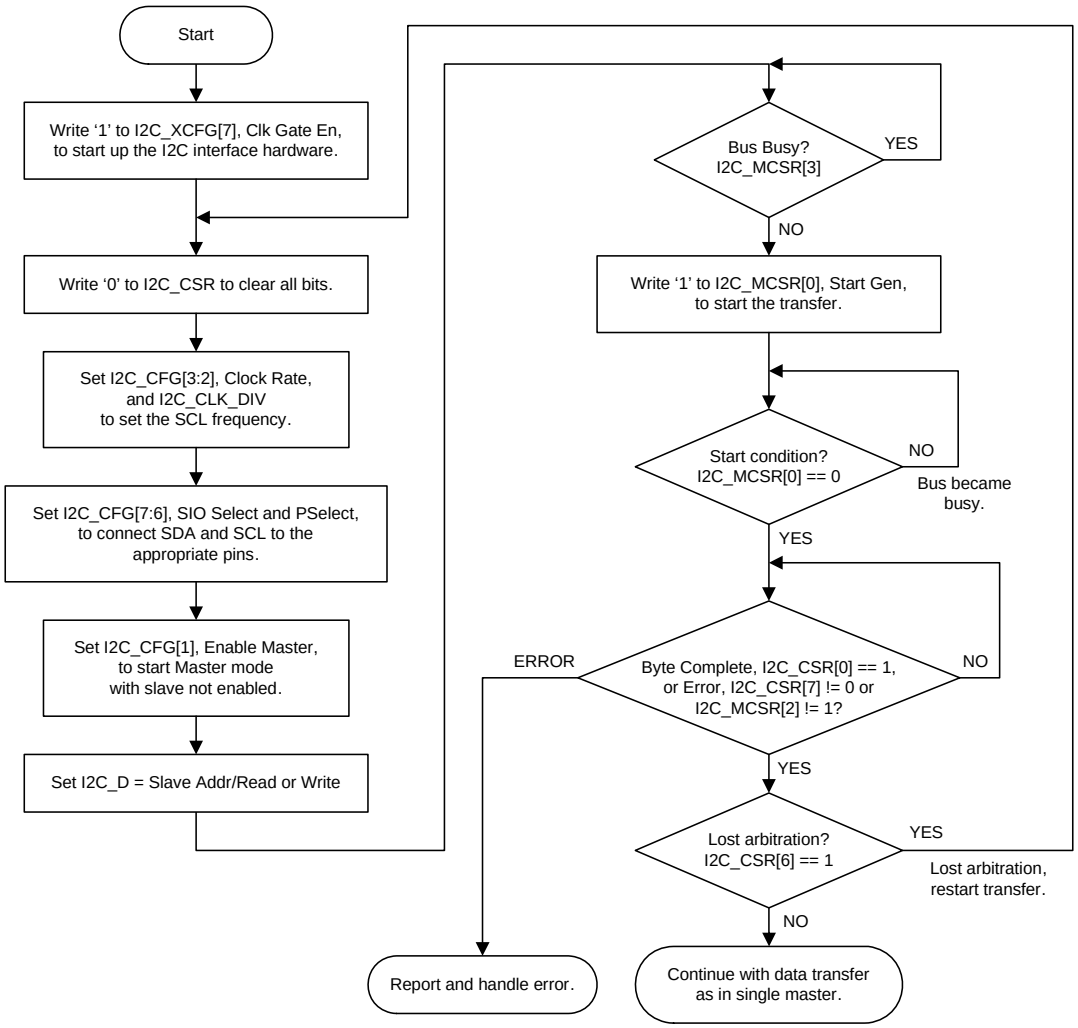
In multi-master mode, data transfer can be achieved with the slave mode not enabled or with the slave mode enabled.

27.7.1 Multi-Master, Slave Not Enabled

A master data transfer operation in a multi-master system, where the slave mode is not enabled is accomplished as shown in Figure 27-9.

Figure 27-9. Multi-Master Mode, Slave Not Enabled Sequence

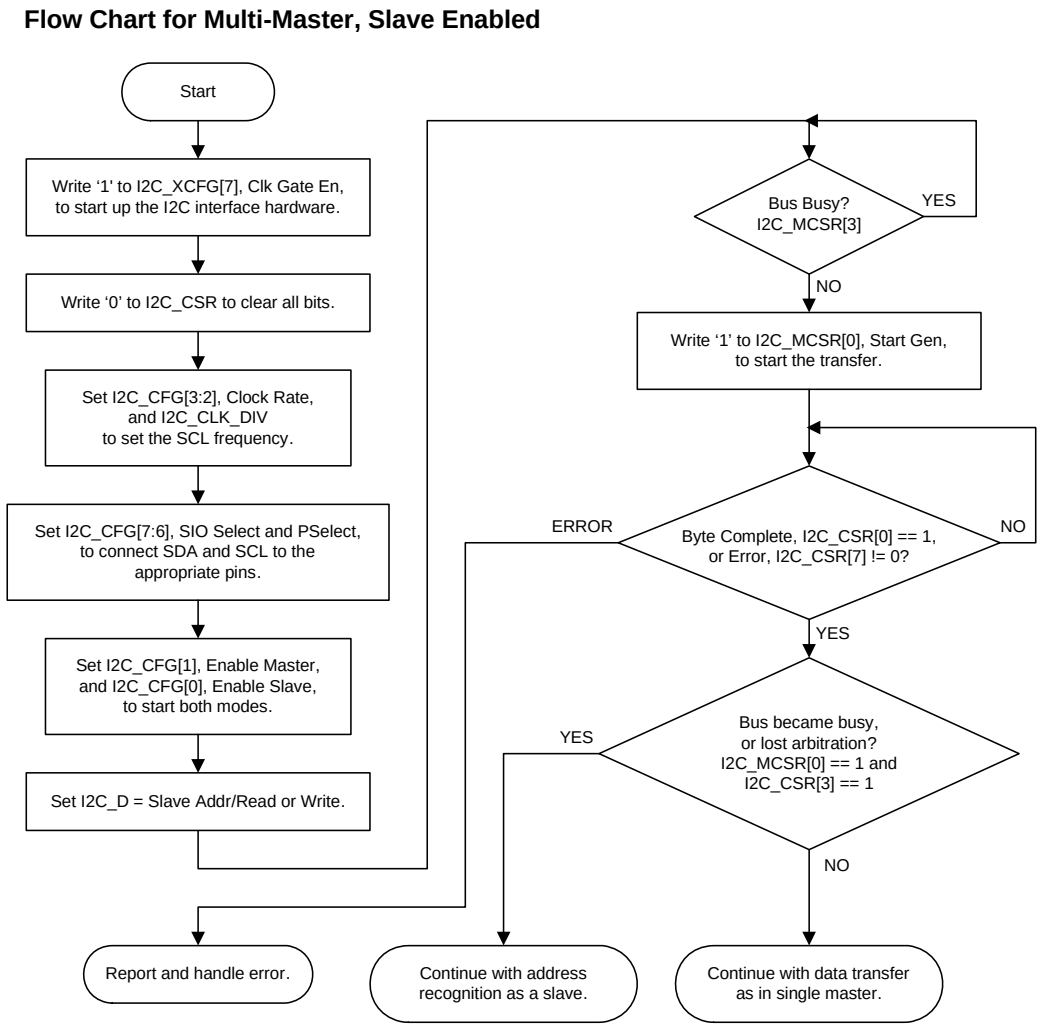
Flow Chart for Multi-Master, Slave Not Enabled



27.7.2 Multi-Master, Slave Enabled

A master data transfer operation in a multi-master system, where the slave mode is enabled is accomplished as shown in Figure 27-10.

Figure 27-10. Multi-Master Mode, Slave Enabled Sequence



28. Digital Filter Block (DFB)

Some PSoC® devices have a dedicated hardware Digital Filter Block (DFB) used to filter applications. The heart of DFB is a multiply and accumulate unit (MAC), which can do 24 bit * 24 bit multiply and 48 bit accumulate in one system clock cycle. In addition, there are data RAMs to store data and coefficients of digital filters.

28.1 Features

- Two 24-bit wide streaming data channels
- Two sets of data RAMs each that can store 128 words of 24-bit width each
- One interrupt and two DMA request channels
- Three Semaphore bits to interact with system software
- Data alignment and coherency protection support options for input and output samples

28.2 Block Diagram

The Digital Filter Block (DFB) is a 24-bit fixed point, programmable limited scope DSP engine. The DFB is made up of four primary subfunctions as shown in the DFB Basic Block diagram in Figure 28-1.

- Controller
- Datapath
- Address Calculation Units (ACUs)
- Bus Interface

The Controller consists of a small amount of digital logic and memories. The memories in the controller are filled with assembled code that make up the data transform function the DFB is intended to perform.

The Datapath subblock is a 24-bit fixed point, numerical processor containing a Multiply and Accumulator (MAC), a multifunction Arithmetic Logic Unit (ALU), sample and coefficient and data RAM (data RAM is shown in Figure 28-1) as well as data routing, shifting, holding, and rounding functions. The datapath block is the calculation unit inside the DFB.

The addressing of the two data RAMs in the datapath block are controlled by the Address Calculation Units (ACUs). There are two (identical) ACUs, one for each RAM.

These three subfunctions make up the core of the DFB block and are wrapped with a 32-bit DMA-capable AHB-Lite Bus Interface with Control/Status registers. Each of these four subfunctions are discussed in the following sections.

Figure 28-1. Digital Filter Block Diagram

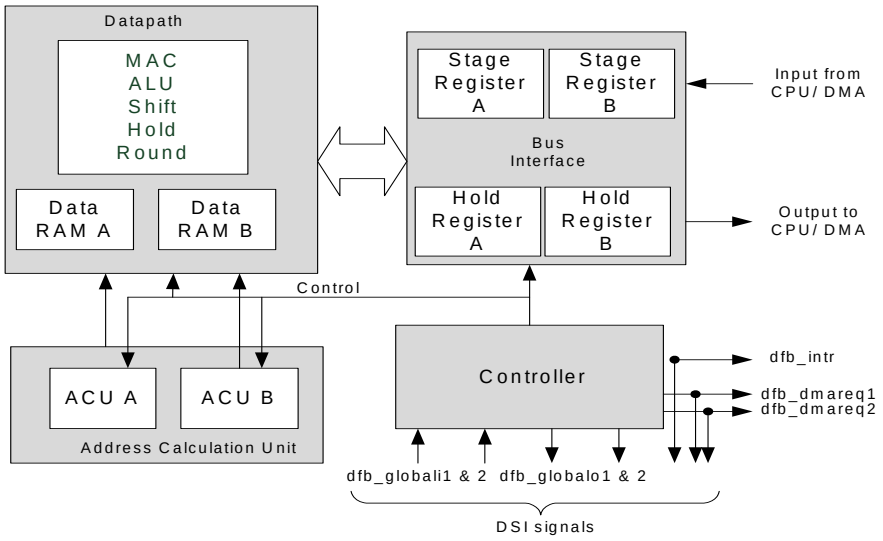


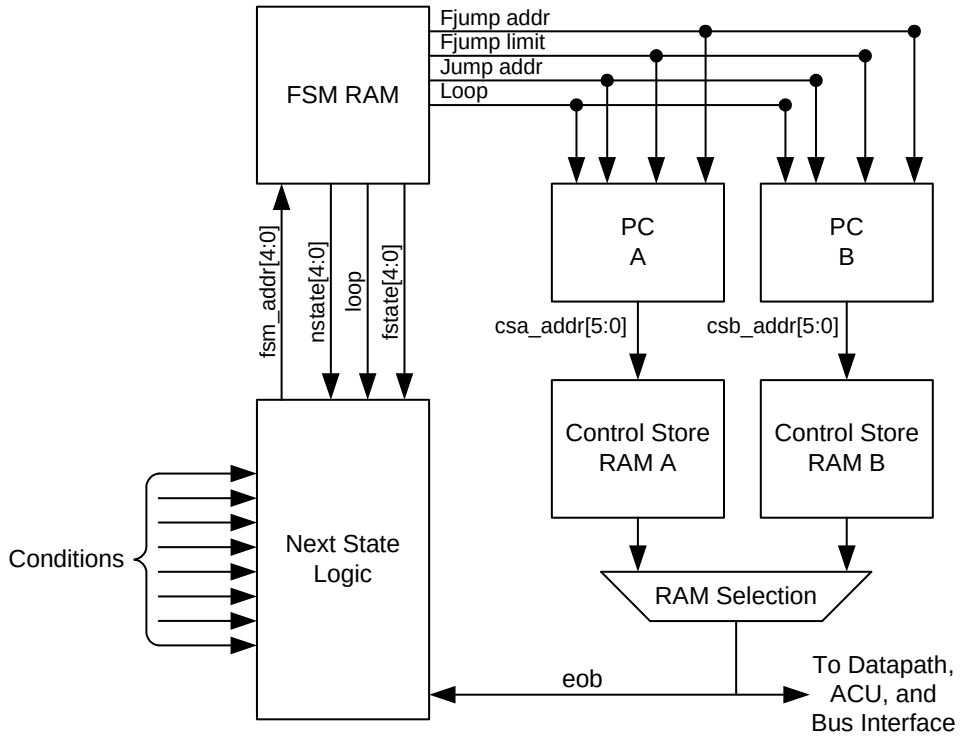
Figure 30-1. Digital Filter Block Diagram

28.3 How It Works

28.3.1 Controller

The controller consists of a RAM-based state machine, a RAM-based control store, program counters, and next state control logic (see Figure 28-2 on page 243). Its function is to control the address calculation units and the datapath, and to communicate with the bus interface to move data in and out of the datapath.

Figure 28-2. Controller Block Diagram



The contents of FSM RAM, the two control store RAMs, the ACU RAM, and potentially the two datapath RAM (if initial conditions are required) must be loaded by the system before use. The contents of the DFB RAMs are stored in flash memory from where they are written into the RAM before the DFB operation is enabled.

The next state decode logic and the FSM RAM comprise the main DFB branch control. The next state decoder generates the FSM RAM's address; the RAM produces next state information and branch flag masks. These masks enable the use of flags as jump conditions for conditional branching. This state machine controls the program counter to produce the address for the Control Store RAMs.

There are two identical Control Store (CS) RAMs and an associated Program Counter to allow an interleaving methodology for CS opcode fetches. The CS RAMs are 64x32 each.

Both CS RAMs are sometimes filled with identical data. It is possible to effectively double the control store instruction space by using different contents in each RAM. It is during branch conditions that next state address calculations happen. Hence, the two possible branch addresses are supplied – one to each RAM. When the branch condition is determined, late in the cycle, the controller simply picks the correct CS RAM output. Opcode execution then switches to and stays with the CS RAM until the next jump condition.

28.3.1.1 FSM RAM

FSM RAM is 64x32 RAM. It is used as ROM. The FSM RAM is filled with control flow information implementing the desired function of the DFB prior to use. This RAM is loaded typically at system boot time, but is not restricted to any particular time as long as the DFB is not running (run is deasserted in DFB_CR[0]). The code in this and the Control Store RAMs can be altered at anytime to change the function performed by the DFB. In fact, some applications have the algorithm loaded routinely and swapped out when several channels of data need processing or when one channel needs multiple transforms – when the code is too large to fit in the available space.

The FSM RAM is addressed as two banks of 32x32. The Bank selection is achieved using the CSR bit (DFB_CR[1]). The primary use of the two banks is to allow two separate code stores to load and jump between without incurring the reload penalty of the FSM RAM.

Table 28-1 shows the bit fields used for the controller by the 32-bit FSM RAM.

Table 28-1. FSM RAM Bit Field Mapping

Name	Enables	Loop	Jump Address	False Jump Limit	False Jump Address	Next State
Signal	enables	loop	jaddr	fjlim	fjaddr	nstate
Bits	31:24	23	22:17	16:11	10:5	4:0
Description	Enables for the top 8 input branching conditions	Signifies a code loop	Jump address for CS RAMs on TRUE	Address loop limit	Jump address for CS RAMs on FALSE*	Next state address for FSM
	* This false jump address is for use only in a loop state, where the controller moves back to the start of the loop on a false condition. If the state is not a loop state, then this address is used for the next state on false value.					

28.3.1.2 Program Counter

The primary purpose of the program counter (PC) is to supply correct addresses to the Control Store (CS) RAMs. This is not as simple as providing a direct address from the FSM RAM because jump addresses must be determined and held in the PC before branches are taken. The PC also controls the incrementing and wrapping of addresses for loops, allowing the FSM to sit in one state during looping processes. For this reason the FSM RAM sends out the jump address and loop conditions to the PC

28.3.1.3 Control Store

The term Control Store (CS) refers to a bank of two interleaved RAMs used to hold control opcodes for the ACUs and the Datapath unit. These RAMs are addressed by the FSM RAM indirectly through the Program Counters and set the per-cycle operation state of the DP and ACUs.

The outputs of these two 32-bit wide RAMs are muxed to one control bus (based upon which is presently the active RAM denoted by DFB_SR[0]) and provide the following bit-fields to the ACUs and Datapath unit listed in Table 28-2.

Table 28-2. Control Store RAM Bit Field Mapping

Name	DP CTRL	Bus WR	ACU-A Opcode	ACU-B Opcode	ACU Addr	End of Block
Signal	dp_ctrl	buswr	acua_op	acub_op	acu_addr	eob
Bits	31:14	13	12:9	8:5	4:1	0
Description	Control bus to the Datapath Unit	Signifies a data output condition to the bus	ACU A's opcode	ACU B's opcode	ACU RAM's address	End of Block marker

28.3.1.4 Next State Decoder

The Next State Decoder is combination logic that controls the state transitions in the FSM RAM. The next state decoder is the logic that gives the address (state address) to the FSM. The result of the next state decoder is governed by the branching signal conditions. You get a state transition when one of these two conditions exist:

- EOB is high and the signal condition goes high. This is the jump on true branch.
- Loop (cfsmram[23]) is low meaning no loop, EOB is high, and condition is low. This is the flow through condition for a false condition.

The branching conditions are:

1. End of block is encountered for a control store block – a condition for a jump because a jump instruction signifies the end of the block.
2. Datapath status inputs such as sign, threshold, and equal.
   - Dpsign – A jump based on the MSB of the ALU output. If ALU output goes negative, assert.
   - Dpthresh – Datapath Threshold – Asserted when the ALU detects a sign change, such as a zero crossing detection.
   - Dpeq – Datapath Equity – Asserted when the ALU hardware detects an output value of zero.
3. Acueq – ACU A or B REG is equal to MREG or LREG, if modflag is set. ACU A or B REG is equal to 127 or 0, if modflag is cleared. This means that the pointer to the DP Data registers has reached its upper/lower limit. Refer to 28.3.2 Datapath on page 245 for clarity.

4. IN1 or IN2 – When new data is available in one of the staging registers A or B. Signals a new input cycle and is available for consumption. Remains asserted until cleared by a bus read command.
5. globali1 – Branch control input from DSI port.
6. globali2 – Branch control input from DSI port.
7. The sat_det flag (Saturation) from ALU – This flag is set when saturation occurs in MAC, ALU, or Shifter.
8. Any of the semaphores (refer to the PHUB and DMAC chapter on page 51).

For branching, the branching conditions must be enabled. The ENGLOBALS, ENSATRND, ENSEM, SETSEM, and CLEARSEM commands are used.

If the ALU command is ENSEM, then the data on acu_addr[2:0] is written to the register sem_en for enabling semaphores to be branching conditions. The acu_addr[2:0] is converted bitwise to enable each of the three semaphores.

The SETSEM and CLEARSEM are used to set or clear the semaphores based on the semaphore selected in acu_addr[2:0].

Acu_addr[2] -> semaphore2
Acu_addr[1] -> semaphore1
Acu_addr[0] -> semaphore0

The ENGLOBALS command is used to enable the use of external dsi inputs and datapath saturation flags as branching conditions. ENGLOBALS shares an ALU opcode with ENSATRND. They are differentiated by the acu_addr[3] bit as shown in Table 28-3.

Table 28-3. ENGLOBALS and ENSATRND Commands

Englobals	Acu_addr[3]=0	Acu_addr[0]: enables globali1 Acu_addr[1]: enables globali2 Acu_addr[2]: enables sat_det
Ensatrnd	Acu_addr[3]=1	Acu_addr[0]: writes to rnd_flag Acu_addr[1]: writes to sat_flag Acu_addr[2]: creates strobe to clear saturation flag

28.3.2 Datapath

Datapath (DP) is the name used to refer to the numerical calculation unit of the DFB. The datapath subblock is a 24-bit fixed-point numerical processor containing a 48-bit MAC, a multi-function ALU, sample and coefficient data RAMs as well as data routing, shifting, holding and rounding functions.

The DP architecture makes use of two 128x24 single-port RAMs (RAM A and RAM B). The RAMs can be loaded from the bus or from the datapath output (feedback). These RAMs hold data and coefficients with size and location under full DFB controller control.

The heart of the DP unit is a 48-bit Multiply and Accumulator (MAC). Two 24-bit values can be multiplied and the result added to the 48-bit accumulator in each clock cycle. This accumulator or any memory value can be routed to the ALU. Results from the ALU can then be stored in either Data RAM. The MAC is the only portion of the DP that is wider than 24 bits. All results from the MAC are passed on to the ALU as 24-bit values representing the high-order 24 bits in the accumulator shifted by one (bits 46:23). The MAC assumes an implied binary point after the MSB which shifts the result down a bit in the output of the MAC. For this reason, bits 46:23 are used instead of 47:24.

The DP unit also contains an optimized ALU that supports add, subtract, comparison, threshold, absolute value, squelch, saturation, and other functions.

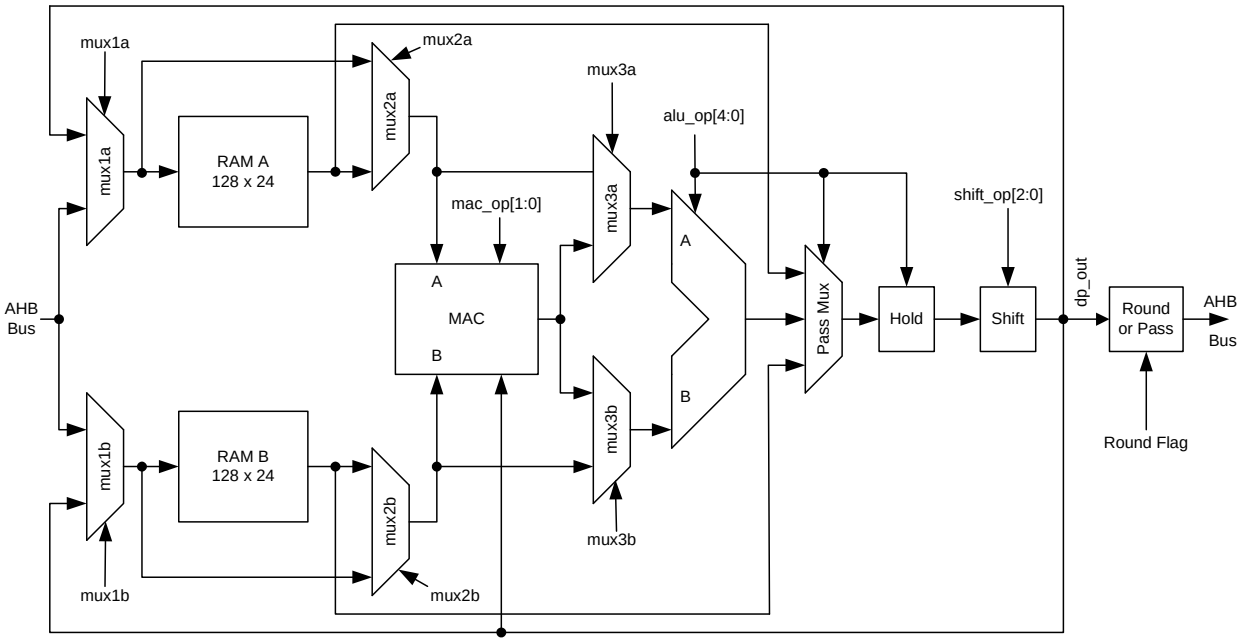
With the exception of the DP RAM addresses, the DP unit is completely controlled by seven control fields totaling 18 bits coming from the DFB Controller as the DP_CTRL control bus (Table 28-2 on page 244). These 18 bits of control are listed in Table 28-4.

Table 28-4. Datapath Opcode Bit Field Mapping

Name	B Mux Ctrl	A Mux Ctrl	MAC Opcode	ALU Opcode	Shift Opcode	RAM A WR	RAM B WR
Signal	muxb*_ctrl	muxa*_ctrl	mac_op	alu_op	shift	dpa_r_wb	dpa_r_wb
Width	3	3	2	5	3	1	1
Description	mux1b mux2b mux3b	mux1a mux2a mux3a	MAC opcode	ALU opcode	DP output shifter opcode	Write signal to RAM A	Write signal to RAM B

Note how the different signals from Table 28-4 affect the functioning of the different elements in the datapath.

Figure 28-3. Datapath



Round Mode – If DP is in Round mode, any result passing out of the DP unit is being rounded to a 16-bit value. This feature status is shown in the register setting, DFB_SR [2].

Saturation Mode – If DP is in Saturation mode, any mathematical operation that produces a number outside the range of a 24-bit 2's complement number is clamped to the maximum positive or negative number. Enabling and disabling saturation and rounding is under the control of DFB controller. See the ALU instruction set. The status is visible at DFB_SR [1].

28.3.2.1 MAC

The multiply add function takes two 24-bit signed numbers and calculates a 48-bit signed result, then adds a signed 48-bit value ((a*b)+c).

The accumulator consists of a 48-bit register and the multiply adder.

Together these two functions, along with some control logic, make up the MAC. Based on the opcode (mac_op) coming from the DFB controller it can do one of the following operation:

- Multiply and accumulate with previous Values
- Clear Accumulator and load with current product.
- Hold accumulator, no multiply (no power in mult)
- Add ALU value to product and start new accumulation

The output of MAC is higher order 24 bits of multiply accumulate operation. The MAC assumes an implied binary point after the MSB, which shifts the result down a bit in the output of the MAC. For this reason, bits 46:23 are used instead of 47:24. The instruction set for the MAC, ALU and Shifter is listed in Table 28-7 on page 253, Table 28-6 on page 253, and Table 28-8 on page 254.

28.3.2.2 ALU

The ALU provides data control on the output end of the data path. ALU supports add, subtract comparison, threshold, absolute value, squelch, saturation, and other functions. See Table 28-6 for various instructions supported by ALU.

The ALU commands and inputs are pipelined. This pipelining can be made use for data movement in some filtering applications. This pipelining causes a delay of two clock cycles for the ALU input to reach the output.

28.3.2.3 Shifter and Rounder

The shifter at the ALU output can be used to shift the ALU results as required. See Table 28-8 for various shifter commands. Rounder rounds the results to a 16 bit value when the data path is operating in round mode.

28.3.3 Address Calculation Unit

The Address Calculation Units (ACUs) generate addresses for each DP RAM. There are two address calculation unit for supporting sophisticated branching operations.

The ACU is capable of saving and restoring address, incrementing or decrementing address by 1 or n (n is a constant value stored in FREG), flagging a programmable terminal count, and a number of other functions.

REG – Stores the current value that the ACU is operating on and outputs it on every cycle, unless a command specifies otherwise.

FREG – Loads with a value to increment or decrement by, when using the ADDF and SUBF commands. For example, load two into FREG and then it is possible to increment through the data RAMs by two.

MREG – Stores the maximum value before a wraparound if modulo arithmetic is turned on. When the address calculated by the ACU exceeds MREG value, it will wraparound to LREG value, if modulo arithmetic is turned on.

LREG – Stores the minimum value before a wraparound to the MREG value when modulus arithmetic is turned on.

Modulus arithmetic is enabled using the SETMODE ACU command and disabled using the UNSETMOD command. Modulus arithmetic prevents the ACU from incrementing past the value of MREG and from decrementing below the value of LREG. Make sure the REG value is within the LREG:MREG range at the time modulus arithmetic is turned on to avoid unexpected results.

The ACU (including the ACU RAM) is initialized whenever a hard reset event occurs or when the RUN bit in the DEC_CR register is ‘0’. Initialization is as follows:

ACU RAM Contents=0, MREG=127, LREG=0, FREG=2.

The current address and state of the register of both ACUs can be stored or retrieved from memory with assembly instructions. This is used in context switching. A 16x14 ACU RAM is used for this purpose. The 16x14 RAM is used by both ACUs. The upper seven bits are for ACU B and the lower seven bits are for ACU A. Thus, each ACU can store 16 addresses or state elements.

The ACU instructions perform incrementing/decrementing of the data RAM addresses by one or the value in FREG. Apart from this, the modulus arithmetic is used to enable a wrap around at user defined limits.

Note Apart from addressing the ACU RAM, the ACU_addr is also used as an argument for other ALU and branching commands. The single ACU_addr value can be used simultaneously for different commands (ACU, ALU...) if coincidence requires the same value on the ACU_addr for all commands involved.

28.3.4 Bus Interface and Register Descriptions

The DFB block is wrapped with a 32-bit AHB-Lite Slave bus interface. A 32-bit bus was chosen to accommodate the fact that the RAMs in the DFB are all 24 bits and most of the bus transfers to the DFB are 24 bits.

The DFB has a set of expanded Control and Status Registers (CSR) that are accessible through the system bus at all times. The registers containing CSR bit information are address mapped as 32-bit registers with active bits only in the low byte. This arrangement works well for both 8-bit and 32-bit MCUs.

The CSRs that hold sample data are 24-bits wide (Staging and Hold register) and coherency interlocking HW is included to allow 8-bit and 16-bit accesses.

In normal mode of operation, the DFB RAMs (except the input staging and output holding registers) is controlled by the DFB controller and is not accessible to CPU/DMA. If CPU/DMA needs control of this DFB RAM memory it should make use of DFB_RAM_DIR control bits (one per RAM) to give the RAM control to system bus.

28.3.4.1 Streaming Mode

In streaming mode the filter coefficients and historic data are loaded into DFB before starting the DFB operations. Runtime data movement is through the staging and holding registers.

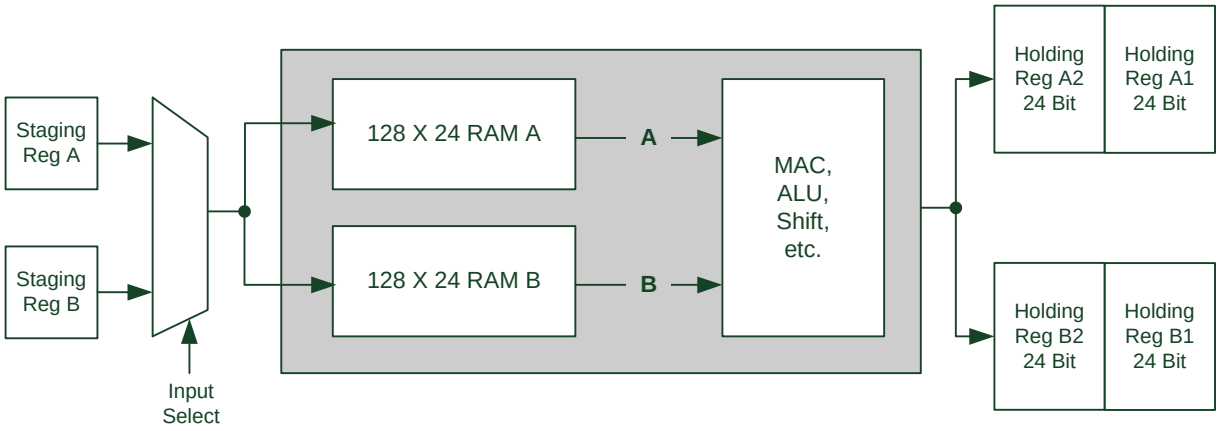
The DFB has:

- Two 24-bit input staging registers
- Two 24-bit output holding registers

These registers can be accessed by both DFB and AHB Bus (CPU/DMA). In reality, these registers are double buffered, but to the DFB controller and the system bus, they appear as single registers. In streaming mode data to be filtered is streamed in to staging registers. Filter output is streamed out through DFB holding registers.

The two sets of input and output registers aid stereo data processing applications. Applications requiring more than two concurrent channels must use block mode.

Figure 28-4. Streaming Mode Transfer



In input Streaming mode, the sample rate is determined by the ADC or other sampling resources providing the input samples. By definition the DFB must be running (processing) samples faster than or at the exact same rate as the sample source to function properly. Therefore, the DFB knows how to stall and wait for subsequent input data or postpone operation on that channel and switch to another channel (if in use).

When the calculation engine is finished processing a sample, a bus read instruction can be issued. At this point, the next staged sample is read or, if not present yet, the DFB controller stalls while waiting for the next input sample. If two streaming channels are being processed, the DFB controller, upon completion of a calculation, can jump to the other channel.

The full or empty status of the two Staging registers is visible to the DFB controller and it can branch based on the status information, allowing it control of which channel it is working on.

When the bus read instruction is issued by the DFB controller, it does not request the bus, generate an interrupt, or DMA request. It simply tells the DFB bus interface that it wants the next sample and will wait until it arrives. In this state, the DFB controller waits until the bus interface signals that the sample has arrived. A one 24-bit word Staging register is used for a sample rate at or below 1 Msps and guaranteed bus latency lower than the sample period. There are two Staging registers: one for each supported channel.

In streaming mode new samples arrive in the staging registers. The DFB controller checks for new data write to staging registers and branch to process data depending on the CFSM code.

The input staging registers are read by the DFB controller by asserting the bus read signal and addressing the two registers with the low order ACU RAM address bit (acu_addr[0]). If the address bit is low, Staging register A is read; if the address bit is high, B is read. When read, the associated Stage Valid signal is automatically cleared by the hardware.

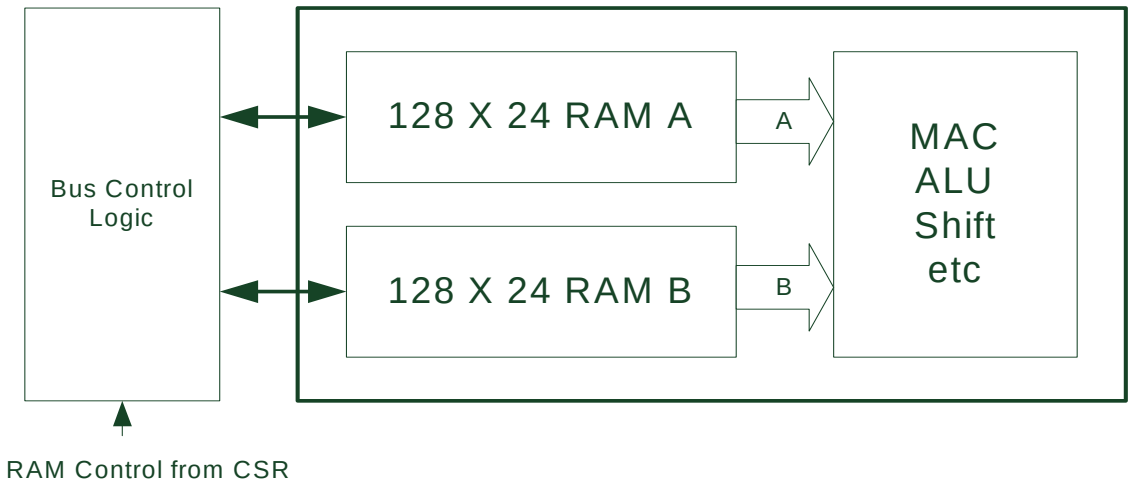
Apart from this, the Staging register also has a key coherence byte setting. This setting is available to reduce errors due to bus access being less wide as compared to the register width. The staging registers are protected on writes, so the underlying hardware does not incorrectly use the field when it is partially updated by the system software. If the system software is in the middle of reading from the holding registers, the DFB will not update the holding registers until the coherency key byte is read. The Key Coherency byte is basically the user (software) telling the hardware which byte of the field is written or read last when an update to the field is required. In the Staging register the new value availability is flagged only when the key coherency byte is written to.

28.3.4.2 Block Transfer Modes

Block mode is defined by the system software moving sets of samples or coefficient data in and out of the DFB data RAMs in blocks. This method of using the DFB supports such features as multi-channel processing and deeper filters than the embedded data RAMs will support. It can also be used to initialize the DFB RAMs for streaming mode operation.

The DFB datapath block has two 128x24 embedded data RAMs. These hold the data (signal or coefficients) used in the calculation of numerical processes. These two RAMs are completely separate memories from the bus' point of view. The DFB views these two RAMs as a working set, as shown in Figure 28-5 on page 249.

Figure 28-5. Block Mode Transfer



The primary concept of Block mode is to allow the system software full control of what is in the data RAM for each calculation cycle of the DFB. In general, this extends the functionality of the DFB by trading performance for fundamental features such as the ability to implement filters with more taps than 128 or to time division multiplex the processing of more than two low sample rate channels. The system software burden of Block mode is in the management of the RAM's contents. Both system and DFB performance is lost due to software servicing of the DFB and because the DFB must stall while the system software reads/writes the data RAMs. Block mode also creates more bus traffic on the system bus for a given sample rate.

The system software takes control of memory by putting it on the system bus with the use of (DFB_RAM_DIR) control bits (one per RAM). It then reads/writes the data and "passes" the memory back to the DFB by toggling the control bit back. While this is happening, the DFB must stall, unless it is performing some function that only requires one of the two data RAMs. The two data RAMs are individually controlled by the system software as to which resource has control of them – the bus or the DFB.

Any number of data channels can be supported with Block mode (within reason). With each added data channel, the system software has the additional burden of tracking and managing and sample rates supported reduces considerably because the DFB must be stalled for data movement operations.

The DFB controller provides a semaphore methodology to communicate with the system software as to the status of the data RAMs when being passed back and forth for block transfers. Optional interrupt support can be associated with the setting and clearing of semaphores.

Typically, results of DFB applications are streaming in nature. However, in cases where results are created as data sets, Block mode can be used to move the resultant data sets out of the DFB data RAMs.

28.3.4.3 Result Handling

Frequently DFB block output results are generated at periodic intervals after a series of mathematical calculations. This also happens after a wait for the input sample stream. The generation rate of these result elements will vary radically based on the function being programmed and run on the DFB.

To assist system software with the handling of resultant data, the DFB implements two Holding registers, 24 bits wide, for output results. In reality, these two Holding registers are double buffered, but to the DFB controller and the system bus, they appear as a single register. They are referred to as a single register hereafter, but keep in mind there are really two registers to deal with bus latency issues. The fact they are double buffered is transparent to both the bus and the DFB controller. Hardware automatically manages the fact that they are double buffered.

The intent of having two fully addressed Holding registers is primarily to allow the controller and system software to map filter channels so that DMA requests are much easier to support. The two Holding registers are addressed with the low-order ACU address bit out of the Control Store.

When bus write is asserted in the CS word and the low-order ACU address bit (acu_addr[0]) is low, Holding register A is written and Holding register B is written when the low-order address bit is high.

There are a couple of methods provided to read the Holding registers on the system bus. These registers are generic

read only CSRs. They can be read manually by software running on the MCU under poled or interrupt control (DFB_INTR_CTRL), or each can be associated with a DMA request signal and read by the system DMA controller (DFB_DMA_CTRL). Pending interrupts from the Holding register update is monitored from the DFB_SR register.

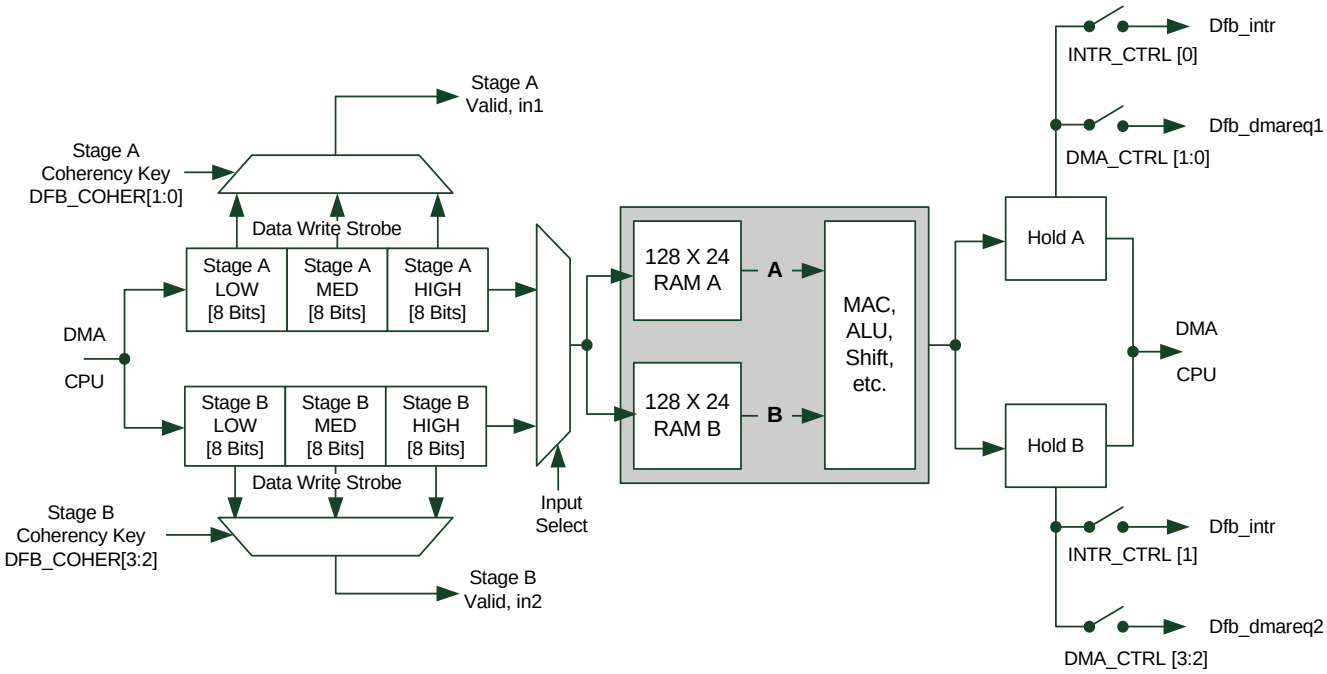
Operations on the Holding registers are protected. The nature of the protection is set by the coherence bits (DFB_COHER). The Holding registers are protected on reads so that the underlying HW doesn't update it when partially read by the System SW or DMA. The key coherency byte is selected in the Coherency register. The Key Coherency byte is basically the user (software) telling the hardware which byte of the field is read last. The Holding registers are considered read after the key coherency byte is read.

Note 1 In Block mode, when more than two channels are being processed, management of the output results is more burdensome to the system software as it can no longer be constantly mapped one-to-one with a Holding register or DMA request.

Note 2 In 8-bit devices, reading the Holding registers manually results in a multi-cycle operation.

Figure 28-6 explains DFB control signals can be used for data streaming and result handling.

Figure 28-6. Control Signals for Data Streaming and result handling



28.3.4.4 Data Alignment

The hardware provides a data alignment feature in the input Staging registers and in the output Holding registers for system software convenience.

Both staging and holding registers support byte accesses that addresses alignment issues for input and output samples of 8 bits or less. Also, all four of these registers are mapped as 32-bit registers (only three of the four bytes are used) so there are no alignment issues for samples between 17 and 24 bits. However, for sample sizes between 9 and 16, it is convenient to read and write these samples on bus bits 15:0, while they source and sink on bits 23:8 of the Holding and Staging registers.

The CSR DALIGN provides bits that enable an alignment feature which allows bus bits 15:0 to either be sourced from Holding register bits 23:8 or sink to Staging register bits 23:8. Each Staging and Holding register can be configured individually with a bit in the DALIGN register. If the bit is set high, the effective byte shift occurs. For example, if an output sample from the Decimator is 12 bits wide, aligned to bit 23 of the Decimator Output Sample register, and is desired to stream this value to the DFB, the similar data alignment feature of the Decimator can be enabled, allowing the 16 bits of the Decimator Output Sample register to be read on bus bits 15:0. Setting the alignment feature in the DFB for the Staging A input register, these 16 bits can be written on bus bits 15:0 and will be written into bits 23:8 of the Staging A register when required.

28.3.4.5 DMA and Semaphores

The DFB bus interface supports two DMA request signals. These can be associated with the two Holding registers (optional) or associated with the semaphore bits (see register DFB_DMA_CTRL).

The DFB provides three generic semaphore register bits that the system software and the DFB controller can use to communicate. The intent of these three semaphores is to allow the system software and the DFB controller to communicate the status of data movement in and out of the DFB and, in particular, the handling of block data transfers. The definition of these three bits is left to the system and controller software architects.

To set and clear semaphores bits, two DP ALU commands are available: SEM_SET and SEM_CLR. For each active high bit of the ACU address, the corresponding semaphore bit is either set or cleared.

For system software to write into a semaphore bit the register DFB_SEMA is used. The mask bit is set when the corresponding semaphore bit in the register is updated.

Any of the semaphore bits can be optionally (programmable) associated with the system interrupt signal (DFB_INTR_CTRL) or either of the DMAREQ (DFB_DMA_CTRL) outputs leaving the DFB, and/or either of the outgoing Global signal. Pending semaphore interrupts are monitored from the DFB_SR register.

28.3.4.6 DSI Routed Inputs and Outputs

The DFB has the option to take two DSI global inputs (globali1 and globali2) and two DSI global outputs (globalo1 and globalo2).

Use of the global outputs is optional. If needed, they can be programmed to carry one of four different DFB internal status/control signals. These can be routed to the DSI and used as inputs to other circuits. The global outputs can be configured to carry semaphore, an interrupt, or DP status signals as listed in Table 28-5 on page 252. This is done using the DFB_DSI_CTRL register.

The DSI inputs into the DFB to control operations of the FSM are optionally used as branching inputs to the Controller's next state decoder. See the section on 28.3.1.4 Next State Decoder on page 244 for more details.

28.4 DFB Instruction Set

Each control word for the DFB is 32 bits long. The fields in the control word are as follows:

- Datapath Mux Control – 6 bits
- Data RAM R/W – 2 bits
- Bus R/W – 1 bit
- ALU Control – 5 bits
- MAC Control – 2 bits
- Shifter Control – 3 bits
- ACU Control – 8 bits
- ACURAM Address – 4 bits
- End of Code Block – 1 bit

The mux control bits are split equally between the A and B paths each having 3 bits. Three bits are allocated and encode the control of the mux1, mux2, and mux3 functions as shown in Table 28-5.

Table 28-5. Mux Functions

Code	Assembly Name	Function MUX1	Function MUX2	Function MUX3	Function
0	BA	mux1 = AHB Bus	mux2 = mux1	mux3 = mux2	AHB ->ALU
1	SA	mux1 = dp_out	mux2 = mux1	mux3 = mux2	dp_out->ALU
2	BRA	mux1 = AHB Bus	mux2 = RAM out	mux3 = mux2	AHB->RAM AHB->ALU
3	SRA	mux1 = dp_out	mux2 = RAM out	mux3 = mux2	dp_out->RAM dp_out ->ALU
4	BM	mux1 = AHB Bus	mux2 = mux1	mux3 = MAC	AHB->MAC->ALU
5	SM	mux1 = dp_out	mux2 = mux1	mux3 = MAC	dp_out->MAC->ALU
6	BRM	mux1 = AHB Bus	mux2 = RAM out	mux3 = MAC	AHB->MAC->ALU AHB->RAM
7	SRM	mux1 = dp_out	mux2 = RAM out	mux3 = MAC	dp_out->MAC->ALU dp_out->RAM

ALU functions are programmed as shown in Table 28-6 and are encoded in 5 bits.

Table 28-6. ALU Functions

Code	Assembly Name	Function
0	SET0	Set ALU output to 0
1	SET1	Set ALU output to 1
2	SETA	PASS A to ALU output
3	SETB	PASS B to ALU output
4	NEGA	Set ALU output to –A
5	NEGB	Set ALU output to –B
6	PASSRAMA	Pass RAM A output directly to ALU output
7	PASSRAMB	Pass RAM B output directly to ALU output
8	ADD	Add A and B and put result on the ALU output
9	TDECA	Put A-1 on the ALU output, set threshold detection
10	SUBA	Put B-A on the ALU output
11	SUBB	Put A-B on the ALU output
12	ABSA	Put |A| on the ALU output
13	ABSB	Put |B| on the ALU output
14	ADDABSA	Put |A| + B on the ALU output
15	ADDABSB	Put A + |B| on the ALU output
16	HOLD	Hold ALU output from previous cycle
17	ENGLOBALS, -	Enables global and saturation jump conditions using a 3-bit field to specify which events are active jump conditions
17	ENSATRND, -	Writes to the saturation and rounding enable register using a 3-bit field to enable and disable them
18	ENSEM, ---	Enables semaphores as jump conditions using a 3-bit field to specify which are active
19	SETSEM, ---	Set the semaphores high using the 3-bit mask
20	CLEARSEM, ---	Set the semaphores low using mask, addr[2:0]
21	TSUBA	Put B-A on the ALU output, set threshold detection
22	TSUBB	Put A-B on the ALU output, set threshold detection
23	TADDABSA	Put |A| + B on the ALU output, set threshold detection
24	TADDABSB	Put A + |B| on the ALU output, set threshold detection
25	SQLCMP	Load squelch comparison register with a value from side A, Pass Side B
26	SQLCNT	Load squelch count register with value from side A, Pass Side B
27	SQA	Squelch side A: If value is above threshold pass it. If value is below threshold and the squelch count register is zero, pass. zero. Otherwise pass A
28	SQB	Squelch side B: If value is above threshold pass it. If value is below threshold and the squelch count register is zero, pass. zero. Otherwise pass B
29-31	UNDEFINED	Undefined Opcodes

MAC functions are programmed as shown in Table 28-7 and are encoded in 2 bits.

Table 28-7. MAC Functions

Code	Assembly Name	Function
0	LOADALU	Add ALU value to product and start new accumulation
1	CLRA	Load accumulator with product but a 0 sum
2	HOLD	Hold accumulator, no multiply (no power in mult)
3	MACC	Default – just accumulate

Shifter functions are programmed as shown in Table 28-8 and are encoded in 3 bits. If deeper shifts are required, data can be passed through the ALU on multiple cycles.

Table 28-8. Shifter Functions

Code	Assembly Name	Function
0	<default>	No shift
1	shift(right,1)	Shift right 1 (divide by 2)
2	shift(right,2)	Shift right 2 (divide by 4)
3	shift(right,3)	Shift right 3
4	shift(right,4)	Shift right 4
5	shift(right,8)	Shift right 8
6	shift(left,1)	Shift left 1 (multiply by 2)
7	shift(left,2)	Shift left 2 (multiply by 4)

Two ACUs are supplied. There are 16 functions per ACU as shown in Table 28-9 and are encoded in 4 bits. This RAM is useful when parallel filters or algorithms are implemented and control flow needs to shift from one to the other, while still maintaining the relative addresses for each filter.

Table 28-9. ACU Functions

Code	Assembly Name	Function
0	HOLD	Put REG on output, hold REG in REG
1	INCR	If (modflag && REG = MREG Put LREG on output, write to REG else If (!modflag && REG = 127) Put 0 on output, write to REG else Put REG+1 on the output, write to REG

2	DECR	If (modflag && REG = LREG)
		Put MREG on output, write to REG
		else If (!modflag && REG = 0)
		Put 127 on output, write to REG
		else
		Put REG–1 on the output, write to REG

3	READ	Read ACU RAM and put value on output Write to REG
4	WRITE	Put REG on output, write output to RAM Hold REG in REG
5	LOADF	Load FREG from ACU RAM, put REG on output
6	LOADL	Load LREG from ACU RAM, put REG on output
7	LOADM	Load MREG from ACU RAM, put REG on output
8	WRITEL	Put LREG on output, assert RAM write enable Hold REG in REG
9	SETMOD	Set modflag true, put REG on output Hold REG in REG
10	UNSETMOD	Set modflag false, put REG on output Hold REG in REG

11	CLEAR	If (modflag)
		Put LREG on output, write to REG
		else
		Put 0 on output, write to REG

Table 28-9. ACU Functions *(continued)*

Code	Assembly Name	Function
12	ADDF	If (modflag && REG+FREG>MREG) Put ((((REG+FREG)-MREG)–1)+LREG) on output else If (!modflag && REG+FREG>127) Put (((REG+FREG)–127)–1) on output else Put REG+FREG on output, write to REG

13	SUBF	If (modflag && REG-FREG<LREG)
		Put MREG-((FREG-(REG-LREG))–1) on output
		else If (!modflag && REG-FREG<0)
		Put 127-((FREG-REG)–1) on output
		else
		Put REG-FREG on output, write to REG

14	WRITEM	Put MREG on output, assert RAM write enable Hold REG in REG
15	WRITEF	Put FREG on output, assert RAM write enable Hold REG in REG

28.5 Usage Model

The instruction set is programmed into the controller based on dual control stores and a control finite state machine.

The control store contains sequential blocks of instructions to execute an algorithm. In the simplest programming model, all of the statements will appear in line and the program counter will step from zero to the end of the last instruction.

In this architecture it is more efficient to reuse blocks of code (such as implementing a biquad IIR section). In this specific case, a block of 12 instructions are looped through with offsets in the ACU adjusted so that the correct coefficients and data are used. To control the use of this subroutine, a branching controller is needed. This is the Control Finite State Machine (CFSM) in the controller.

The CFSM contains information about branching. At the end of each clock cycle, the various datapath flags, ACU flags, and globals are evaluated to determine if a jump condition is met. A jump is only allowed at the end of a CS block, which is indicated when the EOB (end of block) bit in the control store word is set to ‘1’. The CFSM RAM stores information about the current state, bits to control which of the input flags are active, and the jump address.

Loop counters are often found in architectures supporting a single instruction MAC for FIR filtering. The loop counter function can be achieved more generally in this architecture through the use of the ACU. The equal flag in the ACU gets set when the address is equal to the mask in the MREG and when the end of the ram is reached or zero. Thus a branch is triggered when the address reaches a certain address. This is how a single instruction, zero instruction overhead branch loop for FIR filtering can be implemented.

Section F: Analog System

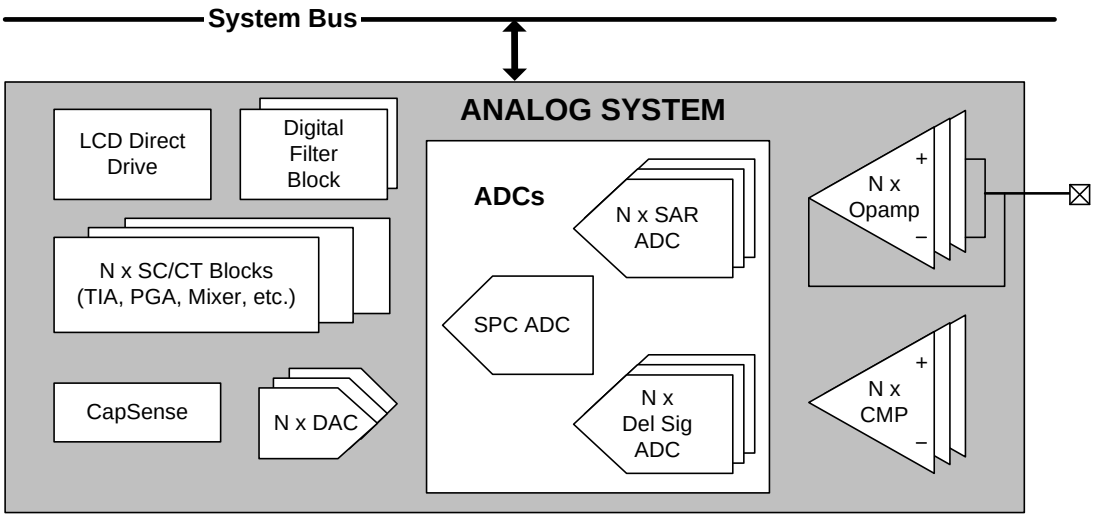
The PSoC® analog subsystem provides the device the second half of its unique configurability. All analog performance is based on a highly accurate absolute voltage reference with less than 0.2% error over temperature and voltage. The configurable analog subsystem includes analog muxes, comparators, mixers, voltage references, analog-to-digital converters (ADC), digital-to-analog converters (DAC), and digital filter bocks (DFB). All GPIO pins can route analog signals into and out of the device, using the internal analog bus. This feature allows the device to interface up to 62 discrete analog signals.

This section encompasses the following chapters:

- Switched Capacitor/Continuous Time chapter on page 259
- Analog Routing chapter on page 273
- Comparators chapter on page 289
- Opamp chapter on page 293
- LCD Direct Drive chapter on page 297
- CapSense® chapter on page 305
- Digital-to-Analog Converter chapter on page 311
- Precision Reference chapter on page 315
- Delta Sigma Converter chapter on page 319
- Successive Approximation Register ADC chapter on page 339

Top Level Architecture

Analog System Block Diagram



30. Switched Capacitor/Continuous Time

The PSoC® 5 Switched Capacitor (SC) – Continuous Time (CT) block is a general purpose block constructed of a rail-to-rail amplifier with arrays of switches, capacitors, and resistors. Register configurations select the block functional topology, power level, and bandwidth.

30.1 Features

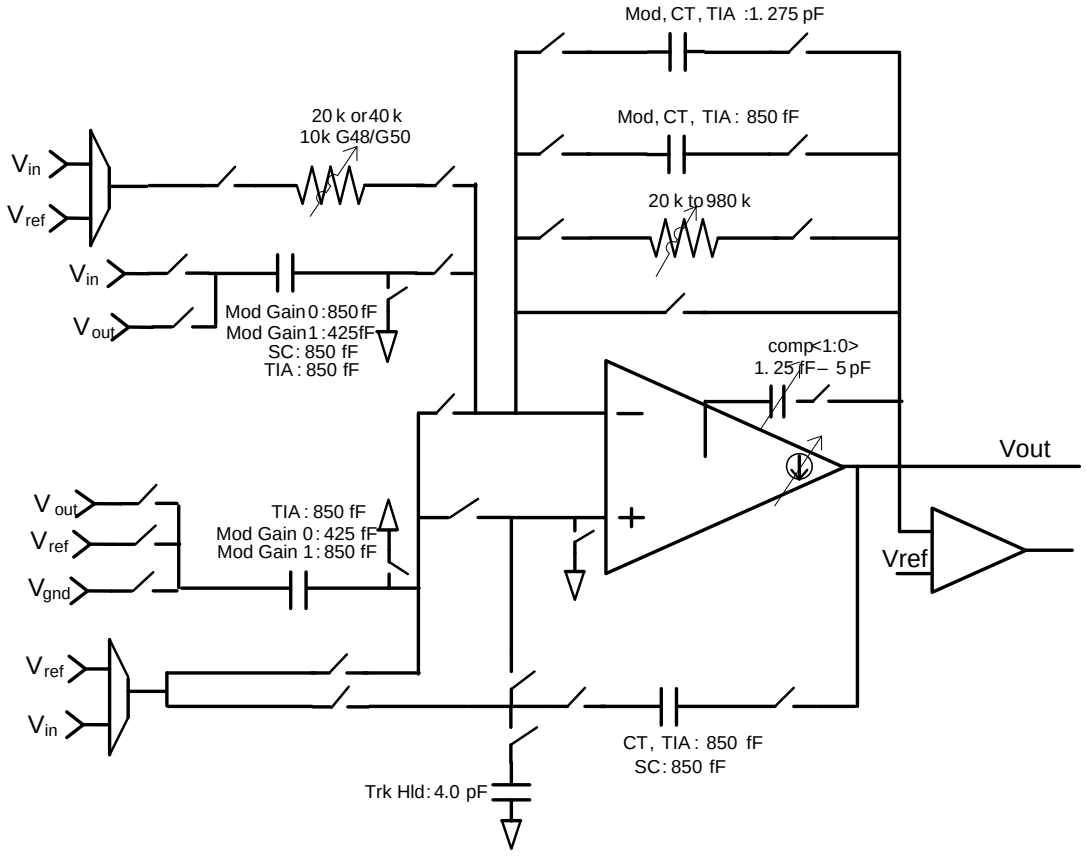
The PSoC SC/CT block has these features:

- Multiple configurations:
  - Naked Opamp
  - Continuous Time Unity Gain Buffer
  - Track and Hold Amplifier
  - Continuous Time Programmable Gain Amplifier
  - Continuous Time Trans Impedance Amplifier
  - Continuous Time Mixer
  - Sampled Mixer (non return-to-zero sample and hold -- NRZ S/H)
  - Delta Sigma Modulator
- Routability to GPIO
- Routable reference selection
- Programmable power and bandwidth
- Sample and hold configuration

30.2 Block Diagram

The overall block diagram of the block is shown in Figure 30-1 on page 260. Individual block diagrams for the possible implementations are shown in separate sections.

Figure 30-1. Switched Capacitor and Continuous Time Block Diagram



30.3 How it Works

Each instance of the SC/CT block is able to implement any of the available configurations. Selection of the mode bits configures most of the resources required to implement these configurations.

30.3.1 Operational Mode of Block is Set

The operational mode of the SC/CT block is selected by setting the MODE[2:0] bits in the SC[0..3]_CR0 register, bits [3:1].

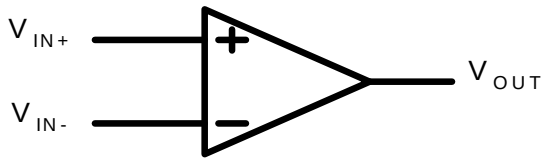
Table 30-1. SC/CT Block Operational Mode Settings

SC_MODE[2:0]	Operational Mode
[000]	Naked Opamp Mode
[001]	Trans Impedance Amplifier
[010]	Continuous Time Mixer
[011]	Discrete Time Mixer -- NRZ S/H
[100]	Unity Gain Buffer
[101]	First-Order Modulator
[110]	Programmable Gain Amplifier
[111]	Track and Hold Amplifier

30.4 Naked Opamp

The naked opamp mode provides direct access to the input and output terminals of the opamp. All of the other circuitry (resistors and capacitors) is disconnected in this mode. This mode is used for applications that require a general purpose opamp with external components.

Figure 30-2. Naked Opamp Configuration



The naked opamp is selected by setting the MODE[2:0] bits in the SC[0...3]_CR0 to 000. The opamp is a two stage design with a rail-to-rail input folded cascade first stage and a class A second stage. The opamp is internally compensated. To accommodate varying load conditions, the compensation capacitor and output stage drive strength is programmable.

The setting to apply is determined from the minimum required slew rate determined from the signal swing and time, and load capacitance. This is primarily a consideration for the stability reasons.

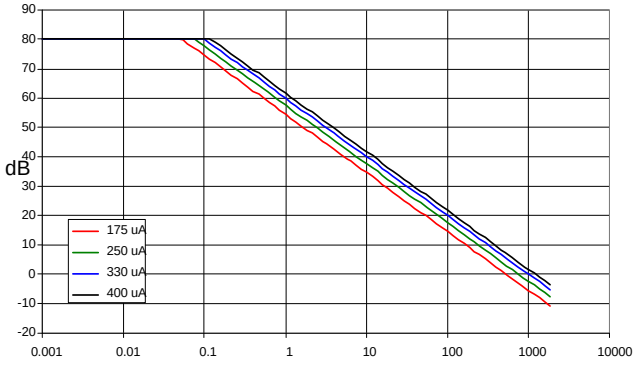
$$I_{load} = C_{load}\frac{\Delta V}{\Delta t} \qquad \textbf{Equation 1}$$

...where C_{load} includes the total internal capacitance at the output node of the amplifier plus any external capacitive loads. A value of 10 pF should be used for the internal load from analog bus routing. Set the drive controls, SC_DRIVE[1:0], according to the slew requirements at the output in SC[0..3]_CR1[1:0] register bits.

Table 30-2. Output Load Current by Drive Setting

SC_DRIVE[1:0]	I_load (µA)
2'b00	175
2'b01	250
2'b10	330
2'b11	400

Figure 30-3. Naked Opamp Drive Control I_LOAD



30.4.1 Bandwidth/Stability Control

This block has three control options for modifying closed loop bandwidth and stability that apply to all configurations: current through the first stage of the amplifier (BIAS_CONTROL), Miller capacitance between the amplifier input and the output stage (SC_COMP[1:0]), and feedback capacitance between the output stage and the negative input terminal (SC_REDC[1:0]).

30.4.1.1 BIAS_CONTROL

The bias control option doubles the current through the amplifier stage. AC open loop stability analysis for all continuous time modes shows that leaving this option set to ‘1’ and then controlling the bandwidth/stability using the capacitor options results in a greater overall bandwidth when the circuit is stabilized than using the option of less current in the first stage. The bias current is doubled by setting the SC[0..3]_CR2[0] register bit.

30.4.1.2 SC_COMP[1:0]

SC_COMP bits set the amount of compensation capacitance used in the amplifier. This directly affects the gain bandwidth of the amplifier and is an important tool in tuning the circuit stability. Follow the recommendations in the upcoming tables for this setting. The Miller capacitance is set to one of the four values in the SC[0..3]_CR1[3:2] register bits.

Table 30-3. Miller Capacitance between Amplifier Output and Output Driver

SC_COMP[1:0]	CMiller (pF)
00	1.30
01	2.60
10	3.90
11	5.20

30.4.1.3 SC_REDC[1:0]

The capacitance option between the output driver and the negative input terminal is another stability control option. Depending on the continuous time configuration, this capacitor option generally contributes to a higher frequency zero and a lower frequency pole, thus reducing the overall bandwidth and gaining some phase margin at the unity gain frequency. This capacitance is set to one of the four values in SC[0..3]_CR2[3:2] register bits.

Table 30-4. C_{FB} in CT Mix, PGA, Opamp, Unity Gain Buffer, and T/H Modes

SC_REDC[1:0]	C_{FB} (pF)
00	0.00
01	1.30
10	0.85
11	2.15

Recommended Settings by Mode

Stability settings for each mode are listed in Table 30-5 on page 262. These are the settings that were used to simulate each mode.

For the transimpedance amplifier (TIA) mode, the analog global load was modeled at the input as 10 pF between two 150 Ω switch impedances with an additional 40 pF added to the input to model the input diode capacitance.

For all continuous time modes, the output was modeled with two 150Ω switches with an 8 pF load in between, then followed b ya 300Ω impedance and a 50 pF external load.

The modulator mode was simulated with a 0.5 pF load at the output.

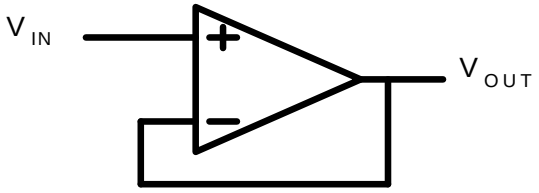
Table 30-5. Recommended Stability Settings by Mode

SC_MODE[2:0]	Operational Mode	BIAS_CONTROL	SC_COMP[1:0]	SC_REDC[1:0]
[001]	Trans Impedance Amplifier	1	3	3
[010]	Continuous Time Mixer	1	2	1
[011]	Discrete Time Mixer -- NRZ S/H	1	2	0
[100]	Unity Gain Buffer	1	2	0
[101]	First-Order Modulator	1	1	0
[110]	Programmable Gain Amplifier	See Table 30-7 on page 264		
[111]	Track and Hold Amplifier	1	2	0

30.5 Continuous Time Unity Gain Buffer

The Continuous Time Unity Gain Buffer is a naked opamp with the inverting input locally connected to the output. Use of routing features external to the block is not required to implement this function.

Figure 30-4. Unity Gain Buffer Configuration

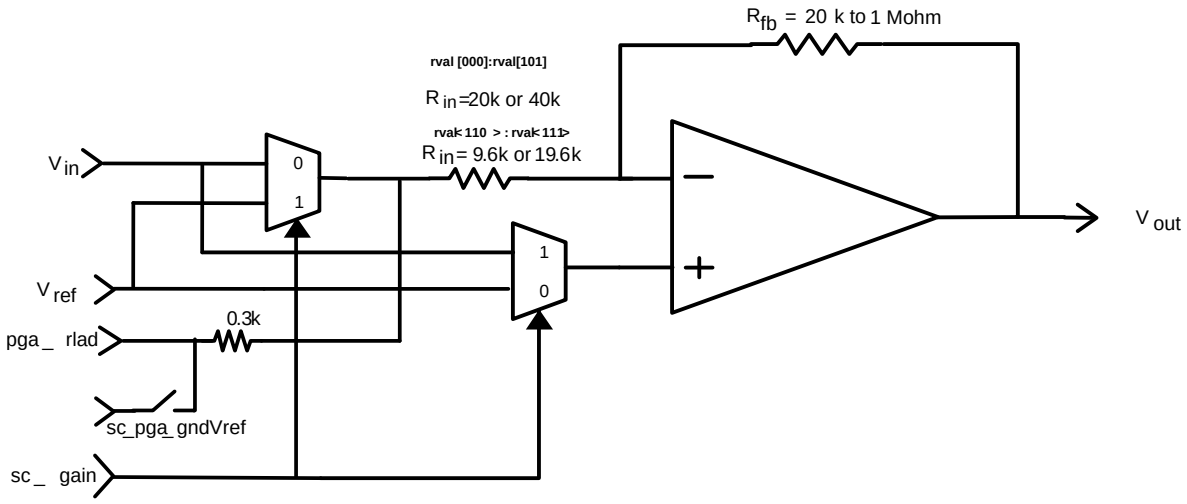


The unity gain buffer is used when an internally generated signal with high output impedance, such as a voltage DAC output, is required to drive a load; or when an external source with a high impedance is required to drive a significant on-chip load, such as the Continuous Time Mixer.

30.6 Continuous Time Programmable Gain Amplifier

The Programmable Gain Amplifier (PGA) is a continuous time opamp with selectable taps for input and feedback resistances. The PGA is selected by setting the MODE[2:0] bits in the SC[0...3]_CR0 register to ‘110’.

Figure 30-5. PGA Configuration



The PGA can be implemented as either a positive gain or negative gain topology, or as half of a differential amplifier. The specific gain configuration is selected by the SC_GAIN bit [5] in register SCL[0..3]_CR1. Any added input resistance from analog routing affects the PGA gain.

The positive gain (non-inverting) topology is shown in Figure 30-6.

Table 30-6. PGA Gain Configuration

SC_GAIN	Gain
0	Inverting ($-R_{FB}/R_{IN}$)
1	Non-inverting ($1+ R_{FB}/R_{IN}$)

Figure 30-6. PGA Positive Gain (Noninverting) Topology

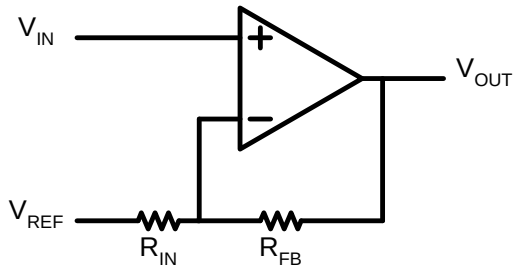


Figure 30-7. PGA Negative Gain (Inverting) Topology

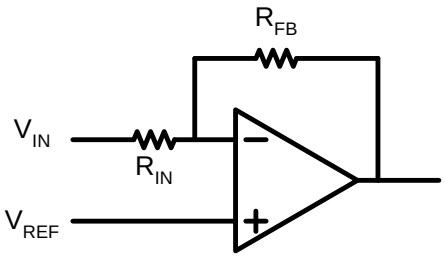
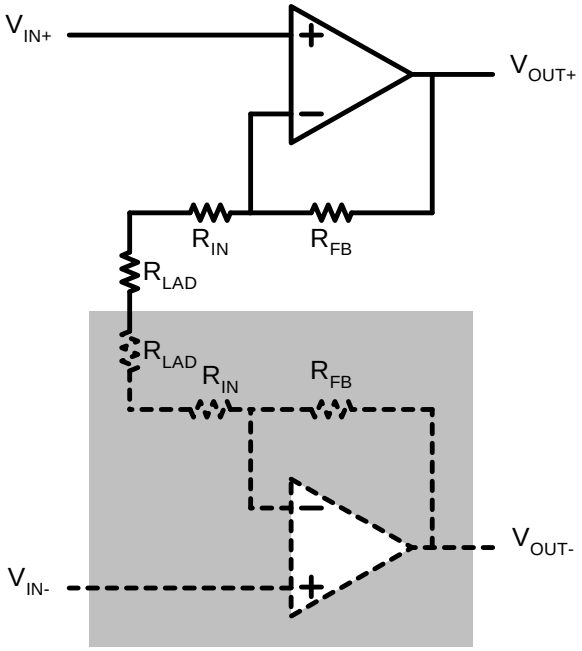


Figure 30-8. PGA Differential Amplifier Topology



The differential amplifier two PGAs in parallel. The connection (R_{LAD}) is external to the SC blocks and has very low impedance to reduce gain error. When not in differential mode, R_{IN} is connected to the analog or global routing and R_{LAD} is at very high impedance to minimize gain errors. The output of the differential amplifier is

$$V_{OUT}+ - V_{OUT}- = Gain*(V_{IN}+ - V_{IN}-). \quad \text{Equation 2}$$

The common mode voltage of the output remains at the common mode voltage of the input.

$$VCM = (V_{IN}+ + V_{IN}-)/2. \quad \text{Equation 3}$$

Because of capacitive loading, each gain step has a different requirement for compensation capacitors.

Table 30-7. PGA Stability Settings by Gain

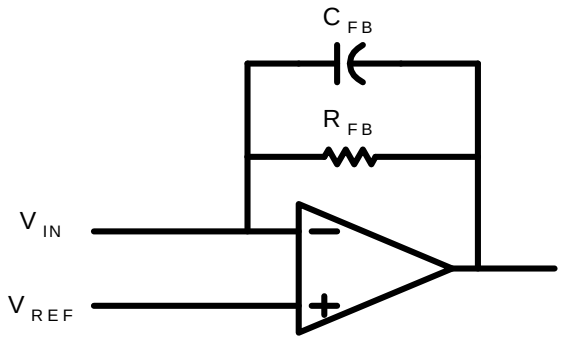
SC_RVAL[2:0]	R20_40B	Non-Inverting Gain (AC)	BIAS_CONTROL	SC_COMP[1:0]	SC_REDC[1:0]
Bin	Bin	Lin			
0	0	1	1	2	0
0	1	1	1	2	0
1	0	2	1	2	1
1	1	2	1	2	1
10	0	4	1	0	1
10	1	4	1	0	1
11	0	8	1	0	1
11	1	8	1	0	1
100	0	16	1	1	1
100	1	16	1	1	1
101	0	24	1	1	3
101	1	32	1	1	3
110	0	24	1	0	2
110	1	48	1	1	0
111	0	25	1	0	2
111	1	50	1	1	0

The negative gain (inverting) topology is shown in Figure 30-7.

30.7 Continuous Time Transimpedance Amplifier

The Transimpedance Amplifier (TIA) is a continuous time opamp with dedicated and selectable feedback resistor. The TIA is selected by setting the MODE[2:0] bits in the SC[0..3]_CR0 register to ‘001’.

Figure 30-9. Transimpedance Amplifier Configuration



The output of the transimpedance amplifier is a voltage that is proportional to input current; the conversion gain is a resistor value, where:

$$V_{OUT} = V_{REF} - (I_{IN} \cdot R_{FB}) \quad \textbf{Equation 4}$$

The output voltage is referenced to V_{REF}, which is routable to the analog globals or through local analog routing to any selected reference.

The feedback resistor can be programmed from 20 kΩ to 1.0 MΩ in eight steps, selected in bits [6:4] of the SCL[0..3]_CR2 register.

Table 30-8. Feedback Resistor Settings

SC_RVAL[2:0]	Nominal R_{FB} (kΩ)
000	20
001	30
010	40
011	80
100	120
101	250
110	500
111	1000

The feedback resistor is untrimmed polysilicon, so the absolute resistance value varies largely with process and temperature. Calibration of the TIA gain is expected to be done by the user using the precise outputs of the current output DAC combined with measurements in the ADC.

Stability of this opamp topology in general is affected by shunt capacitance on the inverting input. This capacitance is determined largely by parasitic capacitances in the analog global routing and at the input pin. An internal shunt feedback capacitor is used to maintain stability. Because the input capacitance is larger in the TIA than in other modes, the stability capacitance is somewhat larger.

The C_{FB} options for TIA mode are larger than for the other continuous time modes, as shown in Table 30-9. The feedback capacitance is set in bits [3:2] of the SCL[0..3]_CR2 register.

Table 30-9. Feedback Capacitance Settings

SC_REDC[1:0]	C_{FB} (pF)
00	0.00
01	1.30
10	3.30
11	4.60

A large source capacitance causes instability in the TIA with the small feedback resistor settings. Therefore, in applications where the internal capacitance is not sufficient to stabilize the TIA, an external capacitance is necessary. This is connected using the analog global routing.

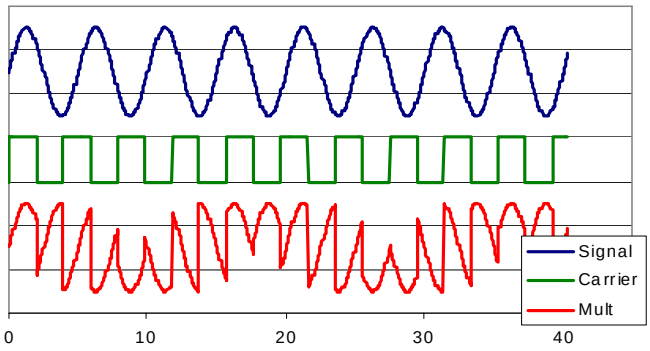
30.8 Continuous Time Mixer

The Continuous Time Mixer uses input switches to toggle a PGA between an inverting PGA gain of –1 and a noninverting PGA gain of +1. The maximum toggle frequency is 1 MHz. The continuous time mixer is selected by setting the MODE[2:0] bits in the SC[0..3]_CR0 register to ‘010’.

The continuous time mode was chosen to achieve up conversion because it provides higher conversion gain relative to the sampled mixer. In the CT mixer the magnitude of the $F_{CLK}+F_{IN}$ and $F_{CLK}-F_{IN}$ are equal, while in the sampled case, there is attenuation between the two configurations.

Example waveforms where the input is at 200 kHz and the carrier is at 255 kHz are shown in Figure 30-10.

Figure 30-10. Continuous Time Mixer Waveforms



The output spectrum of the mixer includes terms at 455 kHz, 55 kHz, at $3*f_{CARRIER} \pm f_{SIGNAL}$, $5*f_{CARRIER} \pm f_{SIGNAL}$, $7*f_{CARRIER} \pm f_{SIGNAL}$, etc. The up conversion is ultimately achieved by filtering out the required harmonic of the mixed product of the input frequency and modulating frequency using gain toggling.

Usage options for the continuous time mixer mode include controlling the sampling function and setting the value of the resistor in the inverting gain configuration. The continuous time mixer configuration can be seen in Figure 30-11.

Figure 30-11. Continuous Time Mixer Configuration

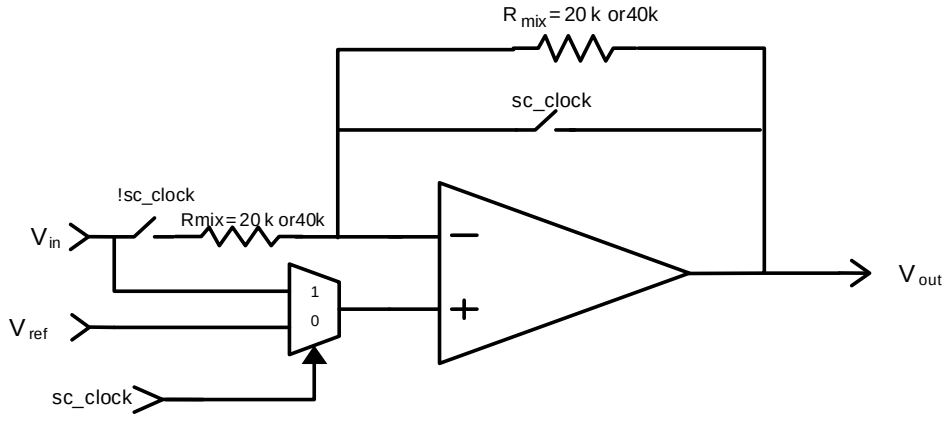


Table 30-10. Sampling Configurations for CT Mixer

SC_DYN_CNTRL	Configuration
0	Inverting Amplifier with Gain of 1
1	Unity Gain Buffer

Table 30-11. Input Resistor Settings for CT Mixer Inverting Mode

R20_40B	R_{MIX}
0	40 kΩ
1	20 kΩ

30.9 Sampled Mixer

The Sampled Mixer is a nonreturn-to-zero (NRZ) sample and hold circuit with very fast response. The mixer is selected by setting the MODE[2:0] bits in the SC[0..3]_CR0 register to '011'. The discrete time mode has a maximum F_{CLK} of 4 MHz. The maximum input frequency in discrete time mode is 14 MHz. The mixer output is designed to either drive an off-chip ceramic filter (i.e., 455 kHz Murata Cerafil) or the internal ADC through the on-chip analog routing. For the ADC to correctly sample the mixer output, the sample clock for the ADC and mixer must be the same.

The sample and hold mixer is primarily used for down-conversion mixing. The down conversion is achieved by filtering the required harmonics of the mixed product of the input frequency and sample clock frequency. Correct frequency planning is required to achieve the desired results. For a given input carrier frequency, F_{IN}, a sample clock frequency, F_{CLK}, can be chosen to provide the desired IF frequency, F_{IF}, for the system.

Provided that F_{CLK} is less than 4 MHz, and F_{IN} is less than 14 MHz:

If

$$\frac{2N-1}{2}F_{CLK} < F_{IN} < N \cdot F_{CLK} \qquad \textbf{Equation 5}$$

then

$$F_{IF} = N \cdot F_{CLK} - F_{IN} \qquad \textbf{Equation 6}$$

If

$$N \cdot F_{CLK} < F_{IN} < \frac{2N+1}{2}F_{CLK} \qquad \textbf{Equation 7}$$

then

$$F_{IF} = F_{IN} - (N \cdot F_{CLK}) \qquad \textbf{Equation 8}$$

Equation 1 and Equation 2 can be summarized as:

$$F_{IF} = abs(N \cdot F_{CLK} - F_{IN}) \qquad \textbf{Equation 9}$$

Consider an example using an input carrier frequency of 13.5 MHz and a desired IF frequency of 500 kHz. We set the sample clock frequency and the ADC sample frequency to be 2 MHz.

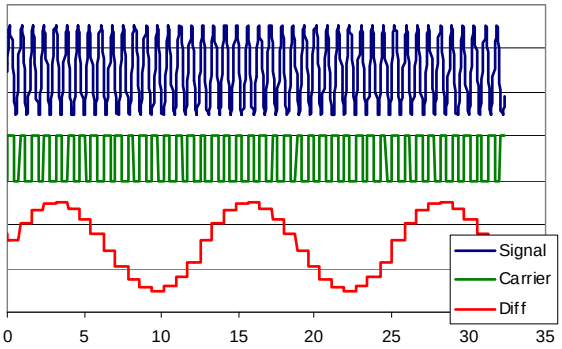
From the down conversion equations above, we can calculate the IF frequency with N = 7.

$$F_{IF} = 7 \cdot F_{CLK} - F_{IN} = 500kHz \qquad \textbf{Equation 10}$$

In this example, we have a 500 kHz down-converted signal, but we are sampling it at 2 MHz. Because the ADC and the switched capacitor block can both run at the same 2 MHz sample rate, there is no need to low pass filter the output of the switched capacitor block. Its output can be fed directly into the ADC input.

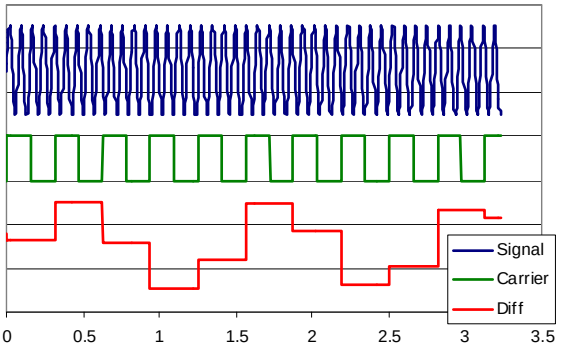
A few examples illustrate the frequency shifting capabilities of the mixer. For a signal frequency at 1.36 MHz, and a carrier at 1.28 MHz, the output frequency is the difference between the two frequencies, as shown in Figure 30-12.

Figure 30-12. Sampled Mixer N = 1



For a higher frequency signal at 13.6 MHz, and the carrier at 3.2 MHz, the output is at the same frequency, but longer separation between the samples, as shown in Figure 30-13.

Figure 30-13. Sampled Mixer N = 3

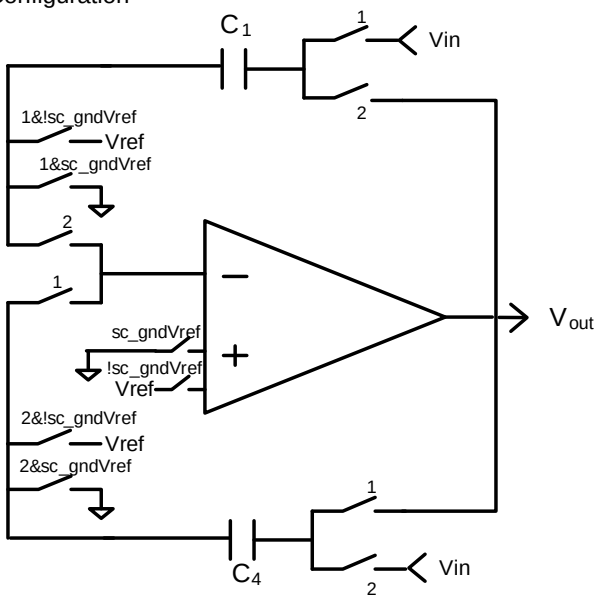


There is no increase in harmonic distortion, only an increase in the level of the sampling aliases. When the mixer output is sampled at the same rate as the carrier frequency, the aliases are suppressed.

The discrete time mixer configuration (NRZ S+H) is shown in Figure 30-14 on page 268. The options specific to this

configuration are the reference option and the clock division option.

Figure 30-14. Switched Capacitor Discrete Time Mixer Configuration



The option exists to either use an external reference voltage or to have the reference grounded internally. This option is controlled by the SC_GNDVREF SC[0..3]_CR2 signal as described in Table 30-12.

Table 30-12. External Reference Option for Sample and Hold Mixer

SC_GNDVREF	Amplifier/Capacitor Reference
0	External Voltage
1	Internal Ground

The use of the internal ground can cause different step sizes up versus down because the amplifier does not respond identically when the negative terminal jumps below ground. To avoid this distortion, use the external reference option and set it to 500 mV or greater.

The architecture of the discrete mixer is such that the output changes with a new hold value on both the rising and falling edge of the input clock. The SC_DIV control signal can be used to designate that output only change on the rising edge of the input clock. This is achieved by resetting the SC[0..3]_CR1[4] bit.

Table 30-13. Clock Division Option for Sample and Hold Mixer

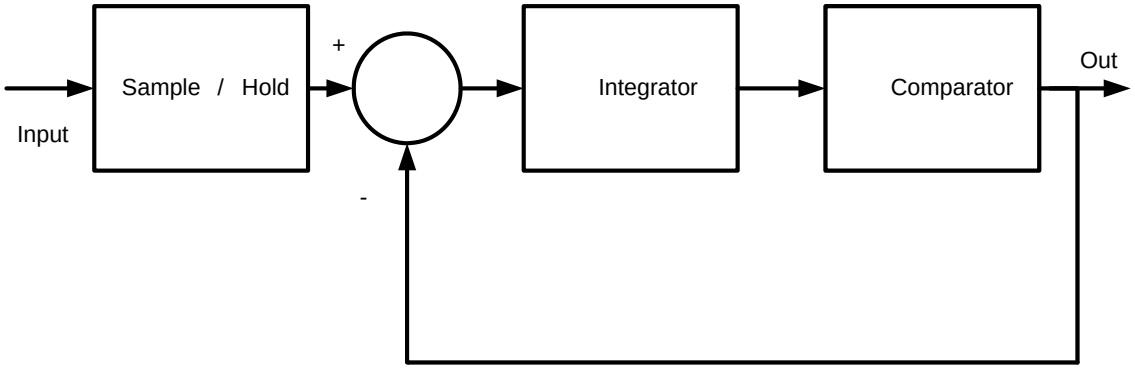
SC_DIV	SC_CLOCK Requirements
0	SC_CLOCK should be set to half the required sample frequency
1	SC_CLOCK should be set to the required sample frequency

30.10 Delta Sigma Modulator

The SC/CT block can be programmed to function as a switched capacitor integrator to use in a first-order modulator loop at high oversampling ratios.

The Delta Sigma Modulator is selected by setting the MODE[2:0] bits in the SC[0..3]_CR0 register to '101'. The integrator output is compared to a reference level and fed back to the input in a feedback loop. The modulator output is clocked at the high sampling rate, and needs to be decimated down to the signal band of interest using a decimation filter.

Figure 30-15. Discrete Time Delta Sigma Modulator Block Diagram



The modulator can also be used as an incremental modulator by using a reset switch that is placed across the integrating capacitor. The accuracy of the sampled data from the first-order modulator is determined from several factors: the maximum input signal bandwidth, oversampling ratio, and the sampling clock jitter. The oversampling clock is limited to a maximum of 4 MHz. Oversampling below x64 does not produce a stable output. Table 30-14 below shows the expected performance from a system simulation.

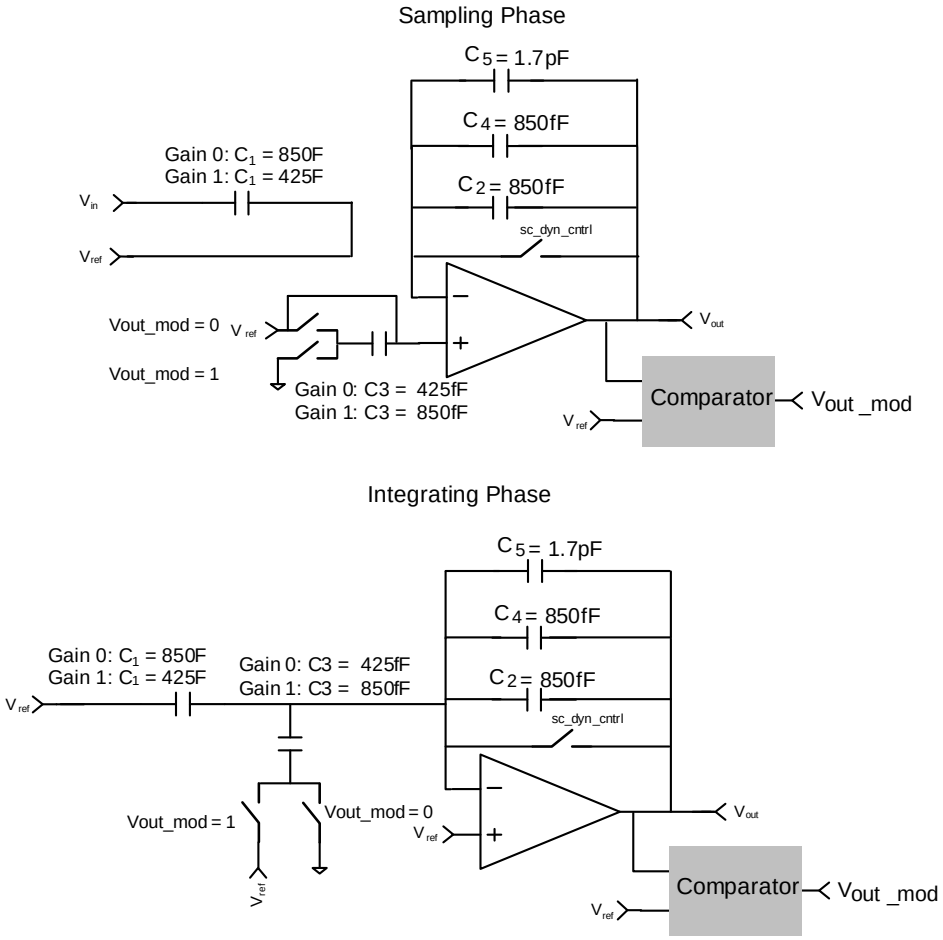
Table 30-14. Incremental Modulator Expected Performance from System Simulation

Maximum Input Signal Frequency	Oversampling Rate OSR (fsamp/fsig/2)	Sampling Clock Frequency (MHz)	Signal-to-Noise Ratio After Decimation by OSR (at Maximum Input Signal)
16 kHz	64	2.048	54 dB
8 kHz	128	2.048	64 dB
32 kHz	64	4.096	54 dB
16 kHz	128	4.096	64 dB

The Signal-to-Noise Ratio (SNR) values include the effects of limit cycle oscillations.

The configuration diagram of the discrete time first-order modulator is shown in Figure 30-16 on page 270. There are two mode-specific usage options: a reset switch placed across the integrating capacitor and a gain setting to adjust the allowable input amplitude range.

Figure 30-16. Switched Capacitor First-Order Modulator Configuration



30.10.1 First-Order Modulator, Incremental Mode

The dynamic control input SC[0..3]_CR1[5] can be used to reset the integrating capacitor if to perform an incremental conversion:

Table 30-15. First-Order Modulator, Integrating/Incremental Mode

SC_DYN_CNTRL	State
0	Integrating
1	Reset. V_{OUT} is connected to amplifier negative terminal.

The range of the allowed input amplitude can be set using the SC_GAIN SC[0..3]_CR1[5] control signal as shown in Table 30-16.

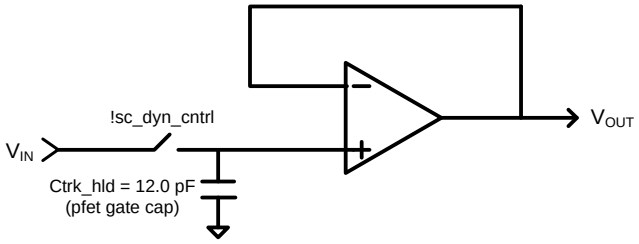
Table 30-16. First-Order Modulator, Input Amplitude

SC_GAIN	Maximum Input Amplitude
0	± half V_{REF}
1	± 2 V_{REF}

30.11 Track and Hold Amplifier

Track and hold amplifier mode is derived using the unity gain buffer amplifier. Implementation is shown in Figure 30-17.

Figure 30-17. Track and Hold Block Diagram



Track and hold mode tracks to 1 percent of a 5.5 V input step in less than 1 µs. The charge injection error from the sample switch is < 1.1 mV. The hold loss is < 0.2 mV.

The control of the amplifier between track and hold is done using the SC_DYN_CNTRL input as shown in Table 30-17.

This feature is enabled by setting the register bit value SC[0..3]_CLK[5].

Table 30-17. Track and Hold Amplifier Control

SC_DYN_CNTRL	Output
0	Track V_{IN}
1	Hold sampled value

31. Analog Routing

PSoC® 5 has a flexible analog routing architecture to route signals between GPIOs and analog resource blocks such as the ADC, switched capacitor, and DAC. One of the strong points of this routing architecture is that it allows dynamic configuration of input/output connections to the different analog blocks. For example, the comparator input can be switched between two GPIOs by DSI control signals and register settings at runtime. Knowing and understanding the routing architecture is essential to correctly and efficiently use the resources available. This knowledge will enable the designer to fit more functionality into the PSoC 5 and create projects that exhibit optimal analog performance.

31.1 Features

PSoC® analog routing has the following features:

- Flexible, configurable analog routing architecture
- Dedicated routing options for LCD drive capability
- Eight analog globals (AGs) and one analog multiplexer bus (AMUXBUS) for GPIOs on each side
- Flexible routing options within the analog core to interconnect analog resource blocks using the analog local bus (abus)

31.2 Block Diagram

The PSoC 5 analog system block diagram is shown in Figure 31-1 on page 274. Figure 31-2 on page 275 shows a detailed view of the analog routing architecture. All the figures used to explain analog routing in this chapter are derived from Figure 31-2.

Figure 31-1. Analog System Block Diagram

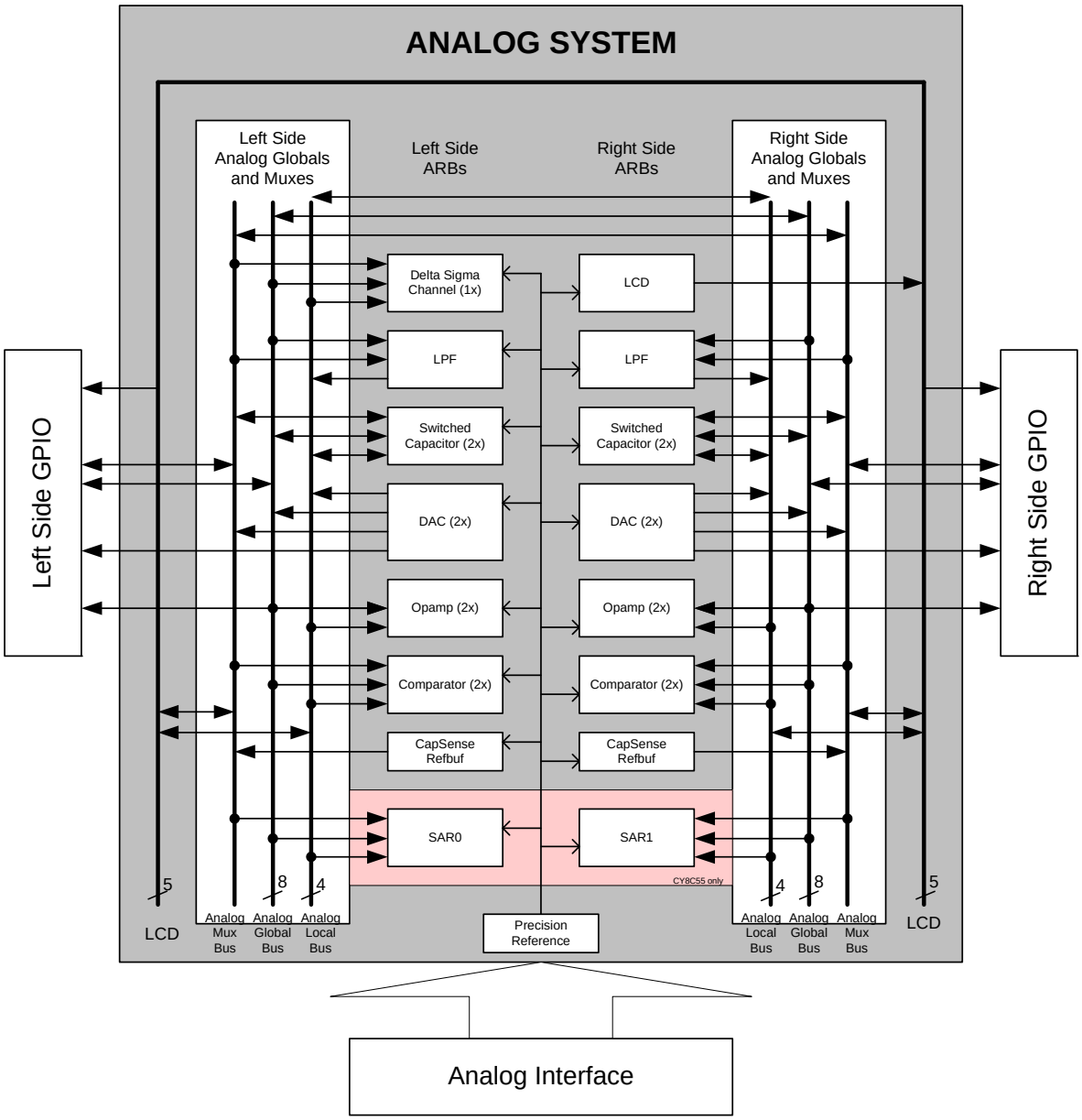
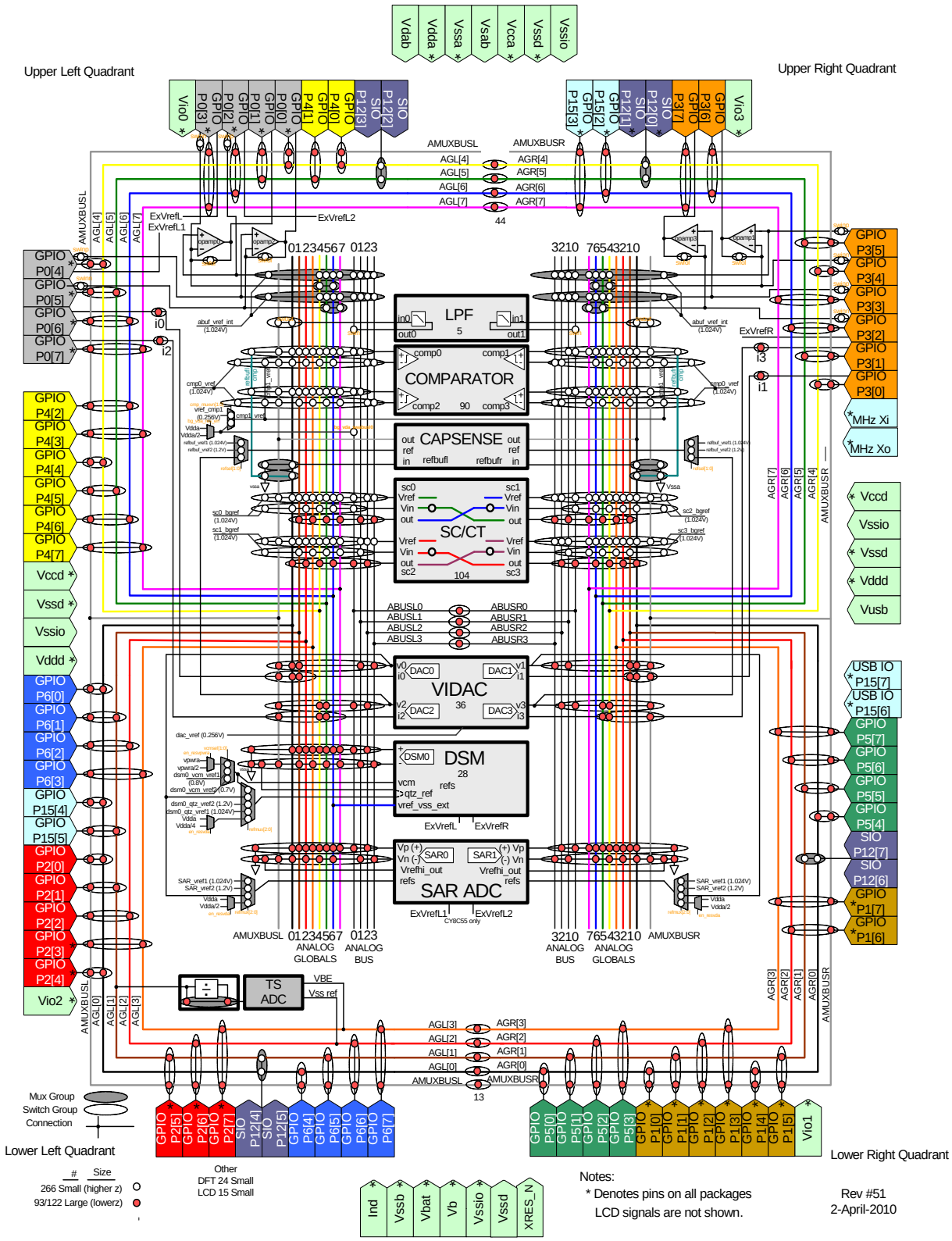


Figure 31-2. Analog Interconnect



31.3 How it Works

The analog routing resources in PSoC 5 devices include an analog mux bus (AMUXBUS), analog globals (AGs), a liquid crystal display bias bus (LCDBUS), and local analog buses (abus). The AMUXBUS and analog globals provide a way to route signals between the GPIOs and the analog resource blocks (ARBs). Analog resource blocks include the following: DACs, comparators, CapSense, switched capacitors, the Delta Sigma ADC, SAR ADCs, and opamps. The analog local buses (abus) are used for connections between ARBs. The LCDBUS is used for LCD bias signal routing.

In addition, there is a V_{REF} bus, as shown in Figure 31-2 on page 275. This V_{REF} bus carries the reference voltages for different analog blocks that are generated by the precision reference block. Refer to the Precision Reference chapter on page 315 for details about these reference voltages.

Analog switches and muxes establish connections between the above mentioned analog routing buses and the ARBs.

All these analog routing resources are explained in detail in the following sections. Figure 31-4 on page 278 illustrates the difference between switches and muxes.

31.3.1 Analog Globals (AGs)

The PSoC 5 die is divided into four quadrants, as shown in Figure 31-2 on page 275 and Figure 31-3 on page 277. The analog global bus has eight routes on each side, AGL[7:0] on the left, and AGR[7:0] on the right. Within each side, the bus is divided into two groups, AGR[3:0] and AGR[7:4] for the right side, and AGL[3:0] and AGL[7:4] for the left side. The lower four globals on each side are routed to the GPIO on the lower half of the die and the upper four globals on each side are routed to the GPIO on the upper half of the die. All eight analog globals on each side get routed to ARBs on the same side. Analog globals can be used as single ended or differential signal paths. The left and right half globals of the same number may operate independently or they may be joined through the switches that are shown at the top and bottom of Figure 31-3 on page 277.

Each GPIO may be connected to an analog global through a switch in the following manner:

- In the lower left half, Px[3:0] maps to AGL[3:0] and Px[7:4] maps to AGL[3:0]
- In the upper left half, Px[3:0] maps to AGL[7:4] and Px[7:4] maps to AGL[7:4]
- In the lower right half, Px[3:0] maps to AGR[3:0] and Px[7:4] maps to AGR[3:0]
- In the upper right half, Px[3:0] maps to AGR[7:4] and Px[7:4] maps to AGR[7:4]

This means that two pins on each port are connectable to the same global as shown in the diagram. The analog global bus connects to inputs and/or outputs of the following ARBs: DAC, comparator, output buffer, switched capacitor, Delta Sigma ADC, SAR ADC, and CapSense (which is a virtual block). These connections are made through switches and muxes, which are described in more detail in Switches and Multiplexers on page 278. PRT[x]_AG registers are used to configure the analog globals (AGs) for each GPIO port pin. Refer to 31.7 Analog Routing Register Summary on page 287 for register details.

Port 12 contains the Special Input/Output (SIO) pins. These pins are grouped in pairs for each quadrant of the device (lower right: P12[6] and P12[7], lower left: P12[4] and P12[5], upper left: P12[2] and P12[3], upper right: P12[0] and P12[1]), with each pair sharing a reference generation (REFGEN) block. The SIO REFGEN block can select from one of two analog globals routed to the pair shown in Figure 31-2 on page 275. The mux selection is controlled by the {PRT12_AG} register. Refer to the I/O System chapter on page 125 for details about SIO operation.

31.3.2 Analog Mux Bus (AMUXBUS)

The PSoC 5 device can be divided into two halves (left and right), with each half having one AMUXBUS (AMUXBUSR, AMUXBUSL). The left and right AMUXBUS may be shorted together with an analog switch. Every GPIO and ARB has the provision to connect to an AMUXBUS through an analog switch. Because the AMUXBUS can connect to every GPIO and ARB, it is the most versatile routing resource in PSoC 5. CapSense applications use the AMUXBUS for their operation. Refer to the CapSense® chapter on page 305 for details about using this bus for CapSense applications. PRT[x]_AMUX registers are used to configure the AMUXBUS routing for each GPIO port pin. Refer to 31.7 Analog Routing Register Summary on page 287 for register details.

31.3.3 Liquid Crystal Display Bias Bus (LCDBUS)

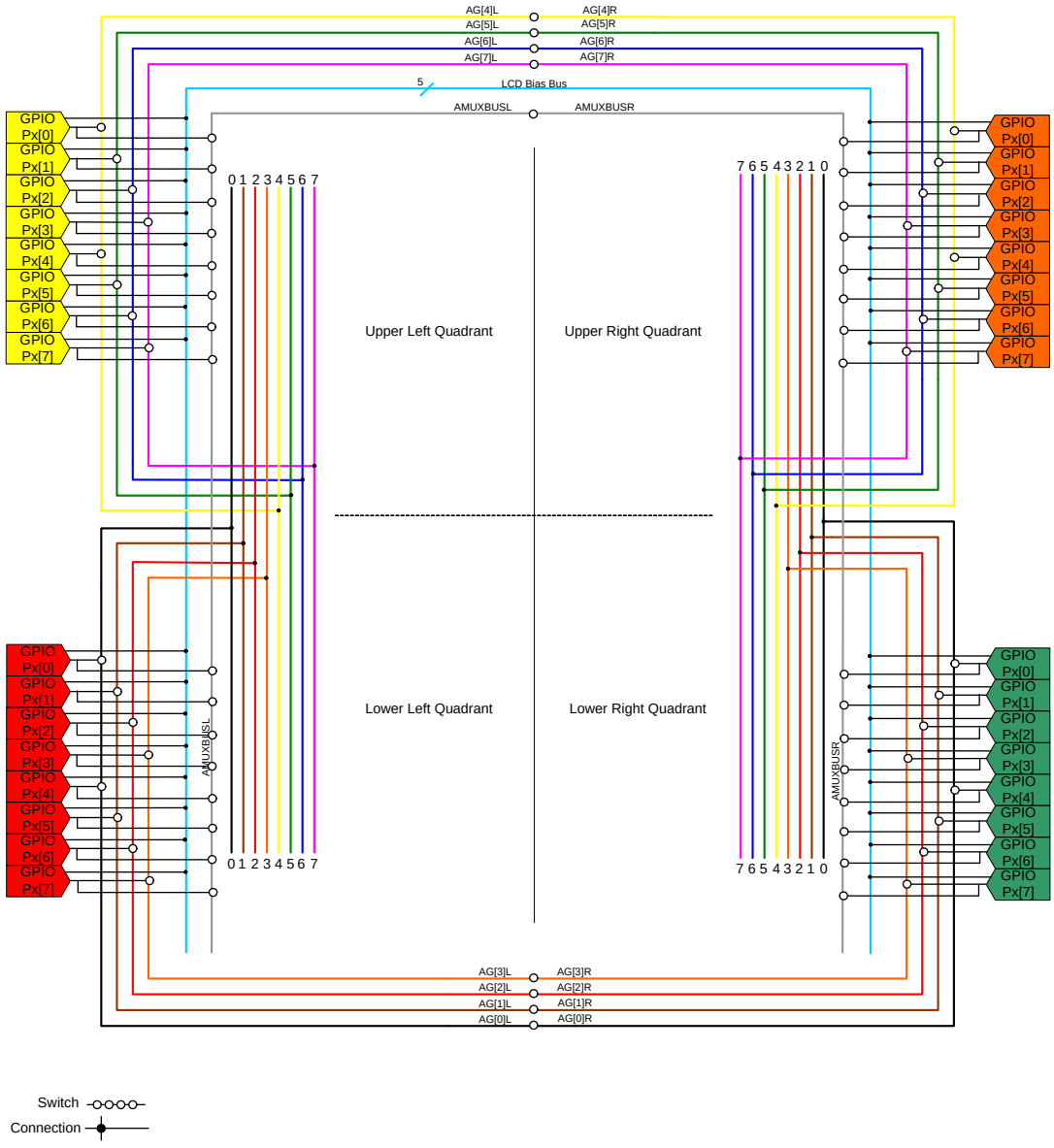
The LCD bias bus contains five routes that connect to every GPIO. These routes are continuous around the device periphery and are not separated by switches at the midline as are the analog globals and AMUXBUS. Each LCD route is individually configurable so that they are driven by the analog local bus or LCD bias voltage to the LCD driver buffer located in the GPIO. Connecting to an analog bus allows low frequency analog signals to be driven off the chip through the LCD driver buffers.

The LCDBUS mux selections are given in the following table. Refer to the LCD Direct Drive chapter on page 297 for LCD operation and biasing. Refer to 31.7 Analog Routing Register Summary on page 287 for register details.

Table 31-1. LCD Bias Bus Mux Selections

Output	Mux Selections
LCD_BIAS_BUS[0]	{0=LCDDAC_V0,1=abusr[0],2=abusl[0],3=NA}
LCD_BIAS_BUS[1]	{0=LCDDAC_V1,1=abusr[1],2=abusl[1],3=NA}
LCD_BIAS_BUS[2]	{0=LCDDAC_V2,1=abusr[2],2=abusl[2],3=NA}
LCD_BIAS_BUS[3]	{0=LCDDAC_V3,1=abusr[3],2=abusl[3],3=NA}
LCD_BIAS_BUS[4]	{0=LCDDAC_V4,1=AMUXBUSR,2=AMUXBUSL,3=NA}

Figure 31-3. Analog Globals, AMUXBUS, and LCDBUS Routing



31.3.4 Analog Local Bus (abus)

There are eight analog local bus (abus) routes in PSoC 5 devices: four in the left half (abusl[0:3]) and four in the right half (abusr[0:3]), as shown in Figure 31-2 on page 275. These are local routes located in the analog subsystem and are for interconnecting ARBs. They do not route to the GPIOs. It is possible to connect the left and right abus of the same number with an analog switch.

ARBs may connect to each other through analog globals (AG) or the analog local bus (abus). Because there are a limited number of available analog globals (eight per side), it is a good idea to route block to block connections using an abus when possible. For example, in Figure 31-2 on page 275, a DAC output (V1, for example) may be used as a reference for a comparator negative input (COMP1, for example). Using an analog switch, the DAC output could be placed on AGR0 and routed to the comparator by also connecting the comparator input to AGR0. However, to save AGR0 for GPIO connections, the DAC output (V1) can alternatively be routed to the analog local bus (abusr3) and then to the negative input of the comparator (COMP1). This saves the GPIO routing resource from being used for interconnecting two ARBs.

31.3.5 Switches and Multiplexers

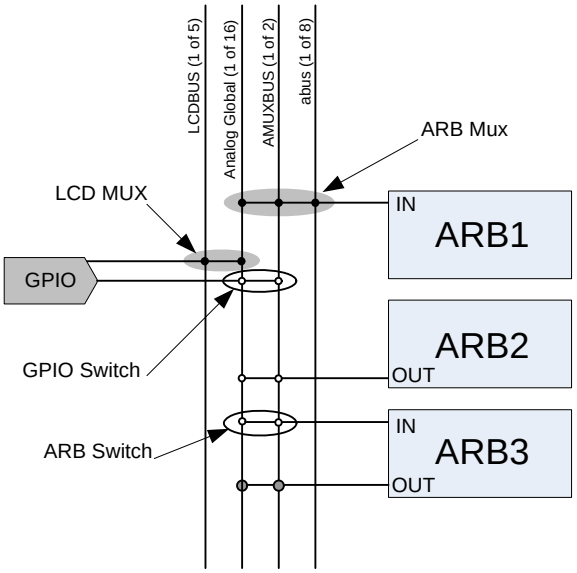
Switches and multiplexers are used to establish connections in the PSoC 5 analog routing architecture. They are placed on the various buses to direct signals into and out of the GPIOs and ARBs. The name “analog switch” is given to the single physical connection made to complete a route.

In a switch with ‘n’ inputs and one output, zero through ‘n’ analog switches may be on at a time, whereas in a multiplexer (mux) with ‘n’ inputs and one output, only one analog switch may be on at a time. Note that a group of eight analog switches requires eight bits for configuration, whereas, a mux with eight analog switches requires only three bits. Figure 31-4 illustrates the difference between switches and muxes.

For example, in Figure 31-4, there are two muxes (ARB, LCD). In both these muxes, only one of the analog switches can be selected for routing. In the same figure, there are two switches (GPIO, ARB). For these switches, more than one analog switch can be selected for routing. Note that both muxes and switches are formed using analog switches.

Each GPIO is connected through two analog switches to an analog global and an AMUXBUS. The ARBs use ARB switches and ARB muxes for input/output routing options.

Figure 31-4. Difference Between Switches and Muxes

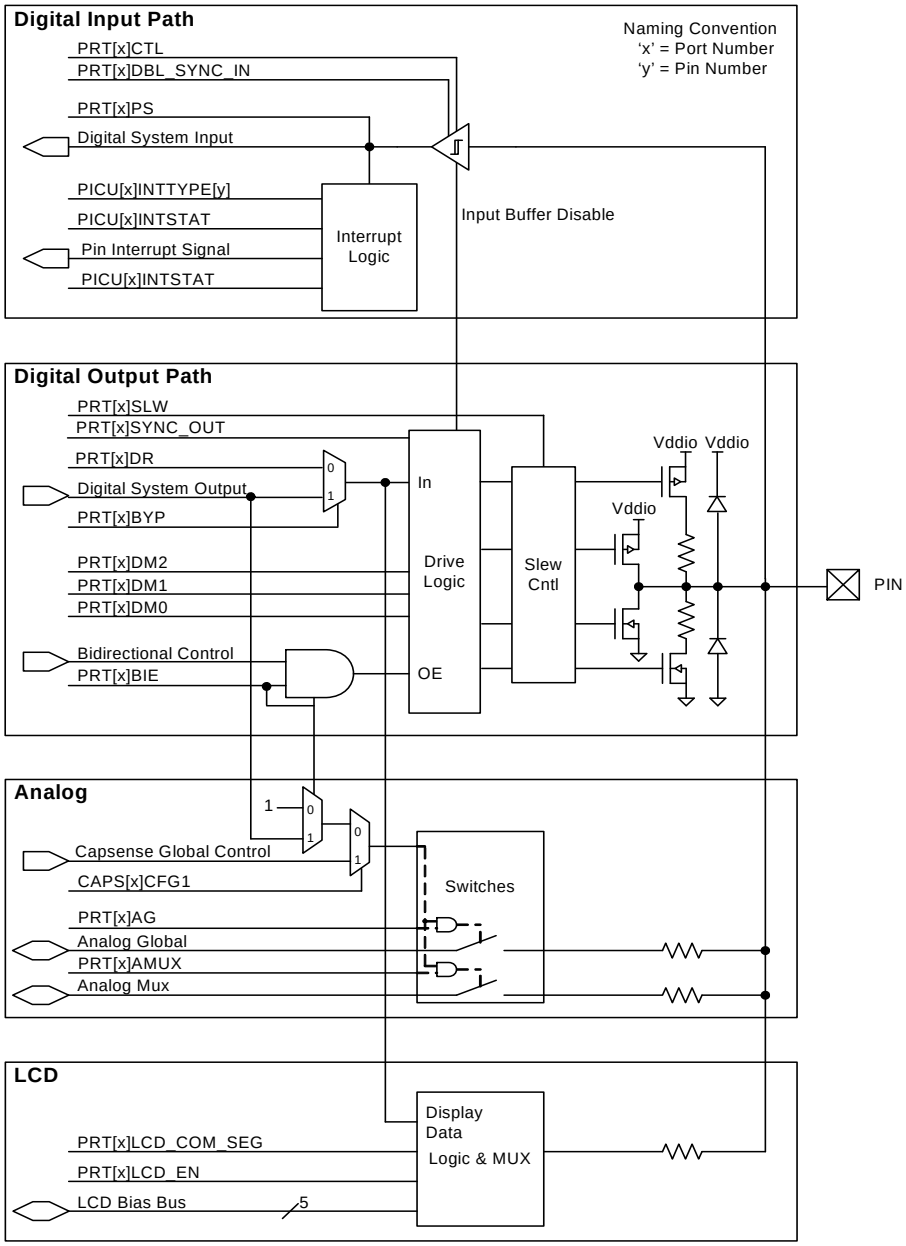


31.3.5.1 Control of Analog Switches

Each GPIO has two analog switches, one to connect the pin to the analog global and the other to connect the pin to the AMUXBUS. The open/close control signals for these analog switches can be generated by either of the following ways:

1. The registers corresponding to the GPIO pin, PRT[x]_AMUX and PRT[x]_AG, can be used to control the open/close state of the analog switches. This is the default option.
2. It is also possible to dynamically control these switches by means of the DSI control signal that is connected to the input of the port pin logic block. This option is enabled by setting the bit in the Port Bidirection Enable register (PRT[x]_BIE). For example, to control pin 3 of port 0, a value of 0x08 is written to PRT[0]_BIE. The switch control signal is the logical AND of the register setting, as in the first case, and the DSI control signal, as shown in Figure 31-5.

Figure 31-5. GPIO Pin Input/Output Block Diagram



In addition, there are control signals that are dedicated for CapSense applications as shown in Figure 31-5. Refer to the CapSense® chapter on page 305 for the usage of these control signals.

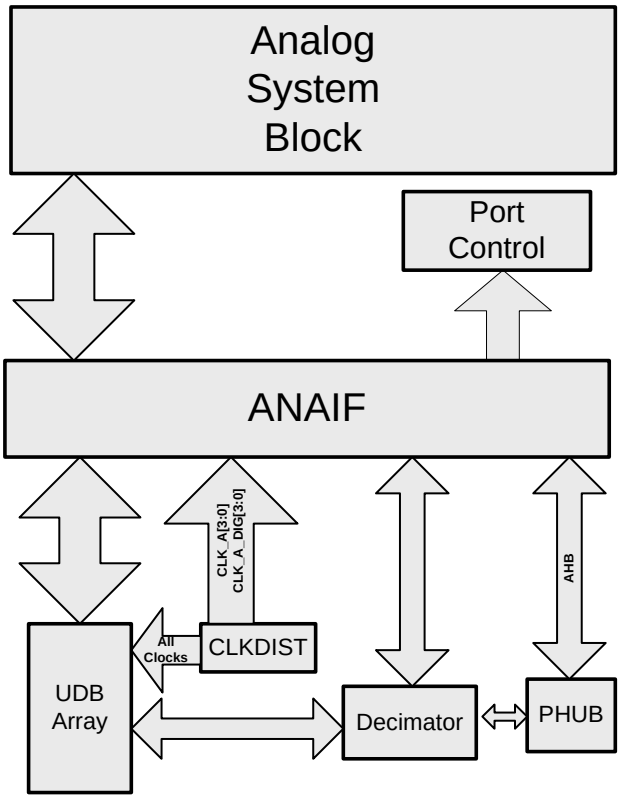
The analog switches corresponding to the analog resource blocks can be controlled only by the register settings of the respective ARBs.

For example, to switch a comparator input between two GPIOs that are connected to the same analog global, the register settings for the input select of the comparator are configured to select the analog global to which the GPIOs are connected. The DSI control signal can dynamically select between the two GPIOs after the corresponding PRT[x]_BIE register has been configured, but can not select which analog global is connected to the comparator input.

31.4 Analog Resource Blocks – Routing and Interface

The Analog Interface (ANAIF) is the interface between the analog blocks and other PSoC systems (UDB, DSI, clock, and decimator). The analog interface has 2 kilobytes of memory, which stores the configuration settings of all analog resource blocks. The configuration space is written to and read by the PHUB. The analog interface also interfaces clock distribution to the various analog resource blocks. For ARBs that deal with both analog and digital signals, such as the ADC, DAC, and comparator, the analog interface connects the digital and analog portions. For example, the comparator and modular (digital) outputs are routed to the Digital Systems Interconnect (DSI) through the analog interface. Similarly, the strobe and other digital signals for the DAC are routed through the analog interface. More details about how the resource block interfaces are controlled by the ANAIF are given in the individual chapters in Section F: Analog System on page 257. The following figure shows the top level diagram of the analog interface.

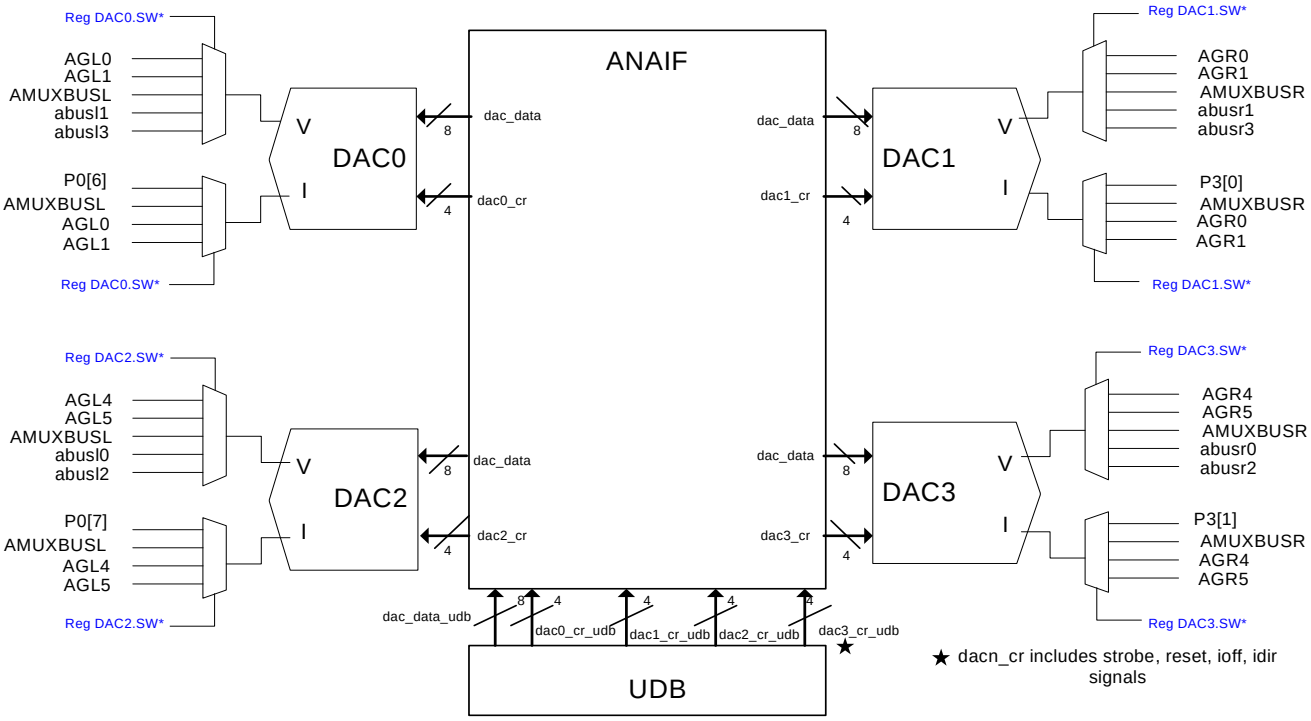
Figure 31-6. Analog Interface System Diagram



31.4.1 Digital-to-Analog Converter (DAC)

The DAC routing options and connections to other PSoC subsystems are shown in Figure 31-7. The output for each DAC is selected by control registers that are connected to multiplexer select lines. The DAC receives input data and control signals from the analog interface. The control signals include the strobe signal for the DAC, the reset signal, the DAC current-off signal, and output current direction. These control signals come from UDBs or control registers. Refer to the Digital-to-Analog Converter chapter on page 311 to learn more about DAC control and operation.

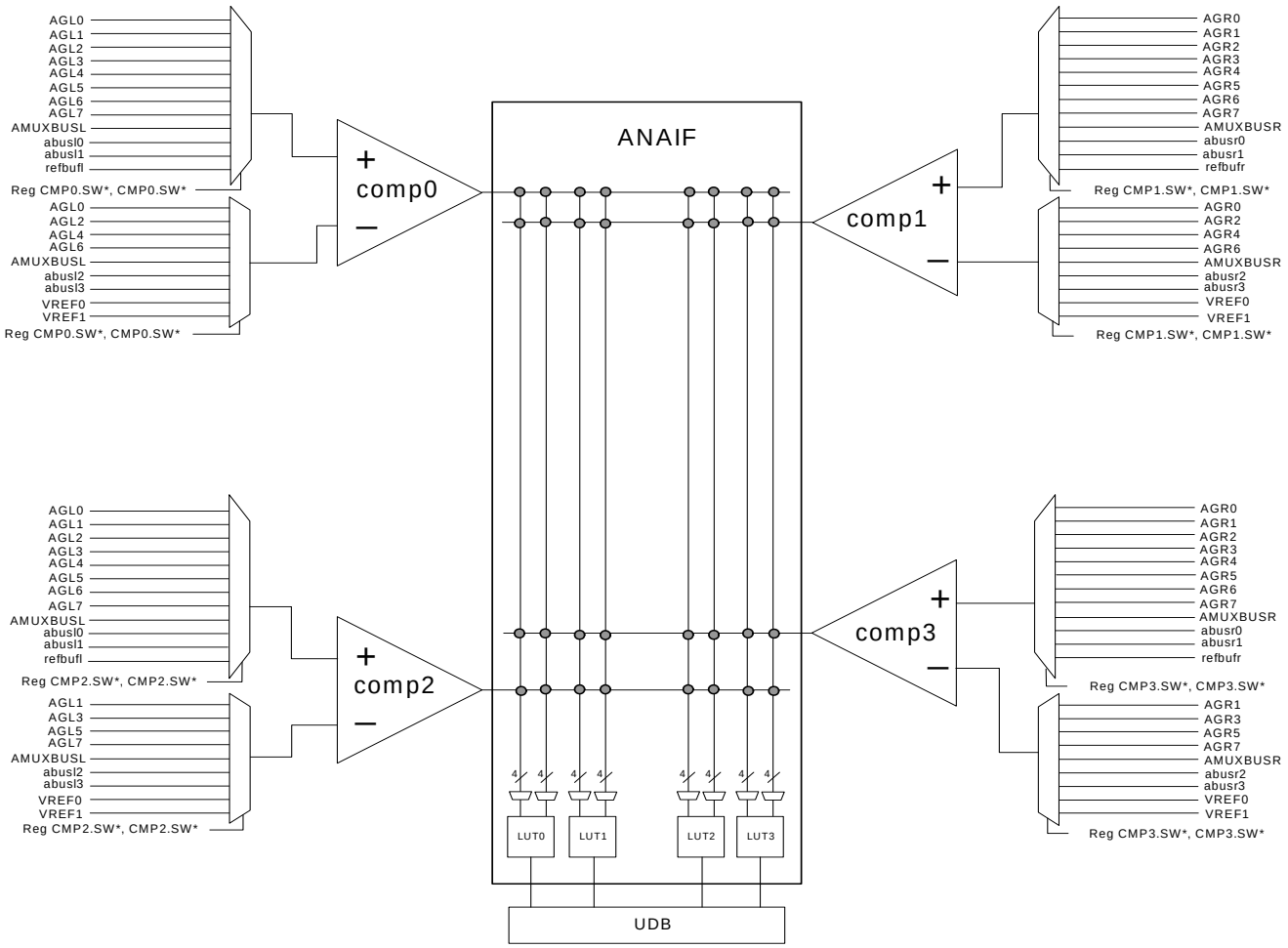
Figure 31-7. DAC Routing, Interface



31.4.2 Comparator

The comparator routing options and connections to other PSoC subsystems through the analog interface are shown in Figure 31-8. The input for each comparator is selected by control registers, which are connected to the multiplexer select lines.The outputs of the comparators are routed to the ANAIF for further processing (see the following figure). The analog interface contains lookup tables (LUTs) that are used to implement logic functions on comparator outputs. The LUT outputs (LUTN_OUT) are routed to the UDB block through the Digital System Interconnect (DSI). In addition, LUT outputs can generate interrupts (LUT_IRQ) to the device. Refer to the Comparators chapter on page 289 to learn more about comparator control and operation.

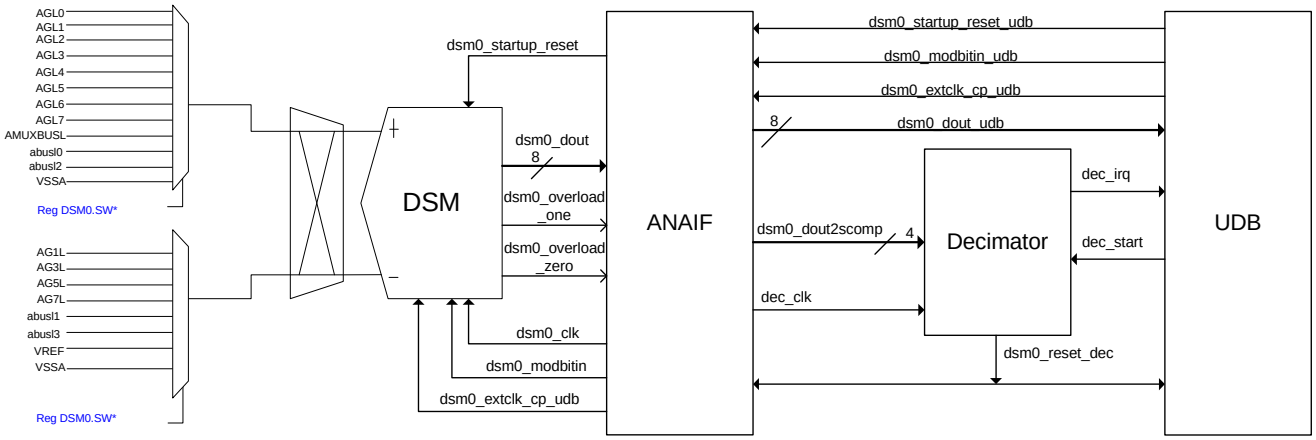
Figure 31-8. Comparator Routing, Interface



31.4.3 Delta Sigma Modulator (DSM)

The Delta Sigma modulator (DSM) is part of the Delta Sigma ADC and consists of various blocks, which are discussed in the Delta Sigma Converter chapter on page 319. The DSM can select its clock from any of the four analog clocks. The decimator block and the synchronization circuit in the ANAIF use the clock “CLK_DEC”, which is selected from the corresponding digitally aligned analog clocks. The DSM output DSM0_DOUT and the overload detect status bits are routed to the ANAIF block for post processing. DSM also receives the reset signals and modulation signal from the analog interface. These control signals may originate from UDBs and/or from control registers. Refer to the Delta Sigma Converter chapter on page 319 to learn more about the control and operation of this block.

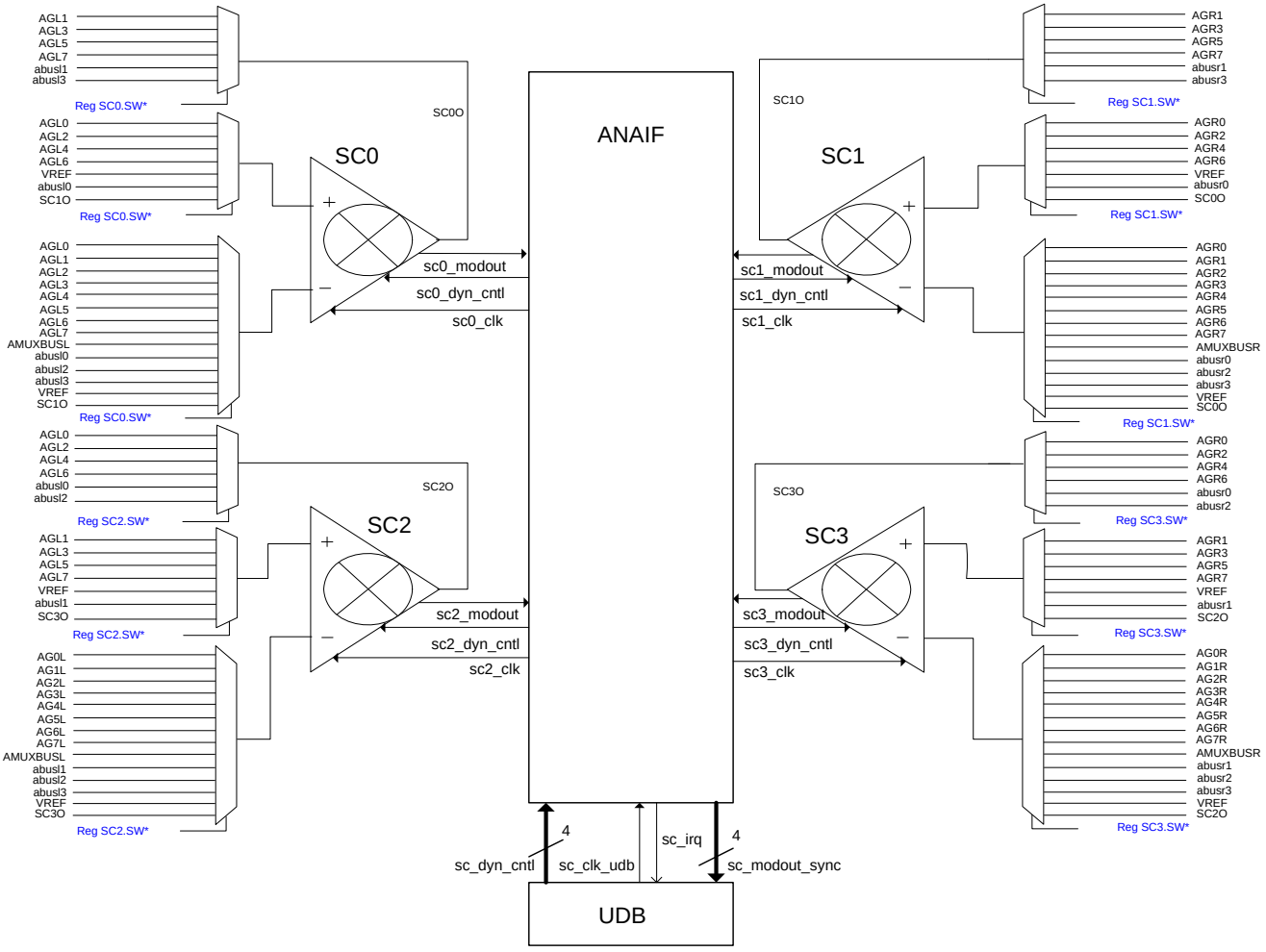
Figure 31-9. DSM Routing, Interface



31.4.4 Switched Capacitor

The switched capacitor block provides various analog functions. It has a modulator output SCN_MODOUT, which is routed to a register and is also routed to the UDB array as SCN_MODOUT_SYNC (Figure 31-10). The four analog clocks and the corresponding digitally aligned clocks, and the UDB generated clock, are selectable for each switched capacitor block instance. The interrupt signal corresponding to the switched capacitor blocks (SC_IRQ) is also routed to the UDB array. The polarity of the dynamic control input, SC_DYN_CNTRL, switches the amplifier between the inverting and non-inverting configuration. Refer to the Switched Capacitor/Continuous Time chapter on page 259 to learn more about SC/CT.

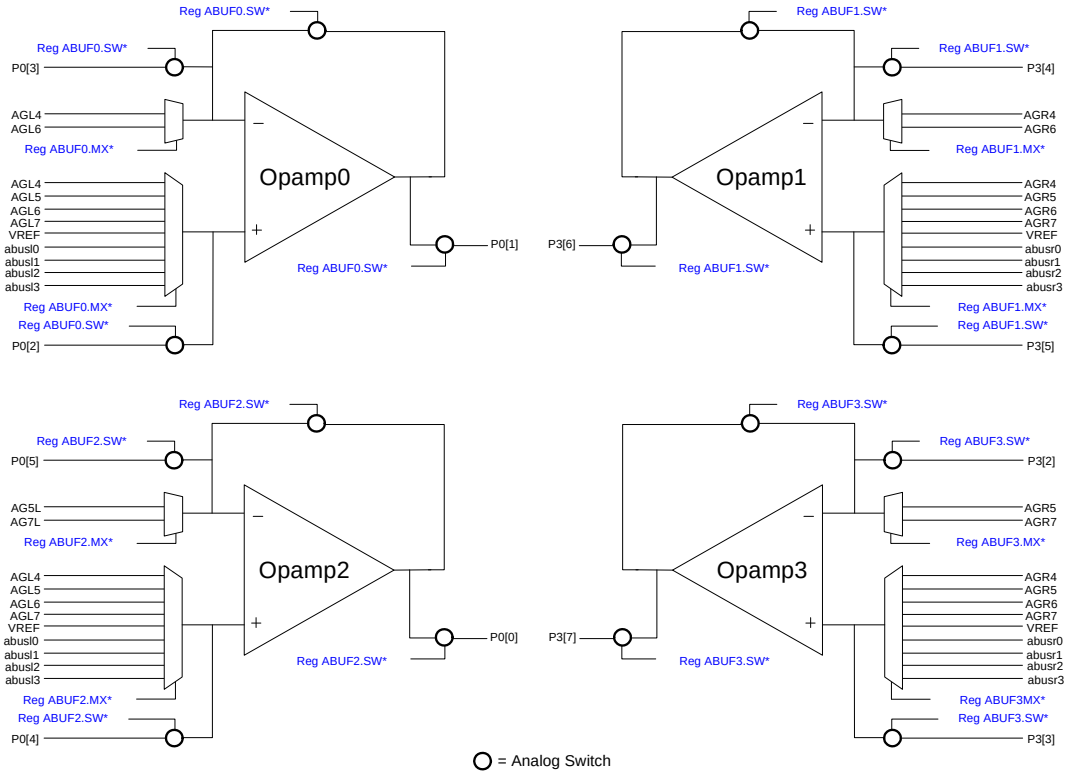
Figure 31-10. Switched Capacitor Routing, Interface



31.4.5 Opamp

The input and output routing options for the output buffer (opamp) are shown in Figure 31-11. Refer to the Opamp chapter on page 293 for details about configuration and operation of this block.

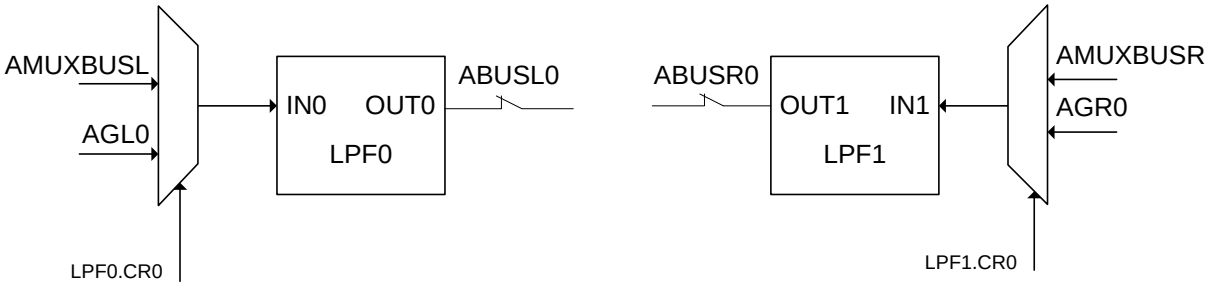
Figure 31-11. Opamp Input/Output Routing



31.4.6 Low Pass Filter (LPF)

Two tunable low pass filter blocks are available. The inputs are selectable in a 2:1 mux for each LPF as shown in Figure 31-12. On the left side, the LPF inputs are AMUXBUSL and AGL0. On the right side, the inputs are AMUXBUSR and AGR0. The outputs are connected through switches to abusL0 and abusR0, respectively. The tunability of the LPF allows the user to select an R of either 1 MΩ or 200 kΩ, and a C of either 5 pF or 10 pF. The LPF control registers are LPF0_CR0 and LPF1_CR0.

Figure 31-12. LPF Routing



31.5 Track Jumping

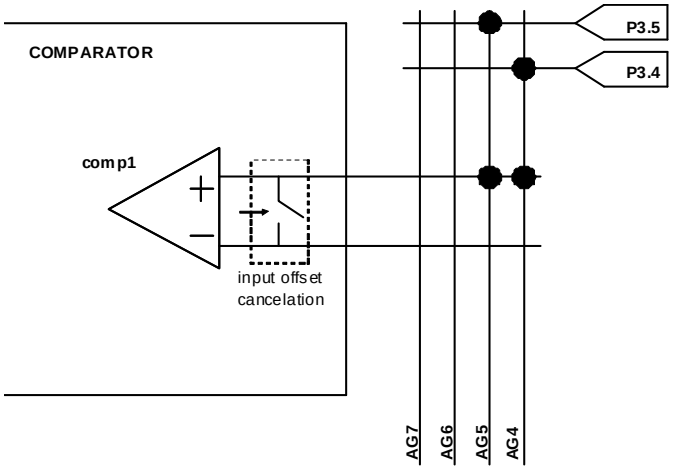
Figure 31-2 illustrates the analog global routing network, overlaid on top of the ARBs. As noted earlier, each ARB has a set of muxes and switches that it uses to connect to the global analog routing. By connecting to one of the analog routing channels virtually any ARB can be connected to any other ARB or pin on the chip.

Not all pins or ARBs are connected to every analog global routing channel. To get a signal from a particular ARB out to a specific pin, the PSoC Creator analog routing algorithm implements a technique known as “Track Jumping”. Track jumping connects two analog globals together via one of the ARBs analog global switching structures, without connecting to that particular ARB resource.

For example, assume we want to connect P3.5 to P3.4 in a simple pass-through configuration. This is illustrated in Figure 31-13. This illustration is taken from the full chip diagram shown in Figure 31-2. P3.5 enters the chip on analog global AG5. P3.4 enters the chip on analog global AG4. To connect these two pins together we need to track jump between AG4 and AG5. To do this, we can use the Comparator ARBs comp1 positive input switches (assuming the rest of our project is not using these switches). The switch for AG4 and AG5 on the Comparator comp1 positive input is closed, while the remaining switches remain open. The inputs to the Comparator ARB itself are isolated from the switch group via a transmission gate.

When this configuration is programmed into the device, any signal seen on P3.5 will show up on P3.4.

Figure 31-13. Simplified Diagram of Routing P3.5 to P3.4 Using Track Jumping on the Positive Input of Comparator 1.



31.6 Mitigating Analog Routes with Degraded Low Power Signal Integrity

When the analog blocks in PSoC 5 are powered down, some of the switches associated with them may become low impedance and provide path for current. It is recommended not to use manual routing when the project involves low power operation. The blocks that cause switches to go low impedance when powered down are listed in the following table. Follow the recommendations in the table when using the switches associated with these blocks for routing.

Block	Switches Affected	Recommendations while using switches of these blocks for routing
VIDAC	Output switches	Only disable it, do not power down the block
Switch Capacitor/Continuous Time Blocks	Input and output switches	Only disable it, do not power down the block
Comparator	Input switches	Only disable it, do not power down the block
Delta Sigma Converter	Input and output switches	Enable and power on. Use low power mode for the block to minimize power
SAR ADC	Input and output switches	Enable and power on. Use low power mode for the block to minimize power

31.7 Analog Routing Register Summary

Table 31-2. Analog Routing Register Summary

Name	Brief Description
{PRT[0..11]_AMUX} {PRT15_AMUX}	These registers control the connection between the analog mux bus and the corresponding GPIO pin
{PRT[0..11]_AG} {PRT12_AG} {PRT15_AG}	These registers control the connection between the analog global buses and the corresponding GPIO pin. Port 12 is the SIO port and the PRT12_AG register is for SIO reference selection.
{CMP[0..3]_SW0}	Comparator positive input to analog globals 0-7
{CMP[0..3]_SW2}	Comparator positive input to analog local bus
{CMP[0..3]_SW3}	Comparator positive input to AMUXBUS and reference buffer Comparator negative input to AMUXBUS and V_{REF}
{CMP[0..3]_SW4}	Comparator negative input to analog globals 0–7
{CMP[0..3]_SW6}	Comparator negative input to analog local bus
{CMP[0..3]_CLK}	Comparator sampling clock selection and clock control register
{DSM0_SW0}	Delta Sigma modulator positive input to analog globals 0–7
{DSM0_SW2}	Delta Sigma modulator positive input to analog local bus
{DSM0_SW3}	Delta Sigma modulator positive input to AMUXBUS and V_{SSA} Delta Sigma modulator negative input to AMUXBUS, V_{SSA}, and V_{REF}
{DSM0_SW4}	Delta Sigma modulator negative input to analog globals 0–7
{DSM0_SW6}	Delta Sigma modulator negative input to analog local bus
{DSM0_CLK}	Delta Sigma modulator clock selection
{DAC[0..3]_SW0}	DAC voltage output to analog globals 0–7
{DAC[0..3]_SW2}	DAC voltage output to analog local bus
{DAC[0..3]_SW3}	DAC voltage output to AMUXBUS DAC current output to AMUXBUS and direct to pad
{DAC[0..3]_SW4}	DAC current output to analog globals 0–7
{DAC[0..3]_SW6}	DAC current to analog local bus
{DAC[0..3]_STROBE}	DAC strobe selection
{SC[0..3]_SW0}	Switched capacitor (SC) V_{REF} (negative) input to analog globals 0-7
{SC[0..3]_SW2}	SC V_{REF} (negative) input to analog local bus
{SC[0..3]_SW3}	SC V_{REF} (negative) input to AMIUXBUS and 1.024-V V_{REF} SC V_{IN} (positive) input to AMUXBUS and 1.024-V V_{REF}
{SC[0..3]_SW4}	SC V_{IN} (positive) input to analog globals 0–7
{SC[0..3]_SW5}	SC V_{IN} (positive) input to analog globals 8–15
{SC[0..3]_SW6}	SC V_{IN} (positive) input to analog local bus
{SC[0..3]_SW7}	SC output to AMUXBUS, other SC inputs
{SC[0..3]_SW8}	SC output to analog globals 0-7
{SC[0..3]_SW10}	SC output to analog local bus
{SC[0..3]_CLK}	SC clock selection
{ABUF[0..3]_MX}	These registers select positive and negative inputs to the output buffer.
{ABUF[0..3]_SW}	These registers control the switch between the output and negative input, the switch between the output and GPIO, the switch between the negative input and GPIO, and the switch between the positive input and GPIO.
{LUT[0..3]_CR}	These registers select the signals to comparator LUT and also select the LUT function.
{LCDDAC_SW[0:4]}	These registers select the signals on the LCD bias bus.
{BUS_SW0}	This register controls the switches that tie AGR[7:0] to AGL[7:0].
{BUS_SW2}	This register controls the switches that tie abusL[7:0] to abusR[7:0] (left and right analog local bus) lines together.
{BUS_SW3}	This register controls the switch that ties AMUXBUSR to AMUXBUSL.
{LPF0_CR0} {LPF1_CR0}	LPF registers

32. Comparators

PSoC® 5 devices have four analog comparator modules. The positive and negative inputs to the comparators come through muxes with inputs from analog globals (AGs), local analog bus (ABUS), Analog Mux Bus (AMUXBUS), and precision reference. The output from each comparator is routed through a synchronization block to a two-input Lookup Table (LUT). The output of the LUT is routed to the UDB Digital System Interface (DSI). An ‘x’ used with a register name denotes the particular comparator number (x = 0 to 3).

32.1 Features

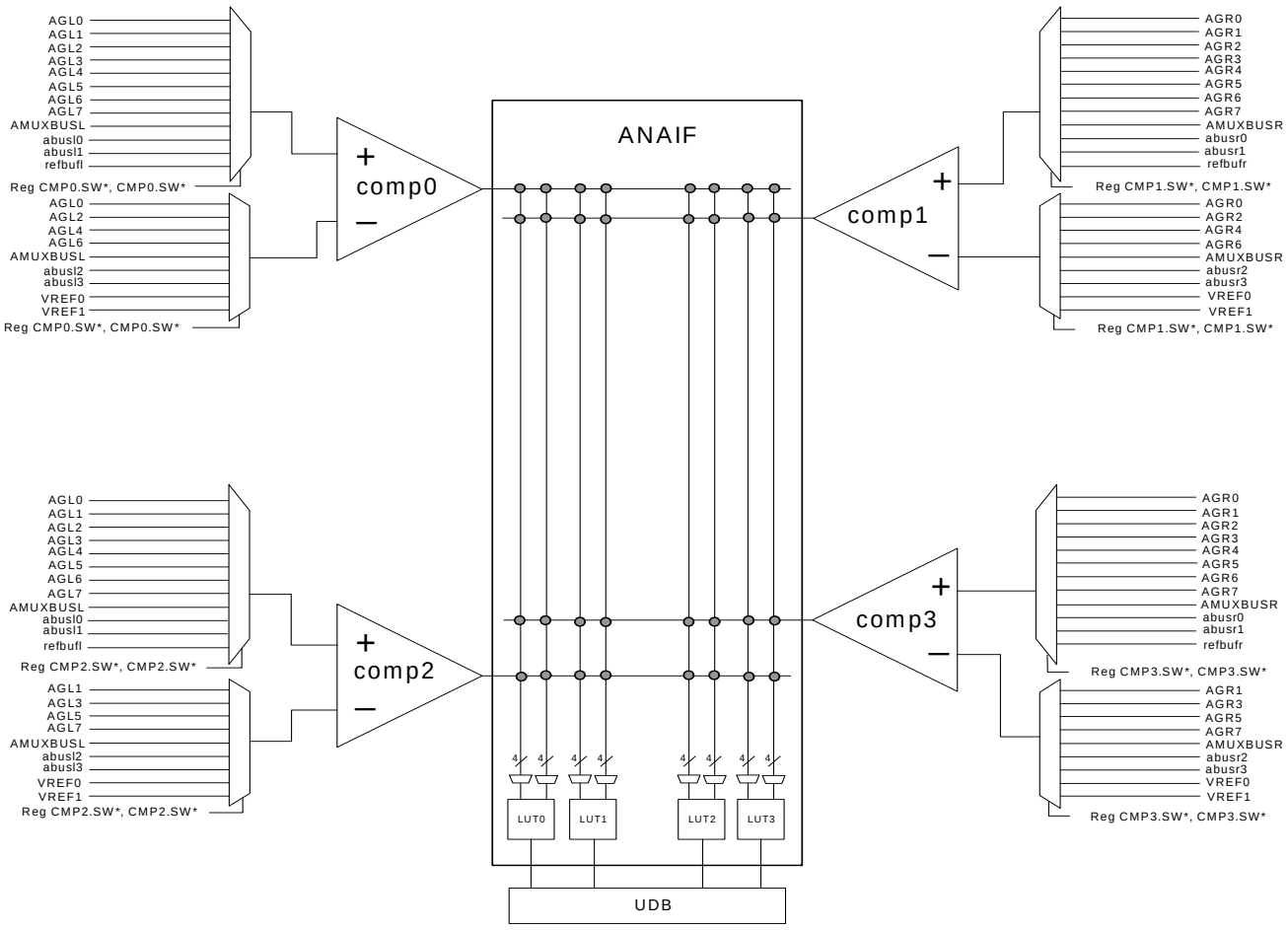
PSoC® comparators have the following features:

- Flexible input selection
- Speed power tradeoff
- Optional 10 mV input hysteresis
- Low input offset voltage (<15 mV)
- Glitch filter for comparator output

32.2 Block Diagram

Figure 32-1 on page 290 is a block diagram of PSoC Comparators.

Figure 32-1. Comparator Block Diagram



32.3 How it Works

The following describes the operation of PSoC comparators.

32.3.1 Input Configuration

Inputs to the comparators are as follows:

- Positive – from analog globals, analog locals, analog mux bus, and comparator reference buffer. Refer to the CapSense® chapter on page 305.
- Negative – from analog globals, analog locals, analog mux bus, and voltage reference.

All of the possible connections to the positive and negative inputs are shown in Figure 32-1. Inputs are configured using registers CMPx_SW0, CMPx_SW2, CMP_SW3, CMP_SW4, and CMP_SW6.

32.3.2 Power Configuration

The comparator can operate in three power modes – fast, slow, and ultra low power. The power mode is configured using power mode select (SEL[1:0]) bits in the comparator control (CMPx_CR) register. The output of the comparators may glitch when the power mode is changed.

Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ultra low power mode. Exact specifications for power consumption and response time are provided in the data sheet.

32.3.3 Output Configuration

Comparator output can pass through an optional glitch filter. The glitch filter is enabled by setting the filter enable (FILT) bit in the control (CMPx_CR[6]) register. The output of the comparator is stored in the CMP_WRK register and can be read over the PHUB interface.

Four LUTs in the device allow logic functions to be applied to comparator outputs. LUT logic has two inputs:

- Input A – selected using MX_A[1:0] bits in LUT control (LUTx_CR1:0) register
- Input B – selected using MX_B[1:0] bits in LUT Control (LUTx_CR5:4) register

The logic function implemented in the LUT is selected using control (Q[3:0]) bits in the LUT Control register (LUTx_CR) register. The bit settings for various logic functions are given in Table 32-1.

Table 32-1. Control Words for LUT Functions

Control Word (Binary)	Output (A and B are LUT Inputs)
0000	FALSE('0')
0001	A AND B
0010	A AND (NOT B)
0011	A
0100	(NOT A) AND B
0101	B
0110	A XOR B
0111	A OR B
1000	A NOR B
1001	A XNOR B
1010	NOT B
1011	A OR (NOT B)
1100	NOT A
1101	(NOT A) OR B
1110	A NAND B
1111	TRUE ('1')

The output of the LUT is routed to the digital system interface of the UDB array. From the digital system interface of the UDB array, these signals can be connected to other blocks in the device or to an I/O pin.

The state of the LUT output is indicated in the LUT output (LUTx_OUT) bit in the LUT clear-on-read sticky status (LUT_SR) register and can be read over PHUB interface.

The LUT interrupt can be generated by all four LUTs and is enabled by setting the LUT mask (LUTx_MSK) bit in the LUT mask (LUT_MSK) register.

32.3.4 Hysteresis

For applications that compare signals very close to each other, hysteresis helps to avoid excessive toggling of the comparator output when the signals are noisy.

The 10 mV hysteresis level is enabled by setting the hysteresis enable (HYST) bit in the control (CMPx_CR5) register.

32.3.5 Comparator Clock

Comparator output changes asynchronously and can be synchronized with a clock. The clock source can be one of the four digitally-aligned analog clocks or any UDB clock.

Clock selection is done in mx_clk bits [2:0] of CMP_CLK register. The selected clock can be enabled or disabled by setting or clearing the clk_en (CMP_CLK [3]) bit. Comparator output synchronization is optional and can be bypassed by setting the bypass_sync (CMP_CLK [4]) bit.

32.3.6 Offset Trim

Comparator offset is dependent on the common mode input voltage to the comparator. The offset is factory trimmed for common mode input voltages 0.1 V and V_{dd} - 0.1 V to less than 1 mV. If the user knows the common mode input range at which to operate the comparator, a custom trim can be done to reduce the offset voltage further.

The comparator offset trim is performed in the CMPx_TR0 register. This register has two trim fields, trim1 (CMPx_TR0[3:0]) and trim2 (CMPx_TR0[7:4]). If shorting of the inputs is wanted for offset calibration, the calibration enable field (cal_en) in the control register (CMP_CR[4]) helps to achieve it

The method for a custom trim is as follows:

1. Set the two inputs 'inn' and 'inp' to the desired value.
2. Change the trim1 register settings:
   a. Depending on the polarity of the offset measured, set or clear trim1 [3] bit.
   b. Increase the value of trim1 [2:0] until offset measured is less than 1 mV.
3. If the polarity of the offset measured has changed, but the offset is still greater than 1 mV, use trim2 [3:0] to fine tune the offset value. This is valid only for the slow mode of comparator operation.
4. If trim1 [2:0] is 07h, and the measured offset is still greater than 1 mV, set or clear trim2 [3], depending on the polarity of offset. Increase the value of trim2 [2:0] until the offset measured is less than 1 mV.

32.3.7 Register Summary

Table 32-2 is a summary listing of applicable registers.

Table 32-2. Registers

Register	Function
CMPx_SW0	Configures connection between positive input and analog globals 0-7
CMPx_SW2	Configures connection between positive input and analog locals 0-1
CMPx_SW3	Configures connection between analog mux bus to the two inputs and the voltage reference to negative input, CapSense® reference buffer to the positive input
CMPx_SW4	Configures connection between negative input and analog globals 0-7
CMPx_SW5	
CMPx_SW6	Configures connection between negative input and analog locals 0-1
CMPx_TR0	Trims the offset. Two groups of 4-bits for lower and higher end of common mode input ranges.
CMP_WRK	Stores the output state of the comparator
CMPx_CLK	These registers enable and disable synchronization of the output for comparators and the clock signal for synchronization
CMPx_CR	These registers are used to select the mode of operation of the comparator between the high speed and low speed modes and to enable/disable the comparator channel
LUTx_CR	Selects the input(s) and function for the LUT
LUT_SR	Stores the status of LUT outputs. It's a clear on read register.
LUT_MSK	Enables interrupt request for a particular LUT output

33. Opamp

PSoC® 5 devices have four operational amplifiers. An ‘x’ used with register name identifies the particular opamp number (x = 0 to 3).

33.1 Features

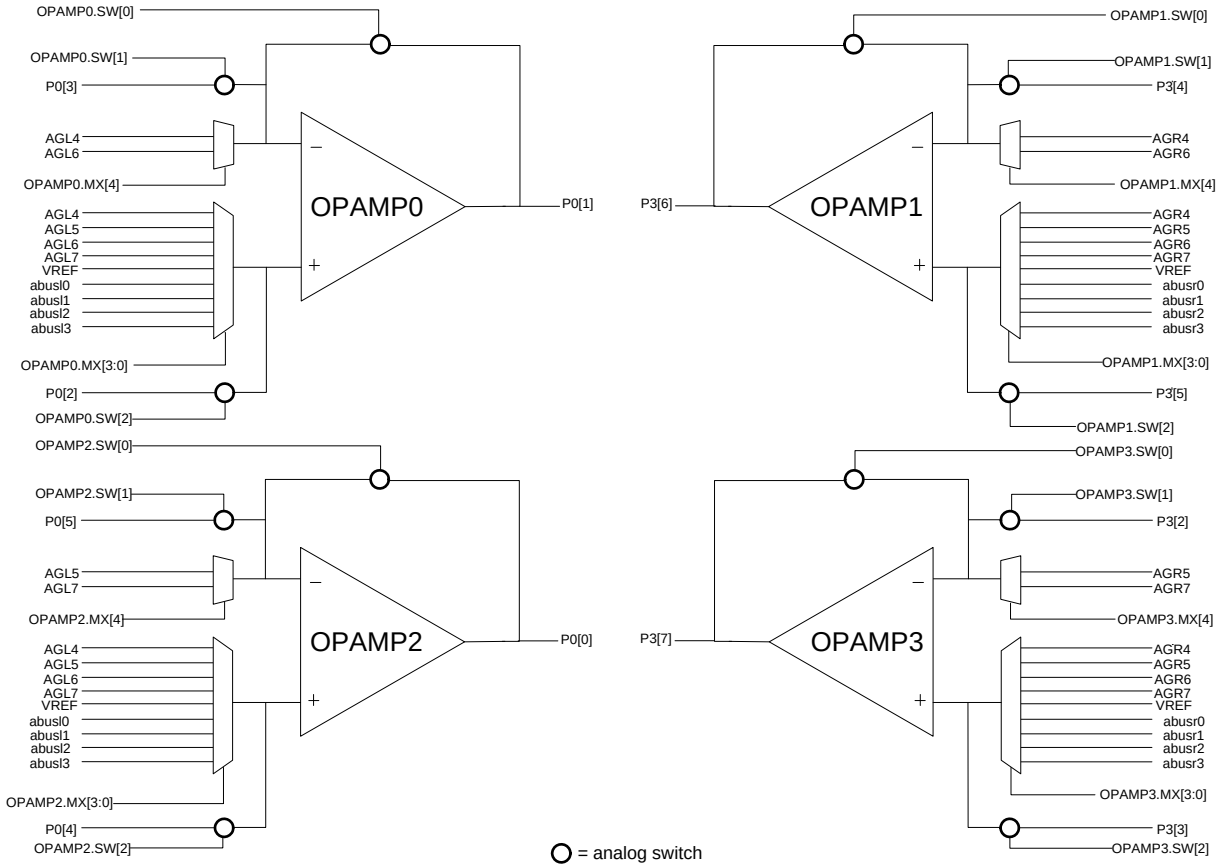
PSoC® operational amplifiers have the following features:

- 10 mA current drive capability
- 2 MHz gain bandwidth for 200 pF load
- Low noise
- Rail-to-rail to within 50 mV of Vss or Vdda for 1 mA load
- Slew rate 3 V/µs for 200 pF load

33.2 Block Diagram

Figure 33-1 is the PSoC operational amplifiers block diagram.

Figure 33-1. Operational Amplifiers Showing Available Connections



33.3 How it Works

PSoC 5 devices have up to four operational amplifiers. The opamps are configurable as a unity gain buffer, to drive high current loads or as an uncommitted opamp. For example, a DAC output or voltage reference can be buffered using an opamp to drive a high current load.

33.3.1 Input and Output Configuration

The positive and negative inputs to the operational amplifier can be selected through muxes and analog switches. A mux is used to connect an analog global, local analog bus, or reference voltage to an input, and an analog switch is used to connect a GPIO to an input. This is shown in Figure 33-1. Inputs are:

- **Positive** – The positive input analog switch, controlled by bit ABUFx_SW[2], is used to select an input from an external pin. The positive input mux (controlled by bits ABUFx_MX[3:0], is used to select an input from an internal signal.
- **Negative** – The negative input analog switch, controlled by bit ABUFx_SW[1], selects an input from an external pin. The negative input mux, controller by bit ABUFx_MX[3:0], selects an input from an internal signal.

The opamp output is connected directly to a fixed port pin. If 1.024 V internal reference is selected as one of the opamp inputs, then opamp power mode should be set to 0x00. Use the API Opamp_SetPower(0x00) to set the proper power mode.

33.3.2 Power Configuration

The opamp can operate in three power modes – low, medium, and high. Power modes are configured using the (PWR_MODE[1:0]) power mode bits in the (OPAMPx_CR

[1:0]) control register. The slew rate and gain bandwidth are maximum in high power mode and minimum in low power mode. Refer to the device datasheet for gain bandwidth and slew rate specifications in various power modes.

33.3.3 Buffer Configuration

The opamp is configured as a unity gain buffer by closing the feedback switch, using the OPAMPx_SW [0] bit. Setting the OPAMPx_SW[0] bit internally connects the output terminal to the negative opamp input.

33.3.4 Register Summary

Table 33-1 summarizes applicable registers.

Table 33-1. Registers

Register	Function
OPAMPx_SW	Controls positive input switch, negative input switch and feedback switch.
OPAMPx_MX	Selects the internal signal for positive and negative input.
OPAMPx_CR	Configures the power mode.

34. LCD Direct Drive

The PSoC® Liquid Crystal Display (LCD) drive system is a highly configurable peripheral that allows the PSoC device to directly drive a broad range of segment LCDs.

34.1 Features

Key features of the PSoC LCD system are:

- LCD panel direct drive
- Type A (standard) and Type B (low power) waveform support
- Wide LCD bias range support (2 V to supply voltage)
- Static, 1/3, 1/4, and 1/5 bias voltage levels
- Internal bias voltage generation
- Up to 62 total common and segment outputs
- Supports up to 16 common glasses (16:1 multiplex ratio)
- Drives up to 736 total segments (16 backplane × 46 front plane)
- 64 levels of software controlled contrast
- Ability to move display data from memory buffer to LCD driver through direct memory access (DMA) without CPU intervention
- Adjustable LCD refresh rate from 10 Hz to 150 Hz
- Ability to invert LCD display for negative image
- Various LCD driver drive modes, allowing power optimization

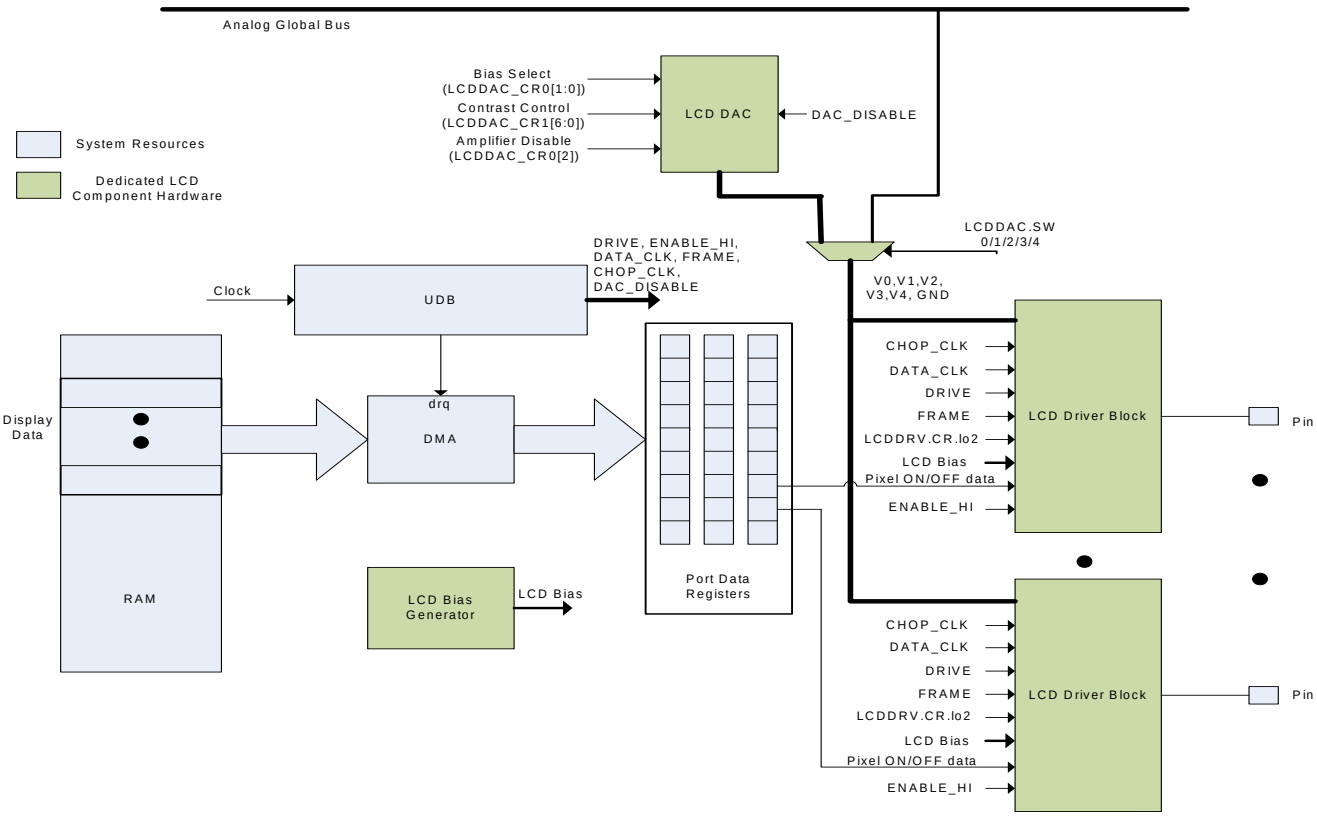
34.2 LCD Drive System

A complete functional LCD drive system is formed using following major blocks

- Dedicated LCD Hardware
  - LCD DAC
  - LCD Driver
- System resources
  - UDB
  - DMA
  - Clocks: Global
  - SRAM

The block diagram is shown in the following figure.

Figure 34-1. LCD Functional Diagram



Any LCD drive system requires the bias generating circuitry and system to interpret the data supplied, to display correctly on the LCD. PSoC 5 contain dedicated LCD drive hardware, which works in conjunction with system resources. It contains a dedicated DAC that generates the five bias voltages, V0 to V4. These bias voltages are distributed to all the drivers of the LCD-capable pins. This DAC also helps to set contrast control.

The LCD driver blocks are the final interface to the pins. Each pin capable of driving an LCD contains driver logic. The function of this block is to select the bias level and drive the pin, depending on the LCD refresh state, whether the pin is configured as common or segment, and the display data. LCD functionality of a pin can be enabled by setting the appropriate bits of the PRT[0..11]_LCD_EN register. The GPIOS can be configured to act as either common or segment drive pins by setting bits of PRT[0..11]_LCD_COM_SEG.

The LCD display data resides in the system memory (SRAM). This display data needs to be transferred to the LCD driver logic. This is done using the direct memory access controller (DMAC). The DMAC takes the display data from the SRAM and loads it into the port data registers. The LCD driver latches this port data register value when a refresh action begins.

Refreshing the LCD requires LCD state updates with accurate timing. This is done using a configurable clock, sourced from the internal main oscillator (IMO), which feeds the UDB block. The UDB is responsible for generating all the control signals required by the remaining blocks of the LCD system.

34.2.1 Functional Description

This section provides details of the LCD DAC, LCD driver, UDBs, and DMA, which all contribute to generating and sequencing the driving voltage for the LCD glass.

34.2.1.1 LCD DAC

The LCD DAC is an independent block that resides in the analog subsystem. It is responsible for contrast control and bias voltage generation for the LCD drive system.

The LCD DAC generates five voltages (V0–V4) that are driven to LCD driver block. Some important points about the are as follows:

- V0 is the highest voltage; the remaining voltage levels (V1–V4) are derived from V0. By default, the five voltages are driven to each of LCD driver blocks.

- Analog mux bus and analog local bus can be selected to drive the LCD driver blocks, instead of the LCD DAC, by setting the appropriate bits of the LCDDAC_SW[0...4] registers. This is useful if you require external dividers to generate the drive voltages and optimize power by switching off the internal DAC. In this mode, there is no software contrast control available.
- When segment LCD is enabled, all Vddio should be greater than or equal to the bias voltage V0.

34.2.1.1.1 Contrast Control

Contrast is controlled by varying the DAC output voltage, V0. This can be done by setting the LCD contrast control register (LCDDAC_CR1[6:0]), which sets the 7-bit DAC input. Thus, it provides 2^6 = 128 levels of contrast. The V0 output ranges from 2.0 V to supply voltage (Vdda) in 25.2 mV steps with a step accuracy of ±3.5 mV and absolute accuracy of ±50 mV.

34.2.1.1.2 Bias Ratio/Multiplex Ratio Selection

Bias ratio/multiplex ratio is selected by setting the bias_sel field of the LCDDAC_CR0 register. This sets the DAC output voltages V1 to V4 as shown in Table 34-1.

Table 34-1. LCD DAC Bias Select

Bias Select Input: lcd_bias_select[1:0]	Multiplex	Bias	V0	V1	V2	V3	V4
b1b0	Ratio		Range in Volt				
11	Invalid – default to 16:1	Default to 1/5	2.0 V to supply	0.800 × V0	0.600 × V0	0.400 × V0	0.200 × V0
10	16:1	1/5	2.0 V to supply	0.800 × V0	0.600 × V0	0.400 × V0	0.200 × V0
01	11:1	1/4	2.0 V to supply	0.750 × V0	0.500 × V0	0.500 × V0	0.250 × V0
00	6:1	1/3	2.0 V to supply	0.666 × V0	0.333 × V0	0.666 × V0	0.333 × V0

34.2.1.2 Configurations Options for DAC

LCD DAC has several configuration registers, which can be used to optimize power. These registers allow the following combinations:

- Active with contrast control

  In this configuration, DAC is fully enabled and V0–V4 voltage levels are active. The DAC block has opamp based buffer, which needs to be active for the contrast control feature. This is done by clearing the amp_disable bit field of the LCDDAC_CR0 register.
- Active without contrast control feature

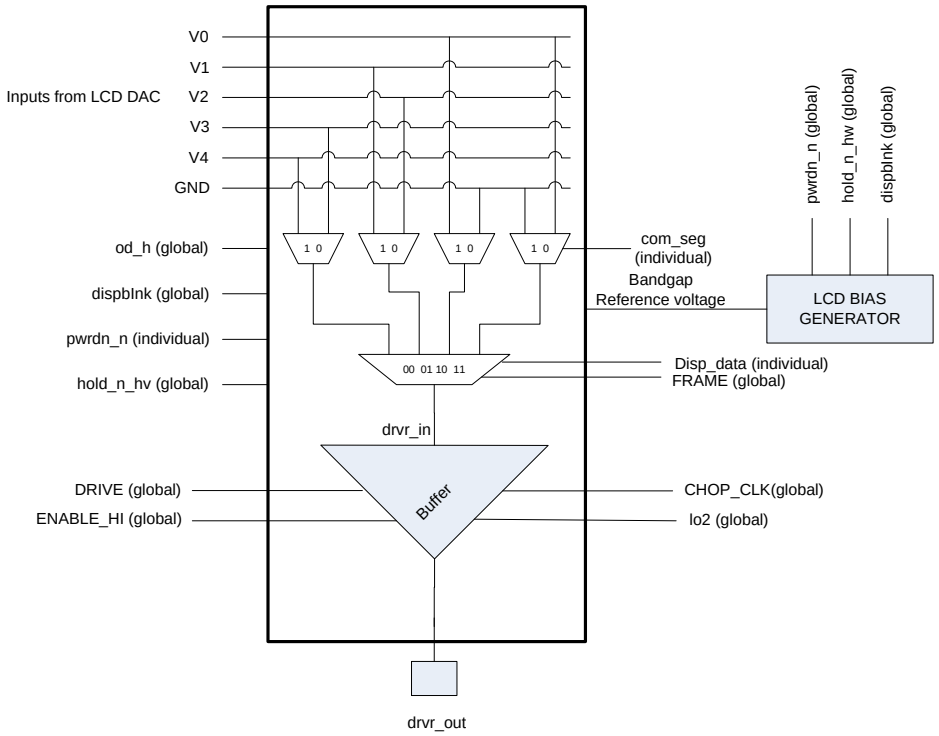
  In this configuration, the opamp inside the DAC block can be disabled to reduce power consumption. Due to this, contrast control feature is not available. The opamp can be disabled by setting the amp_disable bit of the LCDDAC_CR0 register to ‘1’. Note that V0 to V4 bias levels are still available in this configuration.
- LCD DAC disable

  In this configuration, LCD DAC functionality is disabled; its outputs V0–V4 are tri-state. LCD DAC can be disabled by activating the DAC_DISABLE signal generated by the UDB and de-asserting en_lcd bit in the PM_STBY_CF5 register. This configuration can be used in low power LCD applications in which V0–V4 are enabled, driven for a short period of time, and then disabled.

34.2.1.3 LCD Driver Block

The LCD driver block is associated with each GPIO. The output of LCD DAC through MUX is provided to the LCD driver block to drive the LCD glass. The architecture of the LCD driver block is shown in Figure 34-2.

Figure 34-2. LCD Driver Block



The LCD driver contains three major blocks:

- Buffer and associated control logic for power modes
- 4:1 Output multiplexer
- Common/Segment switches

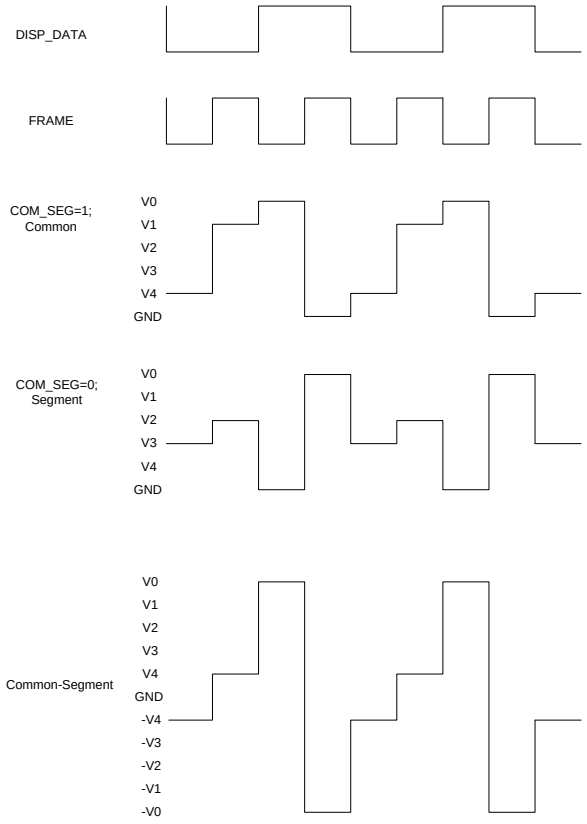
As shown in Figure 34-2 on page 300, the LCD driver block receives bias voltages V0 to V4 and GND voltage. It passes through a set of 2:1 muxes controlled by the COM-SEG bit of the PRT[x]_LCD_COM_SEG register. This register configures the pin as either a common or segment drive pin. If the bit is set, it configures the corresponding pin as common; otherwise, it is configured as a segment drive pin. As shown in Figure 34-2, if the pin is selected as common, only V0, V1, V4, and GND voltages are forwarded to the next mux. If the pin is selected as a segment line, then V0, V2, V3, and GND are forwarded. These are the only voltages required at common and segment lines for any bias ratio, multiplex ratio, and LCD update state. Out of these four bias levels, only one level is selected by the 4:1 multiplexer. The select lines of the multiplexer are driven by display data and the frame signal. Frame is a global signal driven by the UDB control logic. This signal toggles every time the LCD waveform needs to be updated. Table 34-2 shows the 4:1 multiplexer output and driver input for different combinations of COM_SEG, DISP_DATA and the frame signal.

Table 34-2. LCD DAC Output Selection

com_seg	disp_data	frame	drvr_in/out
0	0	0	V3
0	0	1	V2
0	1	0	GND
0	1	1	V0
1	0	0	V4
1	0	1	V1
1	1	0	V0
1	1	1	GND

The following figure shows the waveforms generated by the LCD driver block.

Figure 34-3. Output Waveforms from LCD Driver Block



34.2.1.3.1 Driver Buffer Modes

The output of the 4:1 multiplexer is driven to the buffer, which drives the common or segment line of the LCD. The buffer in the LCD drive block has three power modes:

- High Drive
- Low Drive
- Lo2 Drive

High drive mode is activated when the DRIVE signal and ENABLE_HI signal are set high. Low drive mode is activated when the DRIVE signal is held high and ENABLE_HI is held low. DRIVE and ENABLE_HI signals are generated by UDB; see UDB on page 302. High drive mode charges the LCD pixel with high current. But this mode is not very accurate. Low drive mode is accurate and low power. This mode is used to reduce the offset error and to compensate for the leakages from the LCD display. If the leakage is excessive, a second low drive mode, Lo2, provides twice the output current as compared to low drive. Lo2 drive mode is enabled by setting the lo2 bit field of the LCDDRV_CR register.

The following table summarizes the conditions in which a drive mode is activated:

Table 3: Buffer modes

Buffer Mode	DRIVE	ENABLE_HI
High drive	HIGH	HIGH
Low drive	HIGH	LOW
Lo2	HIGH	LOW (with LCDDRV.CR configured)

The two low drive modes have the option of using a 100 kHz CHOP_CLK to compensate for device offset. Device offset refers to voltage across a pixel not getting averaged to 0-V DC over a frame for Type A and over two frames for Type B waveforms. The 100 kHz clock is derived from the UDBs. The clock flips the polarity of the driver block output, which in turn switches the polarity of any offset at the output. The LCD displays cannot respond to frequencies greater than 10 kHz; therefore, the offset at the output is averaged out when the chop clock is used.

The time for which the LCD driver operates on these modes are configurable. It is decided based on the size and capacitance of the LCD display and the power budget of the application. The device consumes more current when the LCD driver operates in high drive mode when compared to the other two low drive modes.

34.2.1.3.2 LCD Driver Bias Generator

The LCD bias generator block creates a bandgap-based voltage reference for the LCD driver block. The input to this block is a 2.5-µA bandgap current. The output is a bias voltage and the associated ground line.

34.2.1.3.3 Configuration Options for LCD Drive Buffer Block

Similar to LCD DAC, the LCD drive buffer also has several configuration options to optimize power. Besides the high drive and two low drive modes mentioned earlier, the other configuration options available are shown in Table 34-3.

Table 34-3. Configuration Options for LCD Drive Buffer

Configuration	Control Signals					
	ENABLE_HI (Generated by UDB)	DRIVE (Generated by UDB)	lo2 (LCDDRV_CR register)	DispbInk (LCDDRV_CR register)	pwrdn_n (PRTx_LCD_EN register)	od_h (Global signal from power section)
High Drive	1	1	X	0	1	0
Low Drive	0	1	0	0	1	0
Low Drive with increased current (lo2)	0	1	1	0	1	0
No Drive	X	0	X	0	1	0
LCD Sleep	X	X	X	1	1	0
LCD Disable	X	X	X	X	0	0

The control signals ENABLE_HI, DRIVE, lo2, dispbInk, pwrdn_n, and od_h decide the mode in which the LCD driver block operates. The ENABLE_HI and DRIVE signals are generated by UDB. Lo2, dispbInk, and pwrdn_n are from registers. Od_h is a global signal from the power section, which keeps all the drivers in tri-state during power-up sequence of the chip.

No Drive

In this configuration, drive buffers are disabled and outputs are tri-state; but the power supply to the block is still present.

LCD sleep

In this configuration, power supply to the block is disconnected and all LCD pins are pulled to ground. This configuration should not be used for non-LCD applications.

LCD Disable

This is the power down mode for non-LCD applications. In this mode, power supply to the block is disconnected. Pins are in high impedance state. All unused LCD drivers should be kept in this mode to reduce power consumption.

34.2.1.4 UDB

The UDB generates various control signals for the functioning of the LCD system hardware.

DRIVE: This signal along with *ENABLE_HI* decides the drive mode of the buffer as given in table 4.

ENABLE _HI: This signal enables high drive of buffer. It is also used to trigger DMA request to move the next set of pixel data from SRAM to port data registers.

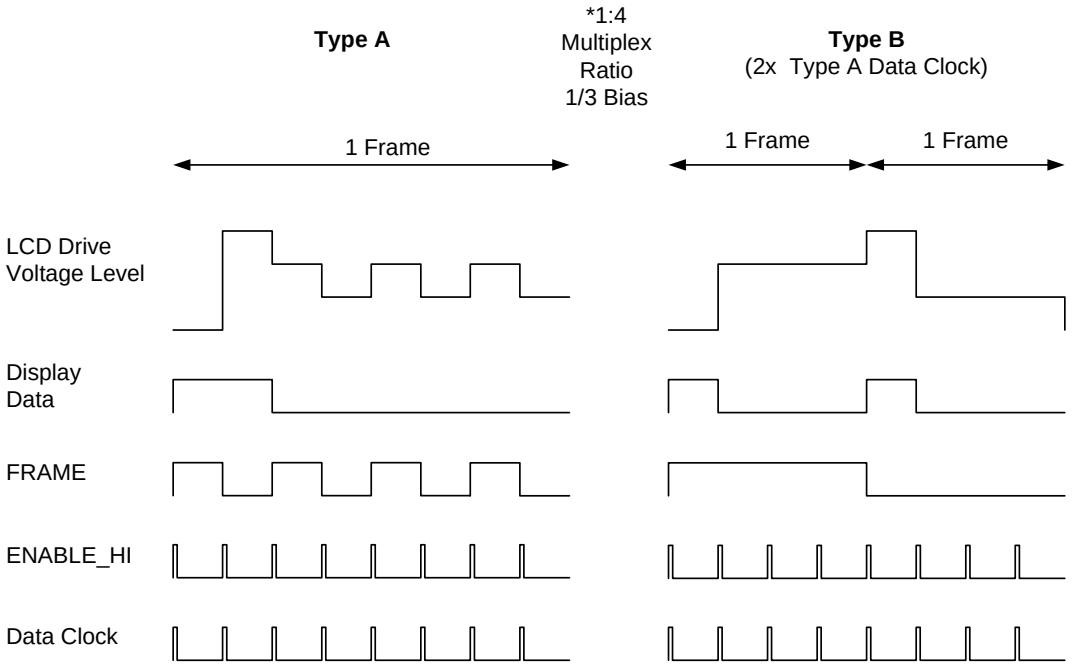
FRAME: This signal is a square waveform used in bias voltage selection depending on the state of the LCD waveform.

DATA_CLK: This is a pulse waveform used to clock in next set of display data from port data register to LCD driver.

CHOP_CLK: This is a 100 kHz clock used as an option during low drive mode to cancel out offset error.

The following figure shows the different signals generated by UDB.

Figure 34-4. Different Signals from UDB



The periodicity of all these signals depends on the refresh rate of the LCD, type A or B waveform, and multiplex ratio.

34.2.1.5 DMA

DMA is used to transfer the display data into various port data registers. The display data is stored in SRAM. Data transfer is initiated by the UDB at the beginning of the LCD refresh cycle. Depending on which and how many ports are configured for the LCD drive, several transaction descriptors (TDs) associated with the DMA channel may need to be chained together.

There is no separate display memory, as such, in PSoC. Display data resides in the SRAM connected to the peripheral hub (PHUB). The image/display buffer can be any block of available memory.

To work more effectively with the DMA in transferring data to the LCD drivers, port data registers are aliased to a separate contiguous region in the memory map. These PRTx_DR_ALIAS registers are contiguous, to reduce the number of TDs required to move data.

An additional set of registers (per port), the PRTx_BIT_MASK registers, mask off the write capability to the PRTx_DR_ALIAS registers on a bit level. This is an advantage if all of the pins on a given port are not being used for the LCD; the unused pins can be masked off and used for other purposes. The port data register (PRTx_DR) can still be used to address pins masked off in the aliased data registers.

34.3 Low Power Mode Behavior (Sleep/Hibernate)

In PSoC 5, the LCD driver hardware does not function when the device is put in low power modes. The status of the LCD enabled pins during low power modes depends on sleep_mode bit of LCDDRV_CR0 register. If this bit is set to ‘0’, the pins are grounded, thus blanking the LCD display. If the bit is set to ‘1’, the pins are tri-stated and signal driven to the LCD display remains for some time until the charge leaks off.

34.4 LCD Register Summary

Register Name	Description	Bit Fields
LCDDAC_CR0	LCD DAC configuration register	Amplifier disable, bias select
LCDDAC_CR1	LCD DAC configuration register	Contrast control
LCDDRV_CR0	LCD driver configuration register	Invert, lo2, sleep_mode
PRT[0..11]LCD_COM_SEG	LCD pin configuration register	Common or segment pin setting
PRT[0..11]LCD_EN	LCD function enable register	One bit for each pin to enable LCD function

35. CapSense®

PSoC® 5 devices have a capacitive sensing feature called CapSense®. This feature allows users to take advantage of the capacitive properties of their fingers to toggle aesthetically superior buttons, sliders, and wheels. Touch pads and touch screens are common examples of capacitive sensing interfaces. The underlying principle of these technologies is the measurement of capacitance between a plate (the sensor) and its environment.

35.1 Features

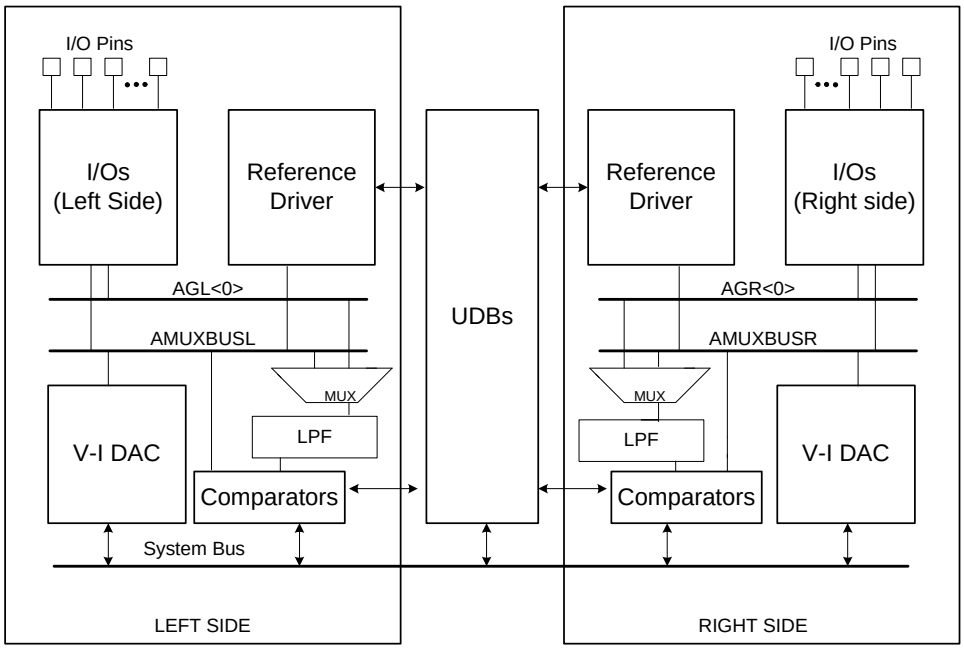
Features of CapSense include:

- Resources to support two capacitive sensors scanning simultaneously
- Configurable low pass filter to remove switching noise for accurate measurement
- Reference buffer with High Drive mode for faster measurement

35.2 Block Diagram

A block diagram of the overall capacitive sensing architecture is shown in Figure 35-1.

Figure 35-1. CapSense Module Block Diagram



35.3 How It Works

The PSoC device has configurable hardware for CapSense to optimize factors such as speed, power, sensitivity, noise immunity and resource usage. It implements CapSense Sigma Delta (CSD) method of capacitive sensing.

35.3.1 Reference Driver

This driver is used to quickly initialize nets to a voltage independent of the power supply. This ability speeds up capacitive scanning and improves Power Supply Rejection Ratio (PSRR). Two reference drivers operate independently; one drives to AMUXBUSL, and one for AMUXBUSR. The driver is connected to the AMUXBUS by setting the out_en bit in the {CAPSx_CFG0}.

The reference driver supports Normal and High drive modes; the drive mode is selected using the boost bit in the {CAPSx_CFG0} register. In Normal mode, capacitances up to 100 pF can be driven in less than 600 ns. In High mode, capacitances up to 30 nF can be driven in less than 15 µs.

35.3.2 Low Pass Filter

Two tunable Low Pass Filter (LPF) blocks are available. The inputs are selectable in a 2:1 mux for each LPF. On the left side, the LPF inputs are AMUXBUSL and AGL[0]; on the right side, the inputs are AMUXBUSR and AGR[0]. LPF input is selected by using the swin[1:0] bits in the LPFx.CR0 register. The outputs are connected through switches to abusl[0] and abusr[0], respectively. The tunability of the LPF allows the user to select a (nominal) R of either 200 kΩ or 1000 kΩ, and a C of either 5 pF or 10 pF. The rsel and csel bits in the LPFx_CR0 register are used to select resistance and capacitance respectively. The LPF control registers are LPF0_CR0 and LPF1_CR0.

35.3.3 Analog Mux Bus

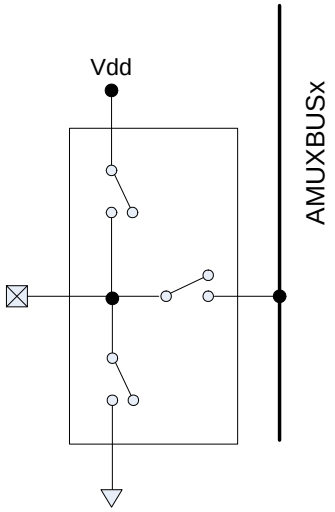
All GPIO pins support CapSense operations except SIO and USB pins. The primary analog mux bus for CapSense is the AMUXBUS, which has two nets (AMUXBUSL and AMUXBUSR) for two simultaneous sensing operations. These can also be shorted to form a single net that connects to all GPIO. Refer to the device datasheet for details about GPIOs available in each package and to the Analog Routing chapter on page 273 for a diagram of AMUXBUS connectivity for the GPIO.

AMUXBUSL and AMUXBUSR nets connect to all GPIO pins on their respective halves of the device. CapSense uses the AMUXBUS net, along with an analog global net (AGR[0] with AMUXBUSR, and AGL[0] with AMUXBUSL) to provide feedback to the reference driver. This feedback is from a pin connected to a large off-chip capacitor serving as integration or modulation capacitor.

35.3.4 GPIO Configuration for CapSense

The GPIO switching structure supporting CapSense is shown in Figure 35-2.

Figure 35-2. GPIO Structure



The port analog global mux register (PRT[x]_AMUX) is used to connect the port pin to the analog mux bus. The pull up or pull down is enabled using io_ctrl[1:0] bits in the CAPSx_CFG1 register.

Sense capacitance is switched in two configurations, shown in Figure 35-3 on page 307 and Figure 35-4 on page 307, to convert the capacitance into equivalent resistance for measurement.

The equivalent resistance can be calculated as:

$$R_s = \frac{1}{(f_s C_s)}$$

Here:

C_S=Sensor Capacitance

$\phi 1$ and $\phi 2$ = Non-overlapping clocks, which may be configured in a pseudo random sequence (PRS).

f_S = Frequency of the clock

C_{mod} = External Modulation Capacitance

The CapSense methods can generally be done with either switching high or switching low at the GPIO pin. The rest of the hardware is configured with the appropriate polarity to match to the pull up or pull down choice.

Figure 35-3. Charging MUXBUS Through Sense Resistor

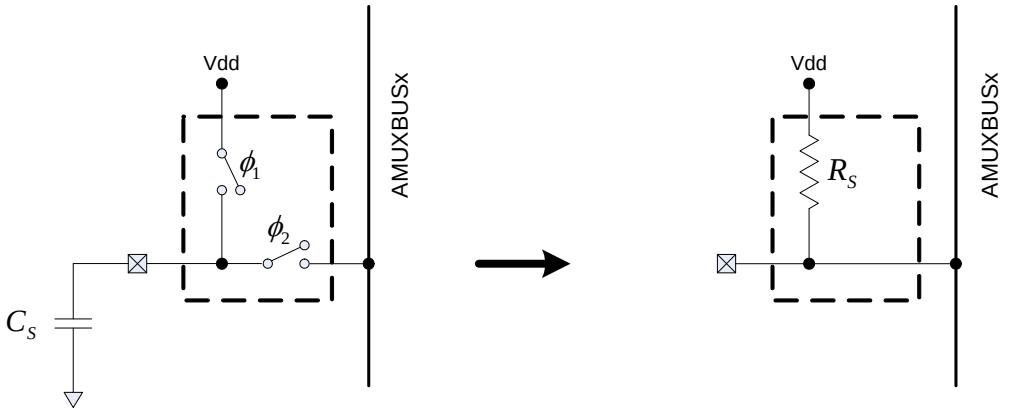
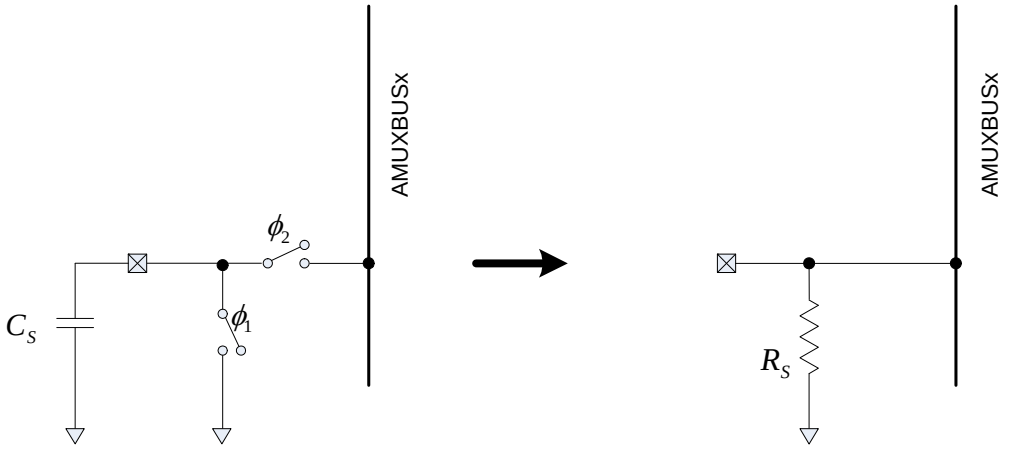


Figure 35-4. Discharging the MUXBUS Through Sense Resistor



The CapSense clock is used for switching. Two alternatives are available to generate the CapSense clock (refer also to Figure 21-1 on page 126).

- The UDB generates two global clocks (caps_dsi_lft and caps_dsi_rt), and routes to GPIO logic of the I/O pins in the respective side. The PRT[x]_CAPS_SEL[y] registers (per port per pin basis) are set to select the global clock for switching the sensor during measurement.
- The DSI output to the I/O pin can be used to source the CapSense clock from the UDB. The PRTx_BIE[y] must be programmed for input (per port per pin basis) and PRT[x]_CAPS_SEL[y] is cleared to select the DSI output signal for the CapSense clock.

With either of these paths, the nonoverlapping clock phases discussed above are automatically generated within the GPIO switching structure.

Note that to connect an external integration capacitance (C_{mod}) statically (without switching), connect it to AMUXBUS using PRT[x]_AMUX register and then PRTx_CAPS_SEL[y] = 0 and PRTx_BIE[y] = 0.

GPIOs pins can be made as Shield Electrodes. The shield electrodes help in reliable operation in presence of water film or water droplets. The effect of these factors on shield electrode is measured and is removed from the CapSense Buttons. The CapSense algorithms discussed below support the shield electrode.

35.3.5 Other Resources

CSD CapSense techniques use many resources in PSoC 5 devices. These include UDBs, Comparators, and V-I DAC. See the Universal Digital Blocks (UDBs) chapter on page 149, Comparators chapter on page 289, and Digital-to-Analog Converter chapter on page 311 for more detail on those.

35.4 CapSense Delta Sigma Algorithm

The CapSense Delta Sigma (CSD) algorithm shown in Figure 35-5 and Figure 35-6 on page 309 measures capacitance with the hardware configured similar to a Delta Sigma modulator. Delta Sigma capacitive sensing operates by holding an integration capacitor voltage near a target threshold, and charging or discharging the capacitor, based on the present state of a comparator output. The sense capacitor is continuously switched between V_{dd} and the integration capacitor, which drives the integrated voltage up on each switching cycle. The CSD algorithm operates as follows:

1. When the integration voltage reaches the reference voltage, the comparator enables current DAC to discharge the capacitor.
2. When the capacitor voltage discharges below the reference voltage, the current DAC is disabled to allow the capacitor to continue charging.
3. As the integration capacitor voltage moves back and forth across the comparator threshold, the comparator high outputs are counted in an interval to give a measure of the sense capacitor.
4. The sense capacitance increases with touch, therefore equivalent resistance decreases. This decreased resistance causes an increase in the current flowing through switch CapSense resistor.
5. To maintain the voltage on C_{mod} near V_{REF} during a touch, the IDAC sinks current for longer duration to compensate for the larger sense capacitance. This changes the count value accordingly.

A PRS (pseudo random sequence) clock may be used instead of a fixed clock source to drive the precharge switches. The PRS clock produces less radiated noise on the sense capacitor, compared to a fixed clock source, hence improving EMI and interference performance.

Figure 35-5. CSD Hardware Configuration

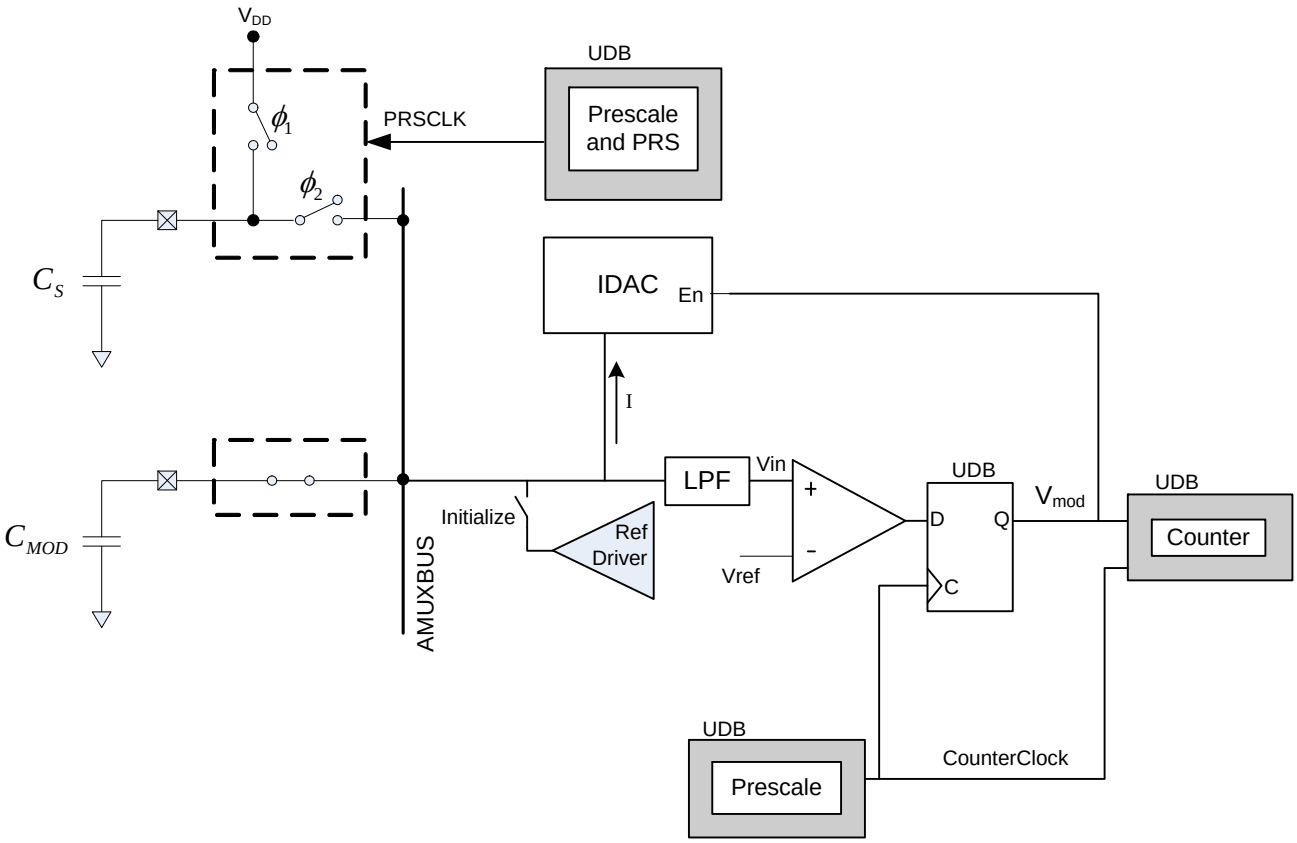
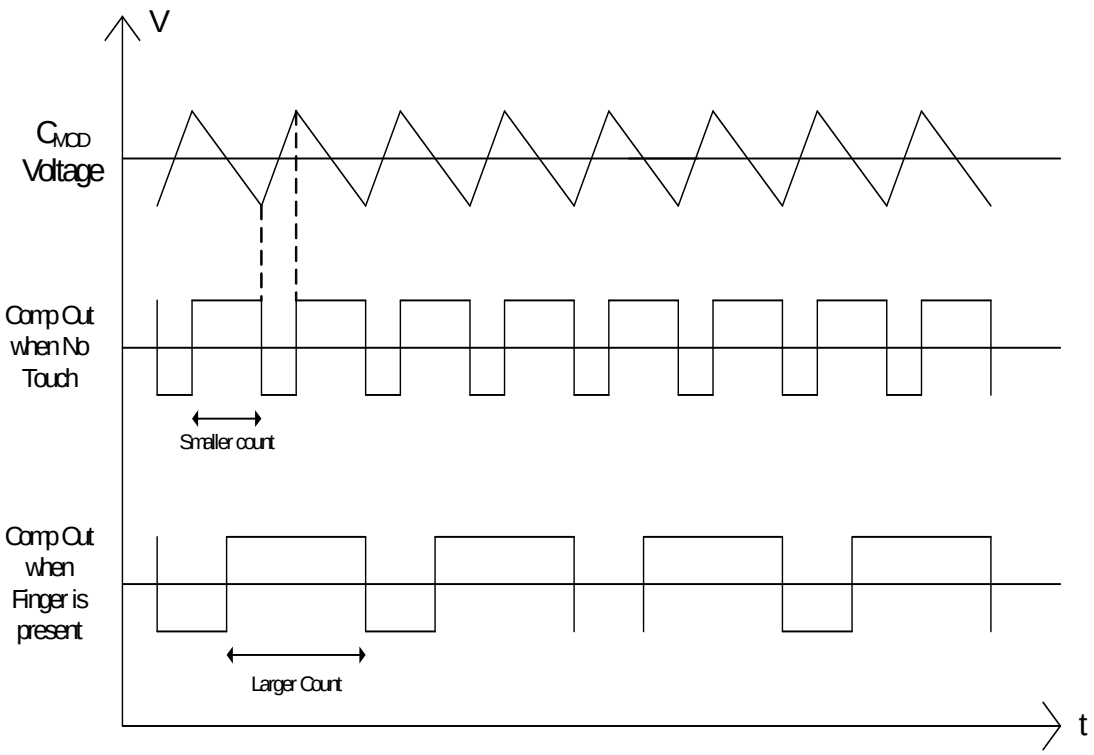


Figure 35-6. CSD Waveform



The PSoC device also supports other variants of the CSD algorithm as follows:

- Switched Capacitor Resistor (see Figure 35-3 on page 307) is used to charge the integration capacitor; an external bleeding resistor is used (instead of IDAC) to discharge the integration capacitor, based on comparator output.
- Polarities are reversed so that the IDAC is used to charge up the integration capacitor and Switched Capacitor Resistor (see Figure 35-4 on page 307) discharges the integration capacitor toward ground, based on comparator output.

36. Digital-to-Analog Converter

The digital-to-analog converter (DAC) is an 8-bit converter that is configured to output either a voltage or a current. The 8-bit DAC supports CapSense®, power supply regulation, and waveform generation.

36.1 Features

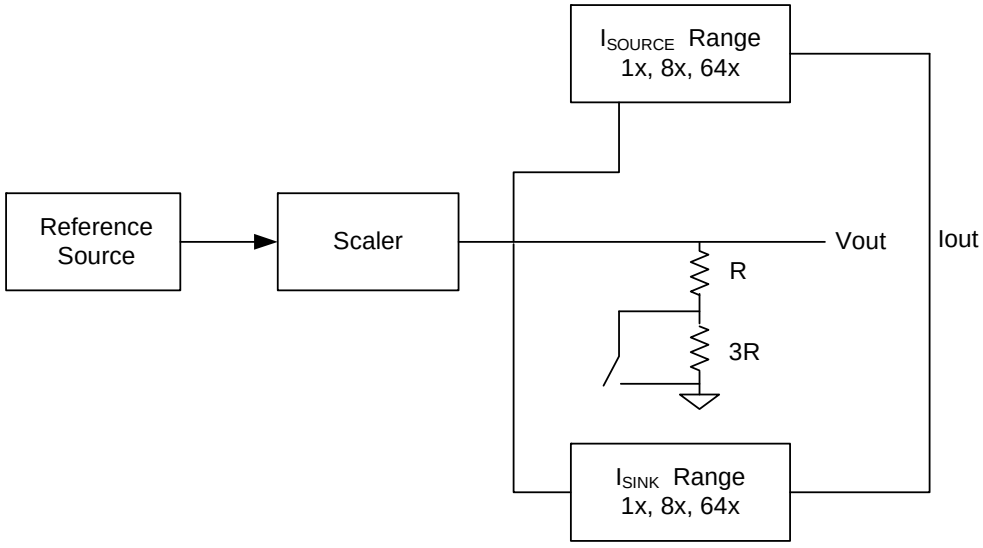
The DAC has the following features:

- Adjustable voltage or current output in 255 steps
- Programmable step size (range selection)
- Eight bits of calibration to correct ± 25% of gain error
- Source/sink option for current output
- Output rate for current IDAC output: 5.5 Msps
- Output rate for VDAC voltage output: 1 Msps
- Monotonic in nature

36.2 Block Diagram

A block diagram of the DAC is shown in Figure 36-1.

Figure 36-1. DAC Block Diagram



36.3 How It Works

This DAC generates either a voltage or a current output. It is built using current mirror architecture; current is mirrored from a reference source to a mirror DAC. Calibration and value current mirrors are responsible for the 8-bit calibration [DACx_TR] and the 8-bit DAC value. The current is then diverted into the scaler to generate the current corresponding to the DAC value. The DAC value can either be given from the register DACx_D or from 8 lines from the UDB. This selection is made using the DACx_CR1[5] bit. Using the UDB to write the DAC value uses the DAC bus. Because there is only one DAC bus available for each device, this bus must be shared by all the DACs in the device.

The DAC is strobed to get its output to change for the input code. The strobe control is enabled by the DACx_STROBE[3] bit. The strobe sources for the DAC can be selected from the bus write strobe and analog clock strobe to any UDB signal strobe. This selection is done on the basis of setting in DACx_STROBE[2:0]. Two successive writes of the new value to the VIDAC data register are required to change the DAC output from the current value to the new value. If an interrupt occurs between the two writes, the DAC output may go to an indeterminate state until the second write occurs. The VIDAC data register may be written from either the CPU or the DMA. To change the DAC output without an intermediate value occurring, do the following:

- CPU Writes: Set the PRIMASK bit immediately before the first write instruction and clear it immediately after the second write instruction. PRIMASK changes the interrupt priority so that only NMI and hardware exceptions can be serviced. Any interrupts occurring while PRIMASK is set will be kept pending and serviced after PRIMASK is cleared. Refer to section 4.2.1.1 Special Registers on page 38 and the Interrupt Controller chapter on page 67 for information on the PRIMASK bit.
- DMA Writes: DMA transfers can be pre-empted by higher priority DMA channels. To ensure that two writes occur in succession, you should select the highest priority for the DMA transfers to the VIDAC.

The VDAC output gives a small glitch of approximately 100 ns on every DAC update.

- **Current (IDAC) Mode** – The two mirrors for the current source and sink provide output as a current source or current sink, respectively. These mirrors also provide range options in the current mode.
- **Voltage (VDAC) Mode** – The current is routed through resistors according to the range and voltage across it provided as output.

The output from the DAC is single-ended in both IDAC and VDAC modes.

36.3.1 Current DAC

When used as an IDAC, the output is an 8-bit digital-to-analog conversion current. This is done by setting the DACx_CR0 [4] register. The reference source is a current reference from the analog reference called IREF(DAC). In this mode, there are three output ranges selected by register DACx_CR0 [3:2].

- 0 to 2.048 mA, 8 µA/bit
- 0 to 256 µA, 1 µA/bit
- 0 to 32 µA, 0.125 µA/bit

For each level, there are 255 equal steps of M/256 where M = 2.048 mA, 256 µA, or 32 µA. In the 2.040 mA configuration, the block is intended to output a current into an external 600 Ω load.

The IDAC is capable of converting up to 8 Msps. The user also has the option of selecting if the output is a current source or a sink. This is done by the DACx_CR1[2] register. The selection between source and sink for the IDAC can also be done using a UDB input. UDB control for the source-sink selection is enabled using the DACx_CR1[3] bit.

36.3.2 Voltage DAC

When used as a VDAC, the output is an 8-bit digital-to-analog conversion voltage to support applications where reference voltages are needed. Here the reference source is a voltage reference from the Analog reference block called VREF(DAC). The DAC can be configured to work in voltage mode by setting the DACx_CR0 [4] register. In this mode, there are two output ranges selected by register DACx_CR0[3:2].

- 0 V to 1.024 V
- 0 V to 4.096 V

Both output ranges have 255 equal steps.

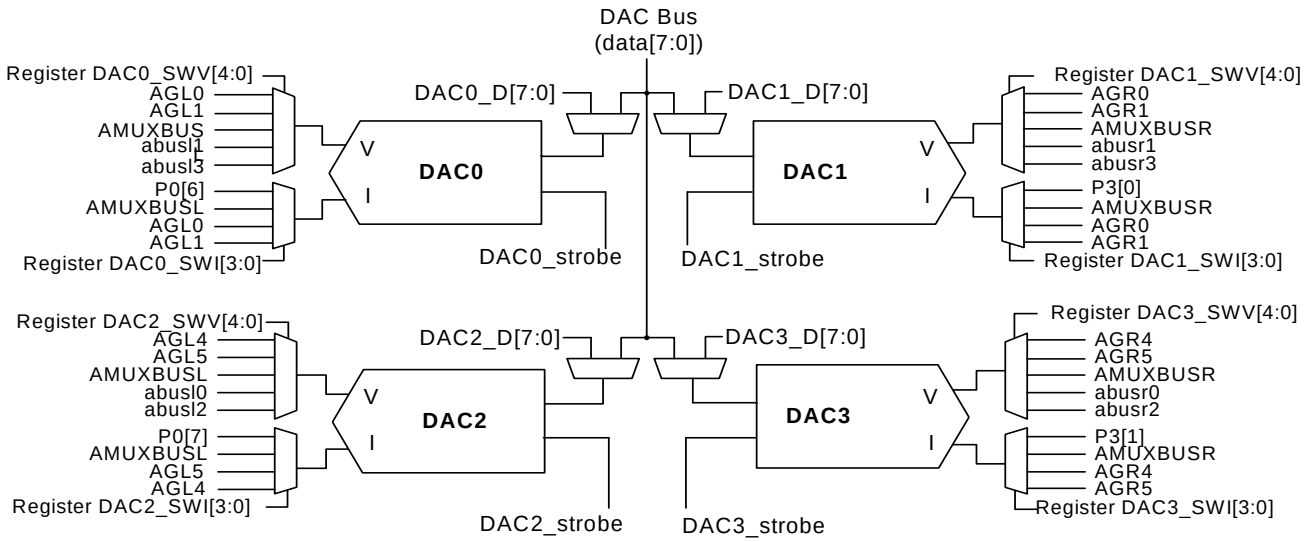
The VDAC is implemented by driving the output of the current DAC through resistors and obtaining a voltage output. Because no buffer is used, any DC current drawn from the DAC affects the output level. Therefore, in this mode any load connected to the output should be capacitive.

The VDAC is capable of converting up to 1 Msps. In addition, the DAC is slower in 4 V mode than 1 V mode, because the resistive load to V_{ssa} is 4 times larger. In 4 V mode, the VDAC is capable of converting up to 250 ksps.

36.3.3 Output Routing Options

Output routing options for the DAC are attained through two separate muxes for current and voltage modes. These muxes are controlled by the DACx_SWx registers, as shown in Figure 36-2 on page 313. VDAC0's output does not go beyond 2.4 V. Therefore, if you need a DAC output of greater than 4 V, do not use VDAC0.

Figure 36-2. DAC Interconnect



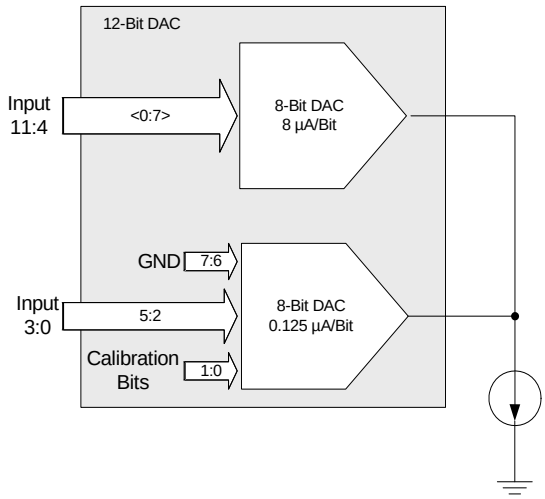
The user can route output as follows:

- **Voltage Mode** – to the analog globals, analog mux bus, or the analog local bus
- **Current Mode** – to the analog globals, analog mux bus, or to a specific port

36.3.4 Making a Higher Resolution DAC

It is possible to achieve a higher resolution current output DAC by summing the outputs of two 8-bit current DACs, each one having a different segment of the input bus for input. The range of the two DACs used partially overlap.

Figure 36-3. Higher Resolution DAC Example

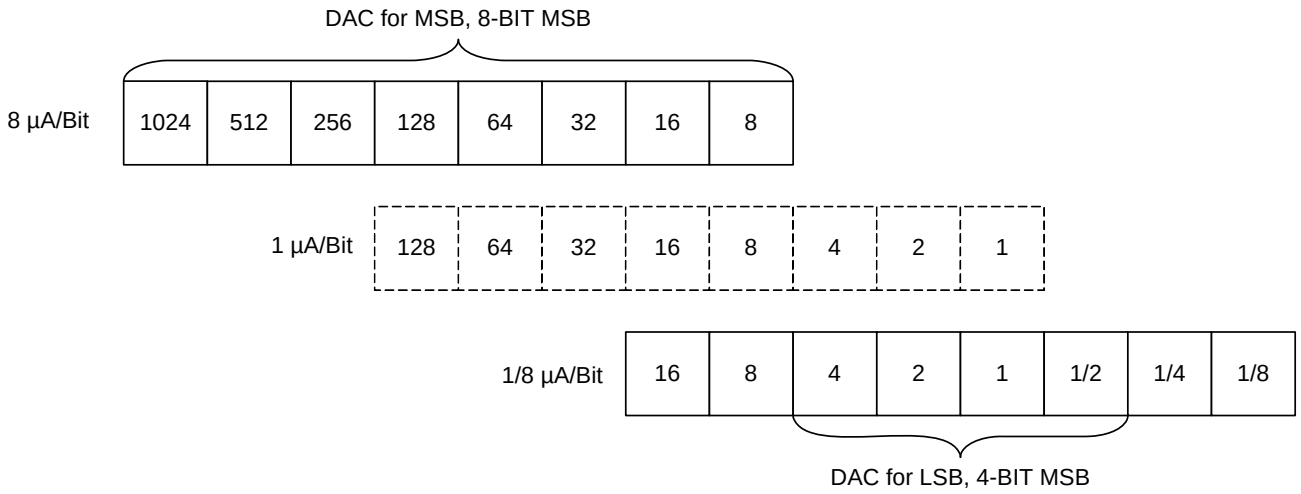


For example, the implementation of a 12-bit DAC using two 8-bit DACs require:

- One DAC scaled to the range 0 to 2.048 mA and the second one scaled to the range 0 to 32 µA.
- The middle 4 bits of the lowest range DAC are used as inputs to the lower 4 bits. See Figure 36-4 on page 314.

This architecture may have problems of mismatch in the two DACs and therefore might require adjustment and scaling. The last two bits of the LSB DAC are used for minor calibration requirements.

Figure 36-4. 12-Bit DAC Using Two 8-Bit DACs Example



36.4 Register List

Table 36-1. DAC Register List

Register Name	Comments	Features
General Registers		
DACx_CR0	DAC Control register 0	Select DAC mode, range, and speed
DACx_CR1	DAC Control register 1	Control DAC data source, reset, and direction
DACx_SW0	DAC Analog routing register 0	Routing for the DAC voltage output to analog (global) bus
DACx_SW2	DAC Analog routing register 2	Routing for the DAC voltage output to analog (local) bus
DACx_SW3	DAC Analog routing register 3	Routing for the DAC current/voltage output to AMUXBUS
DACx_STROBE	DAC Strobe register	DC strobe control
DACx_D	DAC Data register	
DACx_TR	DAC Block Trim register	DAC trim values

37. Precision Reference

A voltage/current reference with value independent of supply voltage and temperature is an essential building block of many analog circuits. For example, accurate biasing voltages are critical for many circuit schemes. In an ADC, a reference voltage is required to quantify an input. In a VIDAC, the voltage/current reference is required to define the output full-scale range.

37.1 Block Diagram

The PSoC® 5 devices have a curvature compensated voltage bandgap along with a trim buffer to get absolute value accuracy. The trim buffer is a multiple reference generator. It takes the bandgap reference voltage as input and produces outputs ranging from 0.256 V to 1.2 V. The various reference voltages are buffered via high accuracy/low power (5 µA) buffers. There is also a temperature corrected current reference that is sent to the current DACs (IDACs).

The precision reference contains four top level blocks:

- Voltage and current output bandgap
- Trim buffer for voltage bandgap
- Low power (5 µA) buffer for reference voltages
- Bias block for 5 µA buffer

37.2 How It Works

The principle of the bandgap circuit relies on two groups of diode-connected bipolar junction transistors running at different emitter current densities. By canceling the negative temperature dependence of the PN junctions in one group of transistors with the positive temperature dependence from a Proportional-to-Absolute-Temperature (PTAT) circuit (which includes the other group of transistors), a fixed DC voltage that does not change with temperature is generated.

Figure 37-1. Reference Block Diagram

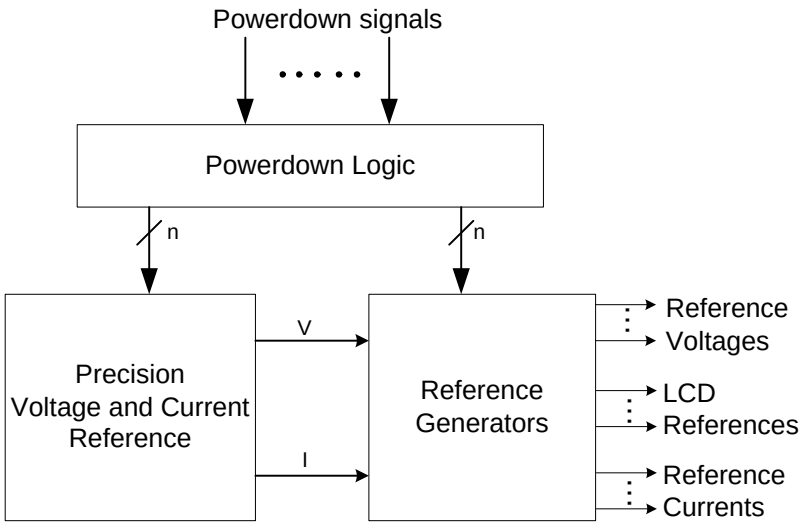
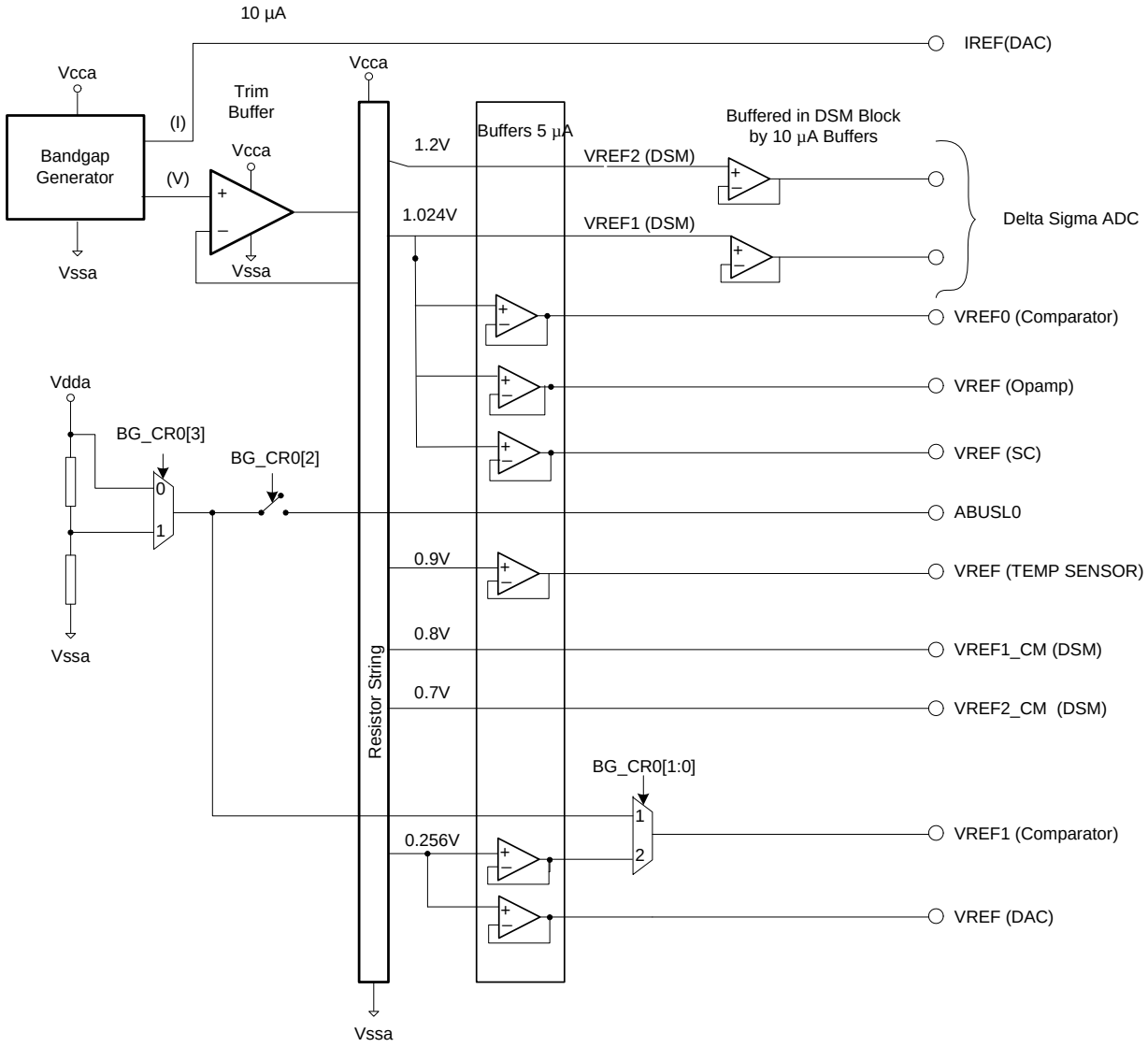


Figure 37-2. Voltage Reference Block Diagram



Dynamic enabling or disabling of analog peripherals may disturb shared internal voltage references. This may parametrically or functionally affect "already-on" or "staying on" components. Because these interactions are at the system level, they cannot always be systematically addressed in firmware component design. Therefore, consider the implications of a disturbed reference on components in use when dynamically enabling or disabling analog components. This includes enabling or disabling analog components as part of system power mode transitions, whether hardware or firmware based.

Note 1 Analog supply Vdda or Vdda/2 can be routed to the analog blocks through the analog local bus, ABUSL0. The voltage level is selected using the BG_CR0[3] bit and the switch is enabled using the BG_CR0[2] bit.

Note 2 Reference voltage input (VREF1) to the comparator is selected using the BG_CR0[1:0] bits. It selects either bandgap reference voltage or the analog supply voltage.

Note 3 IREF (DAC) is the reference current for the DAC during IDAC mode operation.

37.2.1 Precision Voltage and Current Reference

The precision V/I reference outputs a 1-V (nominal) voltage reference and 2.5-µA (nominal) current reference into the reference generator block as shown in Figure 37-1 on page 315. The precision bandgap voltage is designed to provide 20-ppm accuracy. There are seven trim bits plus one fine trim bit to optimize the voltage for variations in temperature and supply. A trim buffer is used to correct the absolute voltage accuracy and has nine trim bits plus one fine trim bit. The precision bandgap current supplies reference currents to analog blocks and has six trim bits to trim the absolute value and six trim bits for temperature coefficient (tempco). There are an additional seven bits for crossover/INL control.

37.2.2 Reference Generators

The reference generator block is comprised of several different blocks. The trim buffer is a multiple reference generator. It takes the bandgap reference voltage as input and produces outputs ranging from 0.256 V to 1.2 V. The 5-µA buffer blocks are used to buffer the trim buffer outputs from the analog blocks that use the reference. The trim buffer bias is the bias block for all the buffers used. Table 37-1 lists the reference voltages and blocks that use them.

Table 37-1. Reference Voltages and Blocks

Voltage	Block	Value	Description
VREF0 (Comparator)	Comparator	1.024 V	To Comparator negative inputs
VREF1 (Comparator)	Comparator	Vdda (or) Vdda/2 (or) 256 mV	To Comparator negative inputs
VREF (Opamp)	Opamp	1.024 V	To Opamp positive inputs
VREF (SC/CT)	SC/CT Block	1.024 V	To SC/CT block positive and negative inputs
ABUSL0	Comparator Opamp DAC SC/CT DSM	Vdda (or) Vdda/2	All blocks connected to the analog local bus ABUSL0 can get this voltage
VREF (DAC)	DAC	256 mV	Reference voltage for DAC during VDAC mode operation
VREF2 (DSM)	DSM	1.2 V	Reference voltage to Delta Sigma Modulator. This voltage is buffered in the DSM block by a 10 µA buffer.
VREF1 (DSM)	DSM	1.024 V	Reference voltage to Delta Sigma Modulator. This voltage is buffered in the DSM block by a 10 µA buffer.
VREF1_CM (DSM)	DSM	0.8 V	Common mode reference voltage for Delta Sigma Modulator
VREF2_CM (DSM)	DSM	0.7 V	Common mode reference voltage for Delta Sigma Modulator
VREF (TEMP SENSOR)	TEMP SENSOR	0.9 V	Analog ground option to auxiliary ADC

37.2.3 Powerdown Logic

The powerdown logic block collects powerdown signals that are generated by the Power Manager. These powerdown signals are used to power-down resource blocks (for example, COMP, DAC, SWCAP, ADC, DSM, IMO, POR, REFBUF, LDO, LCDREF, and FM). Inside the powerdown logic block, glue logic combines these signals and generates reference buffer power-down signals.

The powerdown logic block is divided into unique blocks of glue logic, such as BG Powerdown, DAC Powerdown, and SC Powerdown. As an example, the DAC Powerdown function is described. Each of the four DACs has its own powerdown signal generated by the Power Manager.

The four DACs have a single bandgap referenced voltage input used in the front end V-to-I converter. This voltage is buffered inside the reference generator block. The buffer must not be powered down unless all four DACs are powered down. The logic inside the powerdown logic block keeps the DAC buffer powered unless all four DACs are powered down.

37.2.4 Low Power Mode Behavior

During idle, sleep, and hibernate modes, the precision reference is disabled using power cutoff switches.

37.3 Registers

Table 37-2. Registers

Name	Description
BG_CR0	5 bits for configuring VDA/2 reference
PWRSTS_BG_TR	5 bits for setting bandgap inl_ctrl[4:0], 1 bit for trimbuf trim_fine

38. Delta Sigma Converter

The PSoC® 5 ADC is a high resolution ADC implemented in Delta Sigma technology. Delta Sigma converters are integrating converters that provide high SNR/resolution by oversampling, noise shaping, averaging, and decimation. A Delta Sigma analog-to-digital converter (ADC) has two main components: a modulator and a decimator. The modulator converts the analog input signal to a high data rate (oversampling), low resolution (usually 1 bit) bitstream, the average value of which gives the average of the input signal level. This bitstream is passed through a decimation filter to obtain the digital output at high resolution and lower data rate. The decimation filter is a combination of downsampler and a digital low pass (averaging) filter that averages the bitstream to get the digital output.

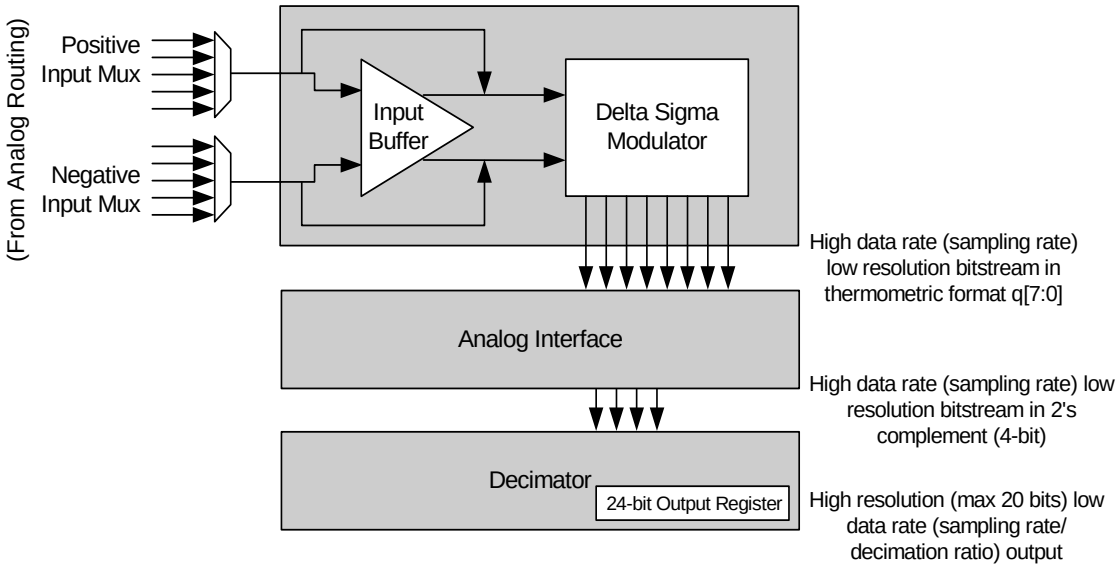
38.1 Features

- 8- to 20-bit resolution
- Configurable gain from 0.25 to 128
- Differential/single ended inputs
- Optional input buffer with RC low pass filter
- Internal and external reference options
- Reference filtering for low noise
- Incremental/continuous mode
- Gain and offset correction

38.2 Block Diagram

Figure 38-1 is the converter block diagram.

Figure 38-1. Delta Sigma Block Diagram



38.3 How It Works

The PSoC 5 Delta Sigma converter has a third-order modulator, followed by a fourth-order decimation filter. The modulator has a high impedance front end buffer followed by a bypassable RC filter.

- The modulator sends out a high data rate bitstream in thermometric format (see 38.3.2.6 Quantizer on page 328).
- The output of the modulator is passed on to the analog interface that converts the thermometric output to two's complement (4 bit) and passes it on to the decimation filter.
- The decimation filter takes 4-bit two's complement input and provides a higher resolution (user selectable) output at a lower data rate.

A detailed description of the individual blocks and their configuration options is given in this section.

38.3.1 Input Buffer

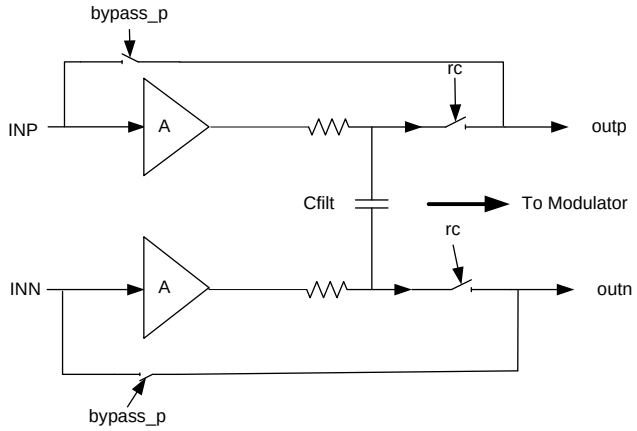
The input impedance of the modulator is very low and not suitable for many applications. For applications that require a higher input impedance, two buffers (one for each differential input) are provided. Figure 38-2 shows the buffer and the RC filter that follows it.

The buffers are of very low noise, are independent of each other, and can be bypassed (DSM_BUF0[1], DSM_BUF1[1]) or powered down (DSM_BUF0[0], DSM_BUF1[0]) individually by setting the bits listed in the braces. The buffer can also be used to amplify the input signal; it can be configured to provide gain of 1, 2, 4, and 8 in DSM_BUF1[3:2] register bits. The buffer has two separate modes, selected in the DSM_BUF0[2] bit to support a 0 to V_{dd} - 0.2 V input common mode range. The modes are:

- **Level Shifted** – Buffer output can be level shifted up from the input when the input is close to 0 V input common mode voltage range. The operating range is vssa-100 to vdda-600 mV.
- **Rail-to-Rail** – This is used when the input is rail-to-rail. The operating voltage range is vssa+100 mV to vdda-250 mV.

The input structure is illustrated in Figure 38-2.

Figure 38-2. Input Buffer Structure



An additional RC filtering option (DSM_BUF2[1]) is provided for lower noise contribution from the buffer, at the cost of the input voltage not settling completely. This incomplete settling causes a gain error that must be corrected later, as a part of the downstream filtering in the decimator. There is also an option to chop (DSM_BUF3[3]) the input and output stages of the buffer to keep the offset as low as 100 µV. The chopping frequency is user selectable (DSM_BUF3[2:0]) and can vary from 1/2 to 1/256 of the input sampling frequency. The buffer can also be operated in a low power mode (DSM_BUF2[0]).

The ADC (buffer) takes its inputs from analog globals, analog locals, analog mux bus, reference, and V_{ssa}. Registers DSM_SW0, DSM_SW2, DSM_SW3, DSM_SW4, DSM_SW6 help configure the positive and negative inputs.

Limit the maximum input signal amplitude to the modulator (after the buffer gain, if used) to the values in Table 38-1 for a proper operation. The values in Table 38-1 are for a 1.024 V reference. For other reference values, scale the maximum input amplitude accordingly.

Table 38-1. Maximum Input Signal Levels (ADC Reference Vref -> 1.024 V)

Gain	Modulator Quantization Levels		
	2 Level	3 Level	9 Level
0.25	3.0000	3.5	3.5680
0.5	1.5000	1.75	1.7840
1	0.75	0.875	0.892
2	0.3750	0.4375	0.4460
4	0.1875	0.2188	0.2230
8	0.0938	0.1094	0.1115
16	0.0469	0.0547	0.0558

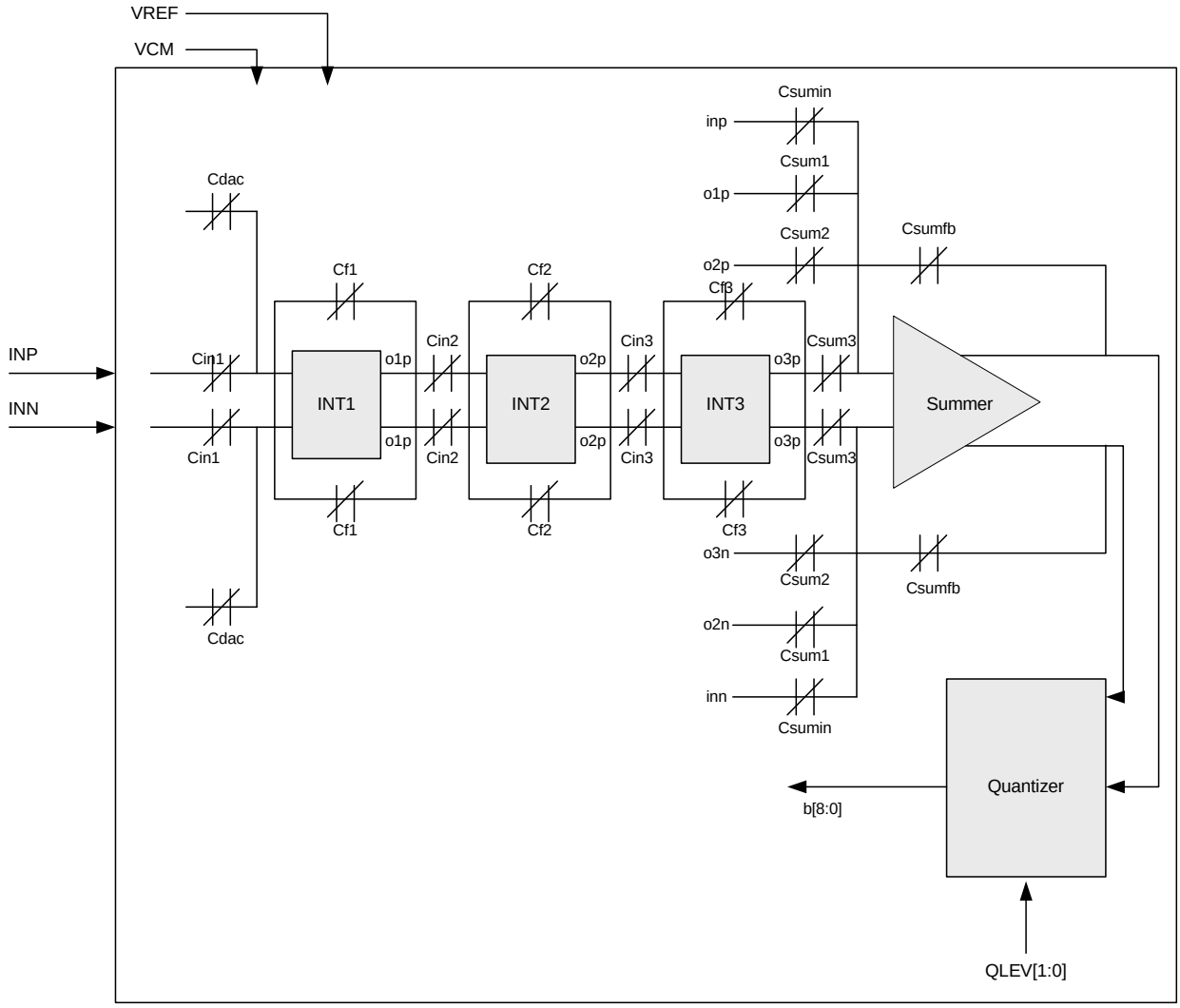
38.3.2 Delta Sigma Modulator

The Delta Sigma modulator does:

- Sampling the input signal (oversampling)
- Optional gain by adjusting the ration of Cin1 to Cref
- Coarse quantization (2, 3, or 9 levels/1, 1.5, or 2.2 bits)
- Overload detection and chopping

PSoC 5 Delta Sigma modulator implementation is shown in Figure 38-3.

Figure 38-3. Delta Sigma Modulator Implementation



The Delta Sigma modulator consists of these subsystems:

- Three active integrators
- An active summer
- A programmable quantizer
- A switched capacitor feedback DAC

A few points about the modulator:

- The three active integrators and the programmable quantizer form the third order modulator. The transfer function of the integrators and the quantizer together account for the high pass noise shaping. Higher the order of the modulator, better is the high pass filter response and lower is the noise in the signal frequency band.
- The three integrators and quantizer stages are followed by an active summer. The analog input and the output of all three opamp stages are summed here.
- The summer output is quantized by a quantizer. The quantizer is programmable to output 2, 3, or 9 levels.
- The DAC (V_{ref}, V_{gnd} and C_{ref} constitute the DAC) connects the quantizer output back to the first stage opamp input. It is this feedback DAC that ensures that the average of the quantizer output is equal to the average input signal level.

38.3.2.1 Clock Selection

Any one of the four analog clocks or a UDB-generated clock can be used as the input sampling clock. The clock input can also be disabled. The DSM0_CLK register helps in selecting the clock source and enabling or disabling it. The maximum clock that can be applied to the modulator is 6.144 MHz. Make certain that the clock to the decimator is equal to fs/n, where n = 2,3,4..., fs is the PHUB clock.

38.3.2.2 Capacitance Configuration

All the capacitors shown in Figure 38-3 on page 321 have binary weighted programmability. The value of a capacitance can be configured by setting the following three fields:

- **Offset Capacitance** – single bit that enables or disables an offset capacitance
- **Cap Array[n:0]** – n+1 binary weighted bits
- **LSB Enable** – additional unit capacitance

Capacitance configuration (configuration of the above fields) is done in registers DSM_CR4 through DSM_CR12.

Capacitance value is described by following equation:

$$\text{Cap value} = (\text{offset} \times C_{off}) + (\text{cap[n:0]} \times C_{unit}) + (\text{EN} \times C_{unit})$$

Where:

- C_{off} is the offset capacitor value
- C_{unit} is the unit capacitor value
- Offset is the binary value (single bit) programmed in the offset field
- EN (LSB enable) is the binary value (single bit) programmed in the EN field
- Cap[n:0] is the decimal equivalent of the binary value programmed in the cap array[n:0] field

The unit capacitance, offset capacitance, and default values for all of the capacitances are given inTable 38-2.

Table 38-2. Capacitance Values

Register Bit	Description	Value	Default	Typical Value
FCAP1OFFSET	Offset cap for first stage feedback cap	3.4 pF	0	8 pF
FCAP1[6:0]	Binary weighted first stage feedback cap	C_{unit} = 100 fF 100 fF - 12.8 pF in 100 fF steps	1010000	
FCAP1EN	Enable for LSB CAP of FCAP1		0	
IPCAP1OFFSET	Offset cap for first stage input cap	4.8 pF	0	4.4 pF
IPCAP1[6:0]	First stage Input CAP (binary)	C_{unit} = 100 fF 100 fF – 12.8 pF in 100 fF steps	0101100	
IPCAP1EN	Enable for LSB cap of IPCAP1		0	
DACCAP[5:0]	DAC cap (each unit) - binary	C_{unit} = 96 fF(2 LSBs) and 100 fF(4 MSBs) 96 fF – 62898 fF in variable steps	101100	4.4 pF
DACCAPEN	Enable for LSB CAP of DAC		0	
RESCAP[2:0]	Resonator cap (binary)	C_{unit} = 12 fF 12 fF - 96 fF in 12 fF steps	000	0 fF
RESCAPEN	Enable for LSB cap of RESCAP		0	
FCAP2[3:0]	Second stage Feedback cap - binary	C_{unit} = 50 fF 50-800 fF in 50 fF steps	1011	0.55 pF
FCAP2EN	Enable for LSB CAP of FCAP2		0	
IPCAP2[2:0]	Second stage input CAP - binary	C_{unit} = 50 fF 50-400 fF in 50 fF steps	101	0.25 pF
IPCAP2EN	Enable for LSB Cap of IPCAP2		0	
FACP3[3:0]	Third stage feedback cap	C_{unit} = 100 fF 100 fF-1.6 pF in 100 fF steps	1110	1.4 pF
FCAP3EN	Enable for LSB Cap of FCAP3		0	
IPCAP3[2:0]	Third stage input cap	C_{unit} = 50 fF 50-400 fF in 50 fF steps	101	0.25 pF
IPCAP3EN	Enable for LSB Cap of IPCAP3		0	

Table 38-2. Capacitance Values *(continued)*

Register Bit	Description	Value	Default	Typical Value
SUMCAPIN[4:0]	Summer cap for input path	C_{unit} = 50 fF 50 – 1.6 pF in 50 fF steps	00101	0.25 pF
SUMCAPINEN	Enable for LSB Cap of SUMCAPIN		0	
SUMCAPFB[3:0]	Summer cap for feedback path	C_{unit} = 50 fF 50-800 fF in 50 fF steps	1010	0.5 pF
SUMCAPFBEN	Enable for LSB Cap of SUMCAPFB		0	
SUMCAP1[2:0]	Summer cap for first stage output	C_{unit} = 50 fF 50-400 fF in 50 fF steps	101	0.25 pF
SUMCAP1EN	Enable for LSB Cap of SUMCAP1		0	
SUMCAP2[2:0]	Summer cap for second stage output		101	0.25 pF
SUMCAP2EN	Enable for LSB Cap of SUMCAP2		0	
SUMCAP3[2:0]	Summer cap for third stage output		101	0.25 pF
SUMCAP3EN	Enable for LSB Cap of SUMCAP3		0	

38.3.2.3 Gain Configuration

The modulator provides gain from 0.25 to 16 to the input signal. Gain is the ratio of input and DAC capacitances, as described in the following equation.

$$Gain = C_{in}/C_{ref} \qquad \textbf{Equation 1}$$

However, increasing only the input capacitance to increase gain disturbs the transfer characteristics of the modulator. Therefore, other capacitors also must be scaled to maintain the modulator transfer characteristics. Recommended values of capacitors for gains of 1, 2, 4, 8 are shown in Table 38-3, and those for 16, 0.25, and 0.5 are shown in Table 38-4.

Table 38-3. Gains 1, 2, 4, and 8

Register Bit	Gain = 1		Gain = 2		Gain = 4		Gain = 8	
	Bit Setting	Typical Value	Bit Setting	Typical Value	Bit Setting	Typical Value	Bit Setting	Typical Value
IPCAP1OFFSET	0	4.4 pF	0	8.8 pF	1	17.6 pF	1	17.6 pF
IPCAP1[6:0]	0101100		1011000		1111111		1111111	
IPCAP1EN	0		0		1		1	
DACCAP[5:0]	101100	4.4 pF	101100	4.4 pF	101100	4.4 pF	010110	2.2 pF
DACCAPEN	0		0		0		0	
SUMCAPIN[4:0]	00101	0.25 pF	01000	0.4 pF	10000	0.8 pF	10000	0.8 pF
SUMCAPINEN	0		0		0		0	
SUMCAPFB[3:0]	1010	0.5 pF	1000	0.4 pF	1000	0.4 pF	0100	0.2 pF
SUMCAPFBEN	0		0		0		0	
SUMCAP1[2:0]	101	0.25 pF	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP21EN	0		0		0		0	
SUMCAP2[2:0]	101	0.25 pF	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP2EN	0		0		0		0	
SUMCAP3[2:0]	101	0.25 pF	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP3EN	0		0		0		0	

Table 38-4. Gains 16, 0.5, and 0.25

Register Bit	Gain = 16		Gain = 0.5		Gain = 0.25	
	Bit Setting	Typical Value	Bit Setting	Typical Value	Bit Setting	Typical Value
IPCAP1OFFSET	1	17.6 pF	0	2.2 pF	0	1.1 pF
IPCAP1[6:0]	1111111		0010110		0001011	
IPCAP1EN	1		0		0	

Table 38-4. Gains 16, 0.5, and 0.25 *(continued)*

Register Bit	Gain = 16		Gain = 0.5		Gain = 0.25	
DACCAP[5:0]	001011	1.1 pF	101100	4.4 pF	101100	4.4 pF
DACCAPEN	0		0		0	
SUMCAPIN[4:0]	10000	0.8 pF	00010	0.1 pF	00001	0.05 pF
SUMCAPINEN	0		0		0	
SUMCAPFB[3:0]	0010	0.1 pF	1000	0.4 pF	1000	0.4 pF
SUMCAPFBEN	0		0		0	
SUMCAP1[2:0]	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP21EN	0		0		0	
SUMCAP2[2:0]	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP2EN	0		0		0	
SUMCAP3[2:0]	100	0.2 pF	100	0.2 pF	100	0.2 pF
SUMCAP3EN	0		0		0	

38.3.2.4 Power Configuration

There are separate power settings for the first opamp stage, the summer, and the quantizer. The second and third stages share the same power settings. The power for all of these stages is configured in registers DSM_CR14 and DSM_CR16. The various configurable power settings are shown in Table 38-5.

Table 38-5. Configurable Power Settings

Register Bit	Description	Truth Table, Typical IDD
POWER1	Power control for first stage	000 - LOW (42 µA) 001 - MEDIUM (114 µA) 010 - HIGH (430 µA) 011 - 1.5X (650 µA) 100 - 2X (900 µA) 101 - C/2 at 3MSPS (254 µA) 110 = C/4 at 3MSPS (170 µA) 111 - 2.5X (1.35 mA)
POWER2_3[2:0]	Power control for second stage/third stage	000 - LOW (4 µA) 001 - MEDIUM (16 µA) 010 - HIGH (62 µA) 011 - 1.5X (100 µA) 100 - 2X (135 µA)
POWER_SUM[2:0]	Power control for summer	000 - LOW (4 µA) 001 - MEDIUM (16 µA) 010 - HIGH (62 µA) 011 - 1.5X (100 µA) 100 - 2X (135 µA)
POWER_COMP[1:0]	Comparator power control	00 - Very Low (2.2 µA) 01 - Normal (8.6 µA) 10 - 6 MHz (17 µA) 11 - 6 MHz (35 µA)

Table 38-5 indicates how to configure power for the individual blocks. Power dissipation, capacitances, clock frequency and quantization levels are interrelated to each other.

Configuring power without varying the other parameters mentioned above affects the proper operation of the modulator. The following tables show a set of operational modes that indicate how to configure power based upon the other parameters or vice versa.

Table 38-6. Power Configuration Based on Quantization Levels and Clock Frequency

Register Bit	Mode - 3 MHz 9 Level		Mode - 6 MHz 9 Level		Mode - 3 MHz 2 Level and 3 Level	
	Bit Setting	Typical Value	Bit Setting	Typical Value	Bit Setting	Typical Value
FCAP1OFFSET	0	8 pF	0	2 pF	1	16 pF
FCAP1[6:0]	1010000		0010100		1111110	
FCAP1EN	0		0		0	
IPCAP1OFFSET	0	4.4 pF	0	1.1 pF	0	4.4 pF
IPCAP1[6:0]	0101100		0001011		0101100	
IPCAP1EN	0		0		0	
DACCAP[5:0]	101100	4.4 pF	001011	1.1 pF	101100	4.4 pF
DACCAPEN	0		0		0	
RESCAP[2:0]	000	0fF	000	0fF	000	0fF
RESCAPEN	0		0		0	
FCAP2[3:0]	1011	0.55 pF	0001	0.1 pF	1011	0.55 pF
FCAP2EN	0		0		0	
IPCAP2[2:0]	101	0.25 pF	001	0.05 pF	101	0.25 pF
IPCAP2EN	0		0		0	
FACP3[3:0]	1110	1.4 pF	0011	0.3 pF	1110	1.4 pF
FCAP3EN	0		0		0	
IPCAP3[2:0]	101	0.25 pF	001	0.05 pF	101	0.25 pF
IPCAP3EN	0		0		0	
SUMCAPIN[4:0]	00101	0.25 pF	00001	0.05 pF	00101	0.25 pF
SUMCAPINEN	0		0		0	
SUMCAPFB[3:0]	1010	0.5 pF	0010	0.1 pF	1010	0.5 pF
SUMCAPFBEN	0		0		0	
SUMCAP1[2:0]	101	0.25 pF	001	0.05 pF	101	0.25 pF
SUMCAP21EN	0		0		0	
SUMCAP2[2:0]	101	0.25 pF	001	0.05 pF	101	0.25 pF
SUMCAP2EN	0		0		0	
SUMCAP3[2:0]	101	0.25 pF	001	0.05 pF	101	0.25 pF
SUMCAP3EN	0		0		0	
QLEVEL[1:0]	10	level=9	10	level=9	00 or 01	level=2 or 3
ODET_TH[4:0]	01100	12	01100	12	01100	12
FCHOP[2:0]	001	Fclk/4	001	Fclk/4	001	Fclk/4
NONOV[1:0]	01	3.5 ns	00	1.5 ns	01	3.5 ns
POWER1[2:0]	010	430 µA	010	430 µA	010	430 µA
POWER2_3[2:0]	010	62 µA	010	62 µA	010	62 µA
POWER_SUM[2:0]	010	62 µA	010	62 µA	010	62 µA
POWER_COMP[1:0]	01	9 µA	10	18 µA	01	9 µA

Table 38-7. Power Configuration Based on Capacitances

Register Bit	Mode - C/2		Mode - C/4		Mode - C/8	
	Bit Setting	Typical Value	Bit Setting	Typical Value	Bit Setting	Typical Value
FCAP1OFFSET	0	4 pF	0	2 pF	0	1 pF
FCAP1[6:0]	0101000		0010100		0001010	
FCAP1EN	0		0		0	
IPCAP1OFFSET	0	2.2 pF	0	1.1 pF	0	0.5 pF
IPCAP1[6:0]	0010110		0001011		0000101	
IPCAP1EN	0		0		0	
DACCAP[5:0]	010110	2.2 pF	001011	1.1 pF	000101	0.5 pF
DACCAPEN	0		0		0	
RESCAP[2:0]	000	0fF	000	0fF	000	0fF
RESCAPEN	0		0		0	
FCAP2[3:0]	0101	0.25 pF	0001	0.1 pF	1011	0.1 pF
FCAP2EN	0		0		0	
IPCAP2[2:0]	010	0.1 pF	001	0.05 pF	101	0.05 pF
IPCAP2EN	0		0		0	
FACP3[3:0]	0101	0.5 pF	0011	0.3 pF	1110	0.3 pF
FCAP3EN	0		0		0	
IPCAP3[2:0]	010	0.1 pF	001	0.05 pF	101	0.05 pF
IPCAP3EN	0		0		0	
SUMCAPIN[4:0]	00010	0.1 pF	00001	0.05 pF	00101	0.05 pF
SUMCAPINEN	0		0		0	
SUMCAPFB[3:0]	0100	0.2 pF	0010	0.1 pF	0010	0.1 pF
SUMCAPFBEN	0		0		0	
SUMCAP1[2:0]	010	0.1 pF	001	0.05 pF	101	0.05 pF
SUMCAP21EN	0		0		0	
SUMCAP2[2:0]	010	0.1 pF	001	0.05 pF	101	0.05 pF
SUMCAP2EN	0		0		0	
SUMCAP3[2:0]	010	0.1 pF	001	0.05 pF	101	0.05 pF
SUMCAP3EN	0		0		0	
QLEVEL[1:0]	10	level=9	10	level=9	10	level=9
ODET_TH[4:0]	01100	12	01100	12	01100	12
FCHOP[2:0]	001	Fclk/4	001	Fclk/4	001	Fclk/4
NONOV[1:0]	01	3.5 ns	01	3.5 ns	01	3.5 ns
POWER1[2:0]	101	254 µA	110	170 µA	000	114 µA
POWER2_3[2:0]	001	16 µA	001	16 µA	001	16 µA
POWER_SUM[2:0]	001	16 µA	001	16 µA	001	16 µA
POWER_COMP[1:0]	10	18 µA	10	18 µA	10	18 µA

Table 38-8. Configuration Based on Power

Register Bit	Mode - Medium Power		Mode - Low Power	
	Bit Setting	Typical Value	Bit Setting	Typical Value
FCAP1OFFSET	0	8 pF	0	8 pF
FCAP1[6:0]	1010000		1010000	
FCAP1EN	0		0	
IPCAP1OFFSET	0	4.4 pF	0	4.4 pF
IPCAP1[6:0]	0101100		0101100	
IPCAP1EN	0		0	
DACCAP[5:0]	101100	4.4 pF	101100	4.4 pF
DACCAPEN	0		0	
RESCAP[2:0]	000	0fF	000	0fF
RESCAPEN	0		0	
FCAP2[3:0]	1011	0.55 pF	1011	0.55 pF
FCAP2EN	0		0	
IPCAP2[2:0]	101	0.25 pF	101	0.25 pF
IPCAP2EN	0		0	
FACP3[3:0]	1110	1.4 pF	1110	1.4 pF
FCAP3EN	0		0	
IPCAP3[2:0]	101	0.25 pF	101	0.25 pF
IPCAP3EN	0		0	
SUMCAPIN[4:0]	00101	0.25 pF	00101	0.25 pF
SUMCAPINEN	0		0	
SUMCAPFB[3:0]	1010	0.5 pF	1010	0.5 pF
SUMCAPFBEN	0		0	
SUMCAP1[2:0]	101	0.25 pF	101	0.25 pF
SUMCAP21EN	0		0	
SUMCAP2[2:0]	101	0.25 pF	101	0.25 pF
SUMCAP2EN	0		0	
SUMCAP3[2:0]	101	0.25 pF	101	0.25 pF
SUMCAP3EN	0		0	
QLEVEL[1:0]	10	level=9	10	level=9
ODET_TH[4:0]	01100	12	01100	12
FCHOP[2:0]	001	Fclk/4	001	Fclk/4
NONOV[1:0]	01	3.5 ns	01	3.5 ns
POWER1[2:0]	010	114 µA	010	42 µA
POWER2_3[2:0]	010	16 µA	010	4 µA
POWER_SUM[2:0]	010	16 µA	010	4 µA
POWER_COMP[1:0]	01	9 µA	01	9 µA

38.3.2.5 Other Configuration Options

The modulator can be chopped for a low offset of 100 µV. The chopping frequency can be set from fclk/2 to fclk/256, where fclk is the input sampling clock. Chopping enable and chopping frequency setting are done in the DSM_CR2 register.

The modulator can be configured for inverting the gain by setting the sign bit in DSM_CR3[7].

The modulator can be reset (all capacitances are reset) by the UDB or decimator, and the reset source is selected by the DSM_CR2[7] register. More details about reset are in the Reset chapter on page 119.

38.3.2.6 Quantizer

The quantizer can be configured for 2, 3, or 9 levels. A 9 level quantizer offers a better SNR and a 2 level quantizer offers better linearity. Depending on the application requirement, the user can choose quantization levels. The number of quantization levels is configured in DSM_CR0[1:0] register bits. The quantizer outputs data in thermometric format. The quantizer output is stored in the register DSM_OUT1.

Thermometric format is explained by the pattern of output levels shown in the following table. In thermometric format, the number of ones increases from LSB to MSB as the quantization level increases.

Table 38-9. Quantizer Output Data

Level	Quantizer Output Data
2 Level Quantizer	
Level 1	00000000
Level 2	11111111
3 Level Quantizer	
Level 1	00000000
Level 2	00001111
Level 3	11111111
9 Level Quantizer	
Level 1	00000000
Level 2	00000001
Level 3	00000011
Level 4	00000111
Level 5	00001111
Level 6	00011111
Level 7	00111111
Level 8	01111111
Level 9	11111111

38.3.2.7 Reference Options

The Delta Sigma channel has selectable analog reference input (REFBUF0) options, as shown in Figure 38-4 on page 329. Also illustrated are the opamp output common mode (VCMBUF0) and the negative input buffer (REFBUF1) selection schemes. The various reference selections for the DSM ADC may be broadly classified into the following modes:

- Internal Reference (reference generated on-chip) that is buffered but unfiltered (Figure 38-5 on page 329)
- Internal Reference that is buffered and filtered with an external capacitor tied between P0[3] and ground or P3[2] and ground (Figure 38-6 on page 330)
- External Reference source driving reference into the DSM (Figure 38-7 on page 330)

Figure 38-4. Delta Sigma Channel Analog Reference Selection

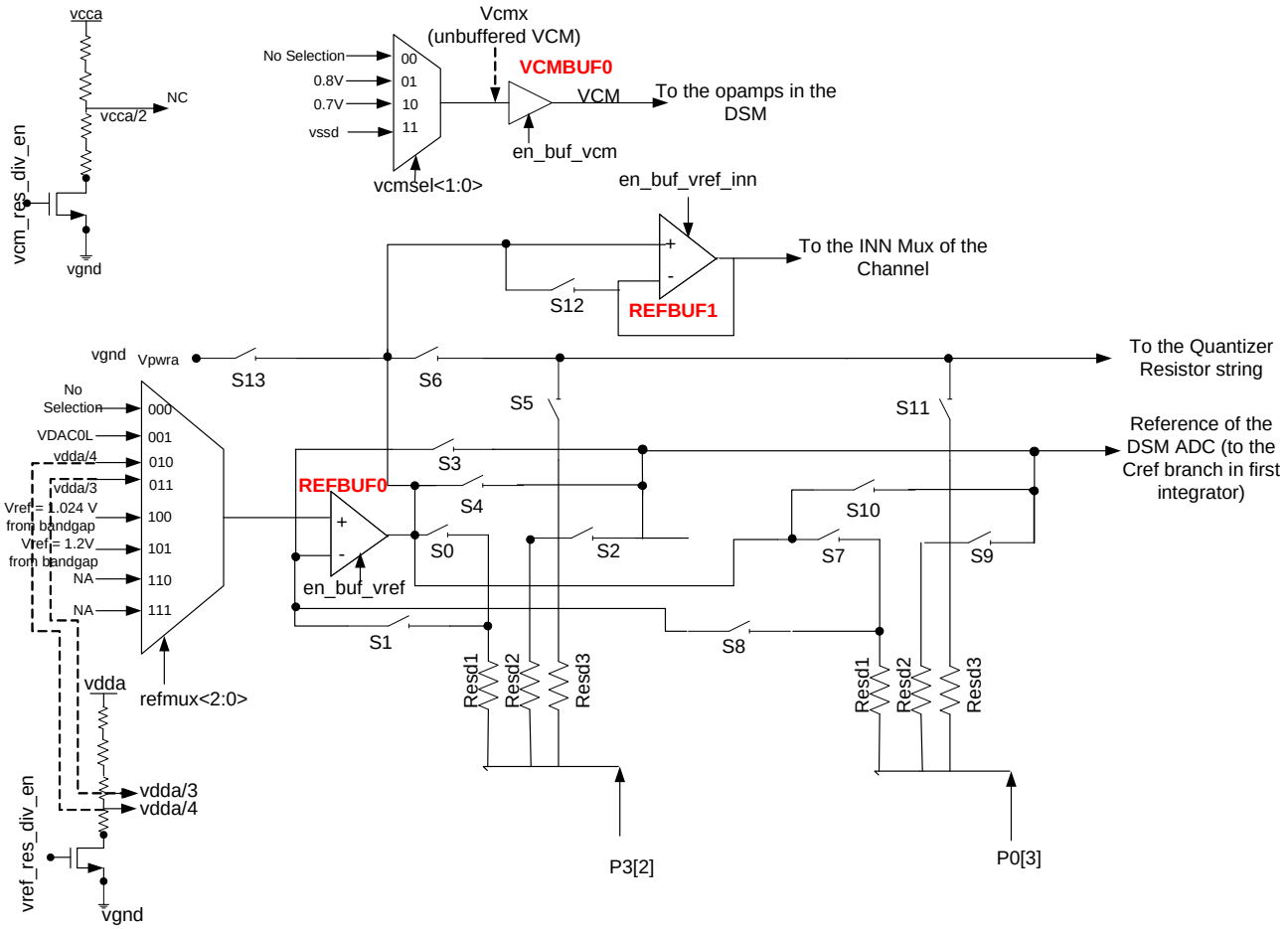


Figure 38-5. Connection Scenario: Internal Reference with No RC Filtering (using P0[3])

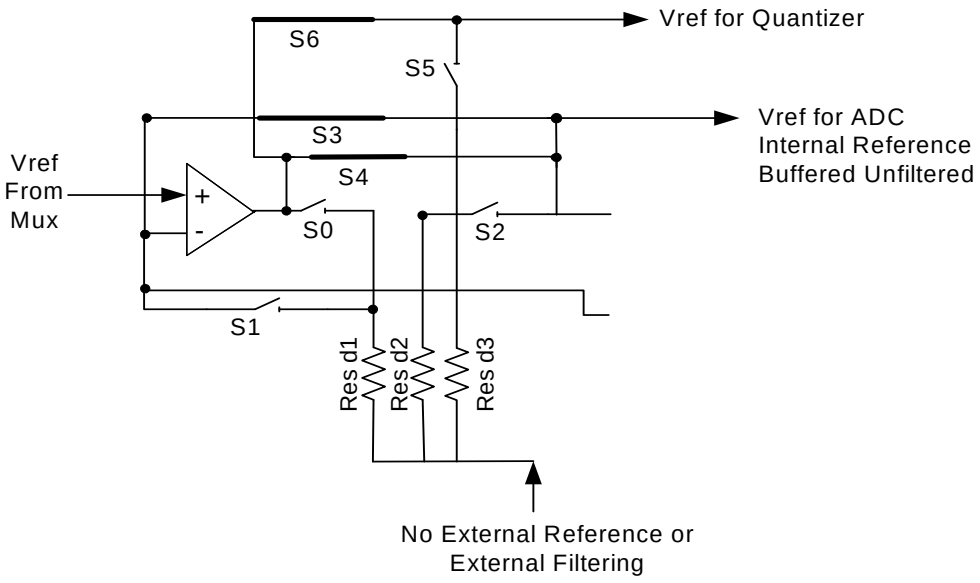


Figure 38-6. Connection Scenario: Internal Reference with RC Filtering

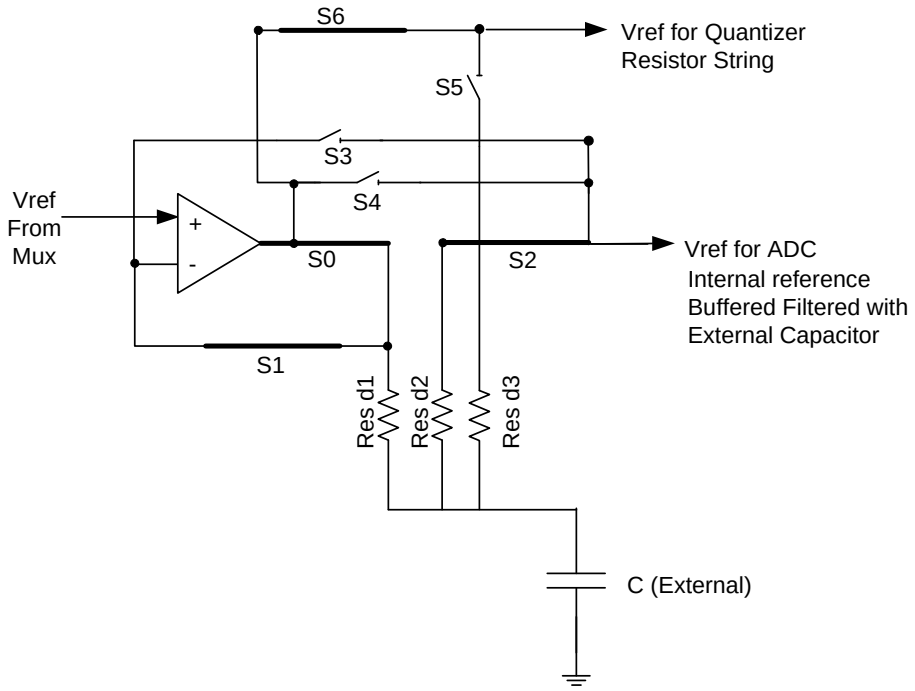
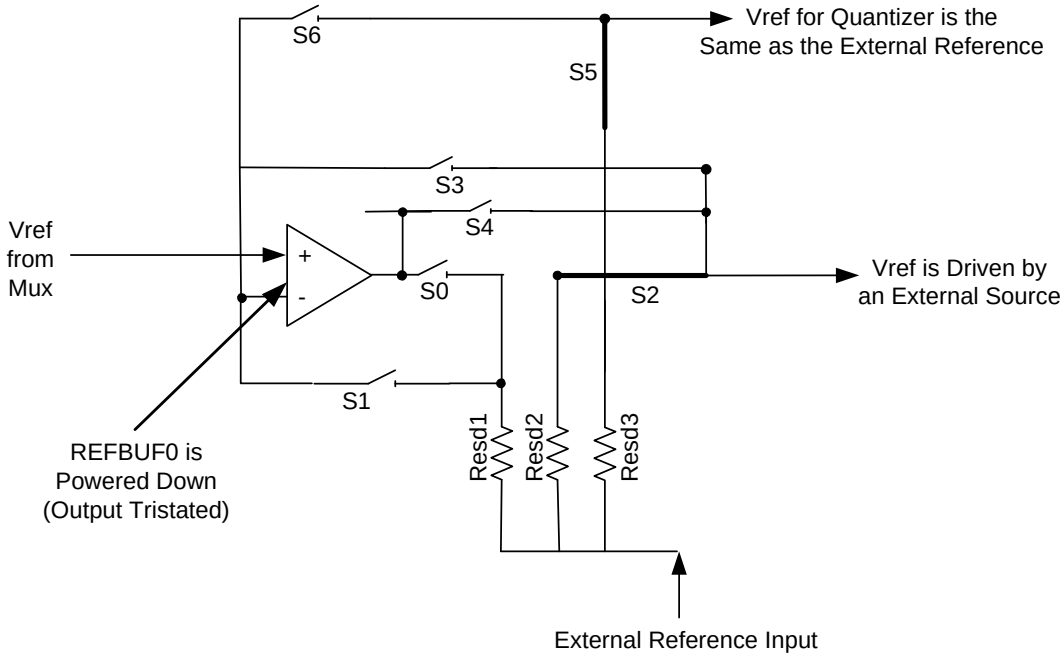


Figure 38-7. Connection Scenario: External Reference Only



There are several selectable options for internal reference, based on refmux[2:0] programming in DSM_REF0 register. The places in the DSM block (Figure 38-3 on page 321) that require a reference value are:

- DAC capacitor (C_{ref}) sampling in the first integrator
- Reference for the resistive ladder inside the quantizer block
- Common Mode Voltage (VCM) for the differential circuits. This voltage is typically 0.8 V with an option to go to 0.7 V for better head rooms. A provision for applying VDD/2 is also provided.

38.3.2.8 Reference for DSM: Usage Guidelines

The following table shows the state of various switches and the two reference buffers for certain selectable reference options.

Not every possible combination of closing the switches marked S0-S13 is discussed in this section. The configuration of these switches (therefore the reference selection) is made in registers DSM_REF2 and DSM_REF3. The reference buffers can be configured in low, medium, high, and turbo power modes in the DSM_CR17 register. The common mode voltage buffer, internal reference voltage buffer, and the negative input buffer are powered down, using DSM_CR17[1]; DSM_CR17[0], and DSM_REF0[3] register bits, respectively.

1. Power on the VCMBUF0 for the DSM to function.
2. Turn on the reference buffer REFBUF1 only when you want to drive the ADC reference to the negative input mux of the DSM channel.
3. Power down REFBUF0 only when you want to drive reference to the ADC from an off-chip source (See the external reference option in Table 38-10).

To get low reference noise, the option to filter is provided with the special connections to pins P3 [2] and P0 [3], as shown in Figure 38-4 on page 329 and Figure 38-7 on page 330. Therefore, for low noise floor requirements, use the external capacitor filter. Only two pins, P3 [2] and P0 [3], are dedicated for this purpose in PSoC 5 devices. The switches in Table 38-10 that are marked as ON mean that the switch is closed, and a path is created for reference to reach DSM. Empty cells indicate that the switches are open.

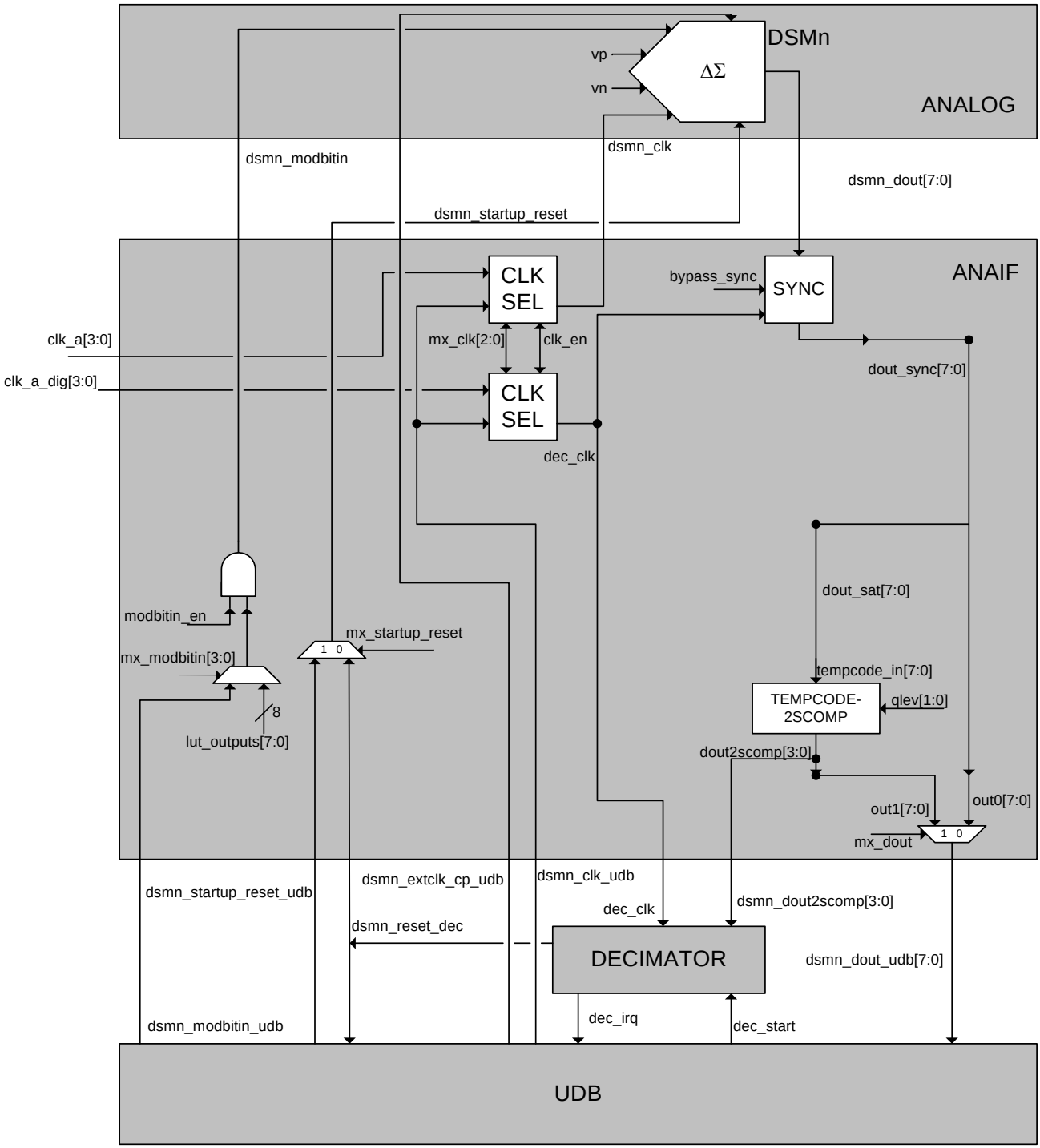
Table 38-10. Analog Reference Modes for the Delta Sigma Channel

SN	Mode	Switch States														REFBUF0
		S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	
1	Internal Reference (No Filtering)				ON	ON		ON								ON
2	Internal Reference (Filter with P3[2])	ON	ON	ON				ON								ON
3	Internal Reference (Filter with P0[3])							ON	ON	ON	ON					ON
4	External Reference only (P3[2])			ON			ON	ON								OFF
5	External Reference only (P0[3])							ON			ON		ON			OFF
6	V_{pwra} is internal reference					ON		ON							ON	OFF

38.3.3 Analog Interface

The analog interface connects the modulator to the other blocks including the decimator and the UDB. As shown in Figure 38-8 on page 332, the analog interface converts thermometric code sent by the modulator to two's complement and allows for selection of modulation input as well as selecting and synchronizing clocks.

Figure 38-8. Analog Interface



38.3.3.1 Conversion of Thermometric Code to Two's Complement

The following table shows the conversion from thermometric format to two's complement for 2, 3, and 9 level quantizations performed by the analog interface. This two's complement input is fed to the decimator.

As shown in Table 38-11, the two's complement output is always symmetric. That is, the output varies from –1 to +1 to –4 to +4. This results in a symmetric ADC output. For example, an 8-bit ADC would give output counts from –128(0xFF) to 128(0x100) instead of –128 to +127(0x7F). The maximum positive count exceeds its ideal value by 1. If the ADC saturates at 0x100 and the result is read as an 8-bit value, you'll read 0x00. This can lead to wrong interpretation of the actual input to the ADC. So, whenever the ADC input is expected to drive the output to saturation, read the output in a variable with bit width greater than the actual bit width of the ADC output.

Table 38-11. Two's Complement Conversion Table

Inputs		Output	
qlev[1:0]	dout[7:0]	dout2scomp[3:0]	
00	00000000	1111	-1
00	11111111	0001	+1
01	00000000	1111	-1
01	00001111	0000	0
01	11111111	0001	+1
1x	00000000	1100	-4
1x	00000001	1101	-3
1x	00000011	1110	-2
1x	00000111	1111	-1
1x	00001111	0000	0
1x	00011111	0001	+1
1x	00111111	0010	+2
1x	01111111	0011	+3
1x	11111111	0100	+4

38.3.3.2 Modulation Input

As discussed in 38.3.2.5 Other Configuration Options on page 328, modulator gain can be inverted by the sign bit in DSM_CR3. The sign can also be changed by a direct digital input from LUTs or the UDB. The modulation input assists in this process. Depending on whether the modulation input is high or low, the gain is normal or inverted. The modulation input can be enabled by setting the DSM_CR3[4] register bit. Modulation input is selected by DSM_CR3[3:0] control bits.

38.3.3.3 Clock Selection and Synchronization

The output of the modulator (quantizer) Q[7:0] can be synchronized with respect to the digitally aligned clock of the analog clock selected for the modulator. As mentioned in 38.3.2 Delta Sigma Modulator on page 321, clock selection is done by DSM_CLK[2:0] register bits. Clock synchronization is enabled by clearing the DSM_CLK[4] register bit.

38.3.4 Decimator

The decimator takes the 4-bit input (low resolution) in two's complement format and converts it into a high resolution output. The 4-bit two's complement values coming into the decimator at the input sampling rate are averaged over a specified number of samples (decimation ratio), down sampled, and passed through an optional post-processing filter, achieving a higher resolution. The decimator in PSoC 5 devices is a fourth order Cascaded Integrator Comb (CIC) filter. The decimator structure is shown in Figure 38-9 on page 335.

38.3.4.1 Start and End of Conversion

The start of conversion (SOC) can be provided either in software or hardware. The software SOC is provided by writing a '1' to the start_conv bit of DEC_CR register. There is a hardware terminal for providing a hardware SOC. To use it, it must be enabled by writing a '1' to the xstart_en bit in the DEC_CR register.

The conv_done bit in DEC_SR register signals the end of conversion. Two separate hardware connection coming from this bit triggers the DMA and interrupt. The interrupt signal can be masked by the intr_mask bit in the DEC_SR register.

The decimator output count goes from -2^n to + 2^n, where n is the resolution. The positive count does not terminate at 2^n - 1 as in a regular 2's complement number.

38.3.4.2 Shifters

There are two shifters in the block — one in front of the CIC filter and another one in front of the post processor. The input shift values are programmed depending on the decimation ratio and quantization level to ensure that ADC results are available in the Q31 format.

The shift values are programmed in register DEC_SHIFT1. The shift values to be programmed in DEC_SHIFT1 and DEC_SHIFT2 for various decimation ratios (DR1 and DR2) and quantization levels are shown in Table 38-12 and Table 38-13 on page 334.

Table 38-12. Programmed Shifter1 Values for Various Decimation Ratios (Programmed in DR1)

Decimation Ratio	Quantization Levels	Max Values in Range	Bit Width	Shift Adjustment
8	2, 3	4096 to –4096	12	Left shift 20
8	9	16384 to –16384	14	Left shift 18
16	2,3	65536 to –65536	16	Left shift 16
16	9	262144 to –262144	18	Left shift 14
32	2, 3	1048576 to –1048576	20	Left shift 12
32	9	4194304 to –4194304	22	Left shift 10
64	2, 3	16777216 to –16777216	24	Left shift 8
64	9	67108864 to –67108864	26	Left shift 6
128	2, 3	268435456 to –268435456	28	Left shift 4
128	9	173741824 to –1073741824	30	Left shift 2

Table 38-13. Programmed Shifter2 Values for Various Decimation Ratios (Programmed in DR2)

Value of D2	Right Shift Value
1	No shift, bypass sync (boxcar) filter
16	4
32	5
64	6
128	7
256	8
512	9
1024	10

38.3.4.3 CIC Filter

The CIC filter has four cascaded integrator sections operating at the modulator sample rate, followed by four cascaded comb sections operating at a lower sample rate (determined by DR1). This combination implements a sinc4 Finite Impulse Response (FIR) filter. The CIC filter is controlled by a finite state machine that allows it to sequence events in the various modes of operation of the decimator. The decimation ratio is programmed in the DEC_DR1 register. The registers in CIC filter are 32-bits wide and, therefore, for proper operation, the decimation ratio should not exceed the values given in Table 38-14.

Table 38-14. Maximum Decimation Ratio Values for CIC

Level	Bit Width	Encoding (Decimal)	Max Allowed
2	32	-1, 1	256
3	32	-1, 0, 1	215
9	32	-4, -3, -2, -1, 0, 1, 2, 3, 4	152

The decimation ratios to be configured for 12, 14, 16, and 20 bit resolutions for 9 level quantization are shown in Table 38-15.

Table 38-15. Decimation Ratios for 9 Level Quantization

Final Resolution	Clock, Decimation Ratio
12-bit	6.144 MHz, 16
14-bit	6.144 MHz, 32
16-bit	3.072 MHz, 64
20-bit	3.072 MHz, 16384 (CIC + post processor)

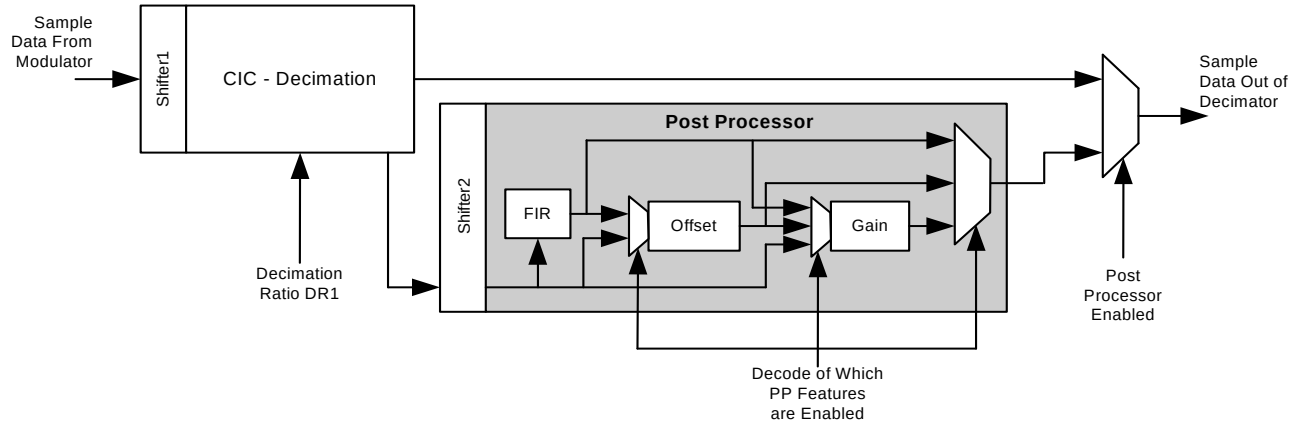
38.3.4.4 Post Processing Filter

The Post Processor receives 28-bit data from the output of the CIC Decimation filter for further convenience or post processing. Available functions are:

- Add a programmable offset coefficient to the CIC result
- Multiply a programmable gain coefficient to the CIC result
- Apply both offset and gain
- Apply a sinc1 FIR filter
- Apply both a sinc1 filter and offset correction
- Apply both a sinc1 filter and gain correction
- Apply all three

When more than one of the three functions is enabled to operate concurrently on the data, they are always performed in the order: FIR > Offset > Gain. The decimator process is shown in Figure 38-9.

Figure 38-9. Decimator



The offset value to be added is programmed in registers DEC_OCOR, DEC_OCORM, and DEC_OCORH. The 24-bit offset is given in signed two's complement format. The registers are coherency interlock protected (see 38.3.5 Coherency Protection on page 335).

The gain correction coefficient is programmed in registers DEC_GCOR, DEC_GCORH. The number of bits that are valid in the above register is programmed in the DEC_GVAL[3:0] register bits. This allows use of a part of the 16-bits for gain correction. The registers are coherency interlock protected. If the gain feature is used, the value programmed into the DR1 register (CIC decimation ratio) cannot be smaller than 2+2*GVAL, allowing time for the hardware to do a shift-add multiple during the decimation period.

The FIR filter is a summer that implements the sinc1 filter. It is used in cases where decimation ratios greater than 128 are required. When the FIR function is enabled, the Post Processor sums samples from the CIC filter, DR2 at a time, where DR2 (10 bits) is the decimation ratio programmed in the DEC_DR2, DEC_DR2H[1:0] registers.

Gain correction, offset correction, and FIR filtering features can be enabled and disabled in the DEC_CR[6:4] register bits. The Post Processor implements saturation logic that prevents over- and under-flow wraparound in the accumulator. If the DEC_CR[7] bit is set, the ALU does not wrap when the most positive or negative number is exceeded.

The output of the conversion is stored in registers OUTSAMP, OUTSAMPM, and OUTSAMPH. In some configurations of the block, output results of interest are placed in bits 23:8 of the output sample field. To allow reading such values in one bus cycle, an alignment feature is added to shift the result right by 8 bits. This feature is enabled by the OUTPUT_ALIGN bit of the DEC_SR register.

38.3.5 Coherency Protection

Coherency refers to the hardware added to a block to protect against malfunctions of the block in cases where register fields are wider than the bus access, leaving intervals in time when fields are partially written or read (incoherent). Coherency checking is an option and is enabled in the DEC_COHER register.

The hardware provides coherency checking on three register fields that are all up to three bytes wide:

- Gain and Gain Value (write protected) – really two fields, but they are checked for coherency as if they are a single field protected on writes so that the underlying hardware does not incorrectly use the field when it partially updated by system software.
- Offset Value (write protected) – protected on writes so that the underlying hardware does not incorrectly use the field when it is partially updated by the system software.
- Output Sample Value (read protected) – protected on reads so that the underlying hardware does not update it when partially read by the system software or DMA. Depending on the configuration of the block, not all bits of the output sample register are of interest.

The coherency methodology allows for any size output field and handles it properly. In the COHER register, coherency is both enabled, and a Key Coherency Byte is selected. The Key Coherency Byte allows the user to tell the hardware which byte of the field will be written or read last when an update to the field is desired. Each for the three protected fields has a Coherency Interlock Flag (CIF). This flag signifies whether the field is coherent.

The coherency hardware understands both 8-bit and 16-bit accesses and when tracking coherency, handles each appropriately. A hard or soft reset sets all CIF to coherent.

38.3.5.1 Protecting Writes (Gain/Offset) with Coherency Checking

Starting from a coherent state (CIF is set), the software can write any of the other non-key bytes. This action flags the field incoherent (clears the CIF). When a field is incoherent, it is ignored by the underlying hardware, and a shadow register containing the last valid value is used. The field remains flagged incoherent until the Key Coherency Byte is written. At this time, the field is flagged coherent (CIF is again set), and the next time the hardware needs the field value, the new value is used, and the shadow register is updated with the new value.

38.3.5.2 Protecting Reads (Output Sample) with Coherency Checking

Starting from a coherent state (CIF is set), the software can read any of the other non-key bytes of the field. This action flags the field incoherent (clears the CIF). When a field is incoherent, it is protected against updates from the underlying hardware, and any new samples that may be generated while incoherent are dropped (without warning). The field remains flagged incoherent until the Key Coherency Byte is read. At this time, the field is flagged coherent (CIF is again set), and the next time the hardware generates a new output sample result, the field is updated.

38.3.6 Modes of Operation

This block has four primary operating modes:

- Single Sample
- Fast Filter
- Continuous
- Fast FIR

In Single Sample mode, the block sits in the standby state waiting for one of two start signals (START_CONV bit in CR register or ext_start). When a start is signaled, the block performs one sample conversion (four decimation periods where a decimation period is the count programmed in register DEC_DR1). It then captures the result, and signals the system by a polling or an interrupt that the process is complete and waits for the next signal as it reenters the standby state.

The Fast Filter mode captures single samples back to back, resetting itself and the Modulator between each sample. Upon completion of a sample, the next sample is initiated continuously. Polling and interrupts mark result events. Fast Filter mode is simply a continuous string of Single Samples with channel resets between them. This mode should be used when multiplexing channels.

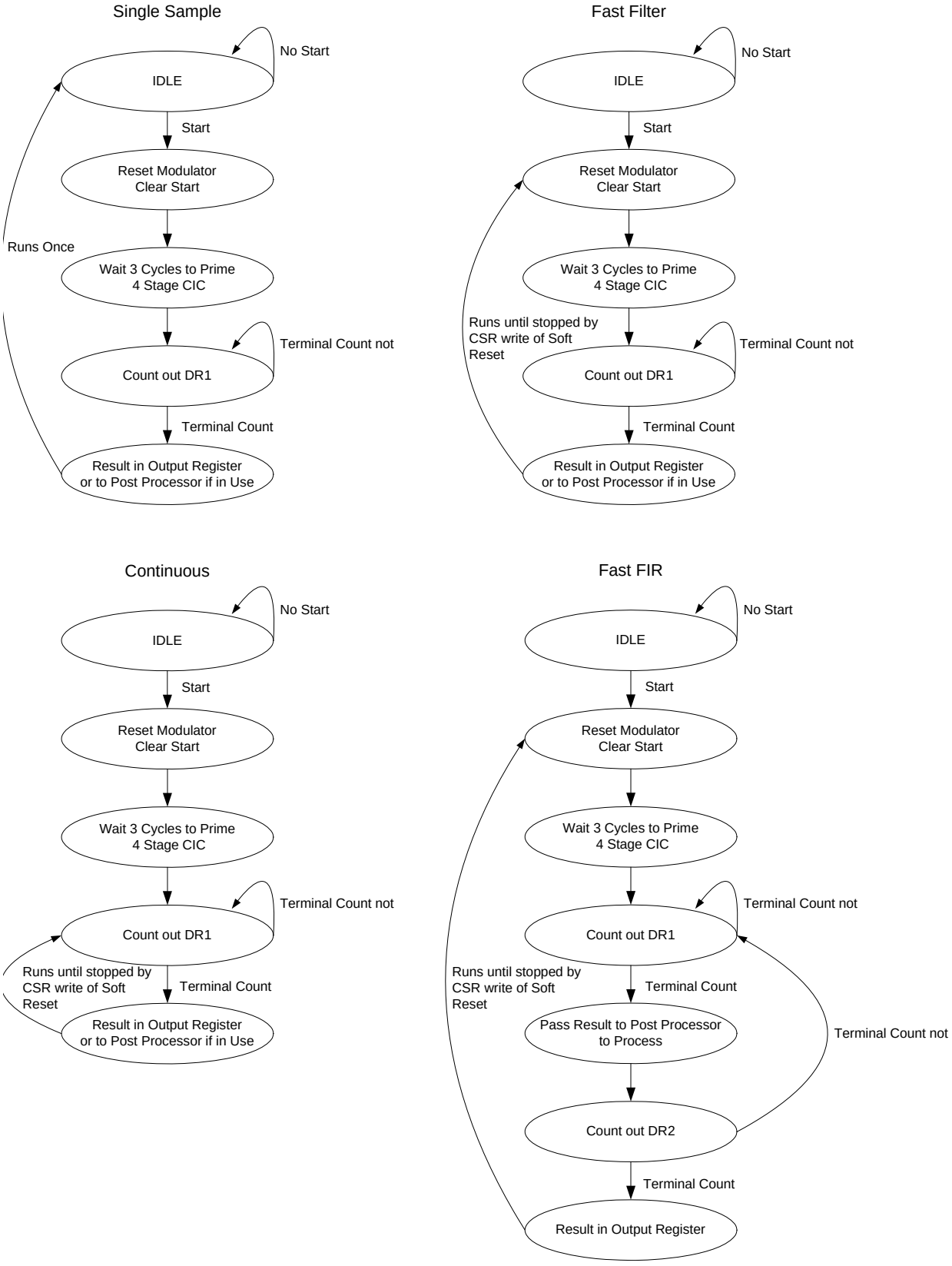
If signaled to run Continuous, the filter resets the channel then runs continuously from that point forward, until signaled to stop, with no intervening resets of the channel. The hardware blocks the first three decimation periods but then provides a result every decimation cycle thereafter.

Fast FIR mode is similar to Continuous mode, except that the ADC channel is reset and the filter restarted when the FIR decimation period (DR2) is reached. For example, if the DR2 register is set to 15 and this mode is selected, the filter:

- Resets the channel
- Blocks the first three decimation periods (DR1)
- Produces 16 samples for the FIR function to operate on
- Generates one output result
- Repeats this sequence until signaled to halt

The decimator is set to one of the four modes by DEC_CR[3:2] bits. All four modes are started by either a write to the start bit in the DEC_CR[0] register or an assertion of the input signal ext_start. Set the DEC_CR[1] register bit when using the external start feature. When set, this bit ignores the DEC_CR[0] start bit. Figure 38-10 on page 337 shows the state diagram of various modes of operation of the decimator.

Figure 38-10. Decimator Modes



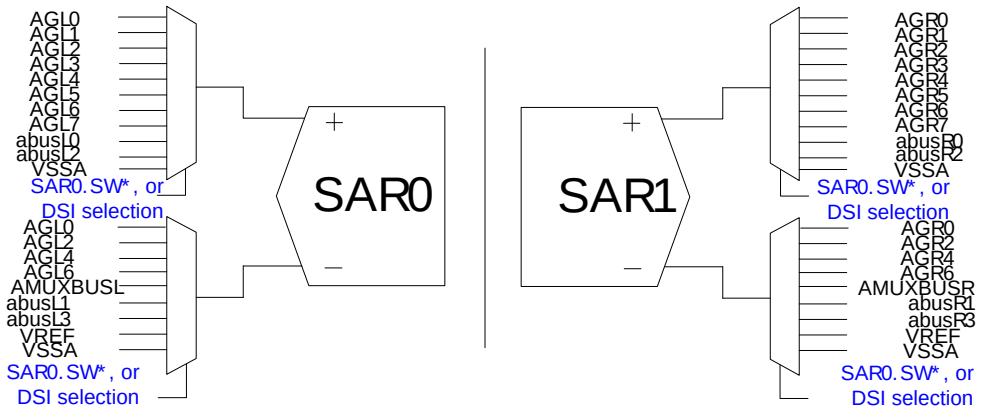
39. Successive Approximation Register ADC

The PSoC® 5 architecture has two successive approximation register analog to digital convertors (SAR ADC) in addition to the delta sigma ADC. The SAR ADC is designed for applications that require medium resolution and high data rate. The SAR ADC takes its input from the analog globals, locals, and the mux bus and the output can be taken from a register or be sent to the UDB for further processing.

39.1 Features

- 12-bit resolution
- Single ended, differential input
- Rail-to-rail input (0 V to Vdda)
- 700 KSPS sample rate
- Single shot or continuous running mode

Figure 39-1. SAR ADC Block Diagram



39.2 How It Works

39.2.1 Input Selection

The SAR ADC takes differential inputs, which are well connected to the analog routing structure. The positive input connects to analog globals, analog locals, and Vssa. The negative input connects to analog globals, analog locals, analog mux bus, voltage reference, and Vssa. The input selection, both positive and negative, is made through the input selection mux, which can be controlled through either the SAR routing registers in the analog interface or through the UDB. Setting the SARx_CSR1[4] bit takes the positive input through UDB and clearing the bit takes the positive input through registers. Similarly, setting the SARx_CSR1[3] bit takes the negative input through UDB and clearing the bit takes the negative input through registers. If the positive and negative input selection is made through the registers, registers SARx_SW0, SARx_SW2, SARx_SW3, SARx_SW4, and SARx_SW6 help in making the selection.

39.2.2 Clock Selection

The clock to the SAR can come from one of the four available analog clocks or a UDB generated clock. The clock selection for the SAR is made in SARx_CLK[2:0] register bits. The clock can be enabled or disabled through the gate control bit SARx_CLK[3]. The maximum input clock that can be applied to the SAR is 14 MHz. The digital output is synchronized with respect to the corresponding digitally aligned clock of the selected analog clock. This synchronization can be bypassed using SARx_CLK[4] register bit.

39.2.3 Input Sampling

The input sampling time can be programmed from 1 to 64 cycles in register SARx_CSR2[5:0] register bits. The user can also retain the earlier DAC value or clear it at the beginning of the new sampling clock. This is done in SARx_CSR0[3] register bit. The conversion time is 19 cycles for input sampling time up to four cycles. The maximum conversion time is 78 cycles for input sampling time of 64 cycles. The sampling time is chosen based on the source's input impedance so that the input settling time is lower than the sampling time.

39.2.4 Power Mode

The SAR ADC can be operated in maximum power mode.

39.2.5 Reference Selection

The SAR ADC can take either an internal or an external reference. The internal reference can be Vdda/2, 1.024 V, 1,2 V or DAC's output voltage. The reference selection is done in SAR_CSR1[7:5] register bits. For the vdda/2 reference selection be available, SAR_CSR3[6] register bit must be set.

Table 39-1. SAR Analog Reference Modes

SN	Mode	Switch States													
		S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
1	External Reference			ON	OFF	OFF			OFF						
2	Internal Reference with External Capacitor			ON	OFF	OFF			OFF						
3	Internal Reference without External Capacitor			OFF	ON	ON			OFF						
4	Vda as Reference Voltage			OFF	OFF	ON			ON						

39.2.6 Operational Modes

The SAR can be configured in two modes, single capture or continuous. In single capture, the SAR ADC completes one conversion on a trigger; in the continuous mode the SAR ADC performs continuous conversion. The trigger can be either software or hardware. The software trigger comes from SARx_CSR0[0] register bit and the hardware trigger is from the UDB. The selection between software and UDB trigger is made in SARx_CSR0[2] register bit.

As long the SOF stays high the conversion continues, the conversion stops when the SOF goes low.

The two modes, single capture and continuous, is realized in the way the SOF bit is configured - level or edge sensitive SOF. In the level sensitive mode, the SAR ADC performs the conversion as long as the SOF bit is asserted high. So, the level sensitive mode is used for continuous conversion.

In the edge sensitive mode, the SAR performs a conversion on the edge and the bit is automatically reasserted low on the completion of the conversion (on the end of frame (EOF)). So, it must be reasserted high for the next edge for the SAR ADC to start conversion. This mode helps in performing single sample conversions.

In case of hardware enabled SOF, the user can synchronize the conversion to a PWM frequency by configuring it in the edge mode.

The level or edge triggered function of the SOF signal is configured in the SARx_CSR0[1] register bit.

The conversion time of the SAR is more than 19 cycles for input sampling time of four cycles when hardware trigger is used. This is because SOF and EOF are routed through DSI routing and these signals encounter a delay, resulting in longer conversion

39.2.7 SAR ADC Output

The SAR ADC output includes:

- End of Frame (EOF) bit
- The output bits of user configured resolution
- An optional interrupt on EOF

The resolution can be configured to 12 bits in SARx_CSR2[7:6] register bits. When a conversion is complete the End of Frame (EOF) bit is asserted high in SARx_CSR1[0] register bit. This bit is a clear on read sticky status bit and is cleared automatically on a data read. The conversion result is stored in the registers SARx_WRK0 and SARx_WRK1 register. The SARx_WRK1 register bits [3:0] stores the higher four bits [11:8] of the output. Coherency protection can be applied to the SAR output by setting SARx_CSR0[4] register bit. It ensures that a new output is written only when both the registers are read.

The EOF output can be used to generate interrupt to the CPU or DMA. The interrupt is enabled by setting the SARx_CSR1[1] register bit. The interrupt can be made edge/ level interrupt by setting/clearing SARx_CSR1[2] register bit.

Table 39-2. SAR Connections

SAR0	Decode Table		
VP	Connection	VN	Connection
0	AGL[0]	0	AGL[0]
1	AGL[1]	1	AGL[2]
2	AGL[2]	2	AGL[4]
3	AGL[3]	3	AGL[6]
4	AGL[4]	4	AMUXBUSL
5	AGL[5]	5	ABUSL[1]
6	AGL[6]	6	ABUSL[3]
7	AGL[7]	7	VREFHI
8	ABUSL[0]	8	VSSA
9	ABUSL[2]	9	Hi-Z (N/C)
A	VSSA	A	Hi-Z (N/C)
B	Hi-Z (N/C)	B	Hi-Z (N/C)

Table 39-2. SAR Connections

SAR0	Decode Table		
VP	Connection	VN	Connection
C	Hi-Z (N/C)	C	Hi-Z (N/C)
D	Hi-Z (N/C)	D	Hi-Z (N/C)
E	Hi-Z (N/C)	E	Hi-Z (N/C)
F	Hi-Z (N/C)	F	Hi-Z (N/C)

Section G: Program and Debug

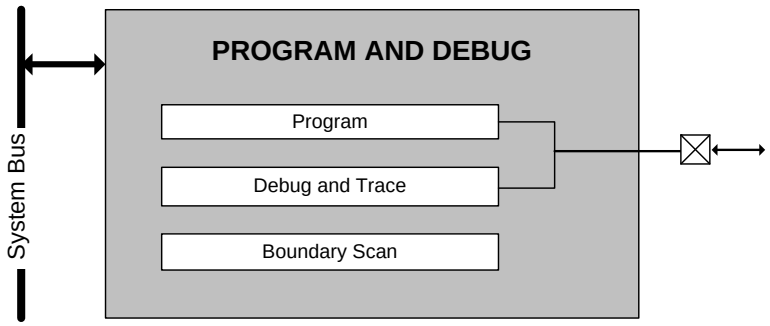
Serial Wire Debugger (SWD) (2 wire) interface is used for programming and debugging PSoC 5. The 1-wire Single Wire Viewer (SWV) can also be used for “printf” style debugging. By combining SWD and SWV, the designer can implement a full debugging interface with just three pins. Using these standard interfaces enables the designer to debug or program the PSoC® device with a variety of hardware solutions from Cypress or third-party vendors.

This section encompasses the following chapters:

- Test Controller chapter on page 345
- Cortex-M3 Debug and Trace chapter on page 351
- Nonvolatile Memory Programming chapter on page 357

Top Level Architecture

Program and Debug Block Diagram



41. Test Controller

The PSoC® 5 architectures include a test controller used to access the device memory and registers (via the PHUB) through the PSoC 5 Cortex-M3 Debug Access Port (DAP) for functional testing, device programming, and program debugging.

The test controller connects to off-chip devices via the Serial Wire Debug (SWD) interface. This interface uses I/O port pins (SWDIO, SWDCK).

41.1 Features

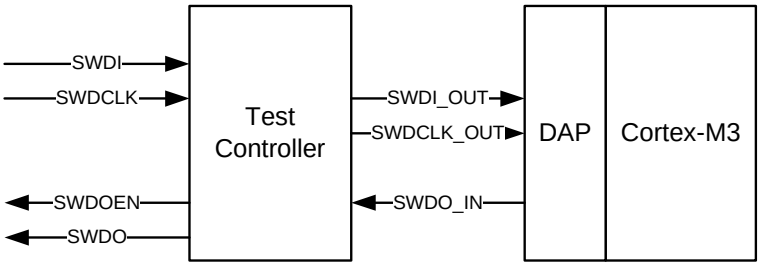
The test controller has the following features:

- Supports SWD interface to a debug host
- SWD interface available on either GPIO or USB pins
- Interfaces to PSoC 5 debug modules for access to the rest of the device for program and debug operations

41.2 Block Diagram

Figure 41-1 shows the Test Controller architecture in PSoC 5.

Figure 41-1. PSoC 5 Test Controller Block Diagram



41.3 Background Information

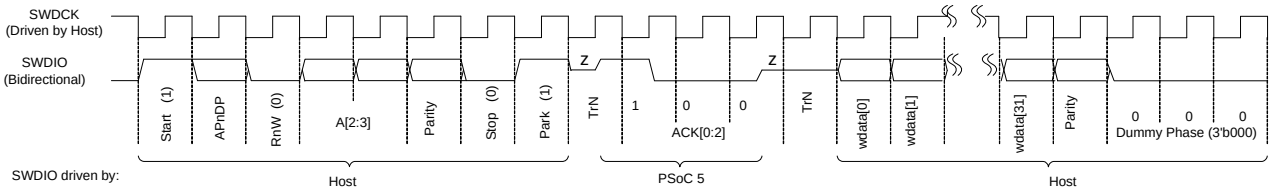
The following information is provided to familiarize the user with the SWD interface.

41.3.1 Serial Wire Debug Interface

41.4 Serial Wire Debug (SWD) Interface

PSoC 5 supports programming through the serial wire debug (SWD) interface. There are two signals in SWD interface: data signal (SWDIO) and a clock for data signal (SWDCK). The host programmer always drives the clock line, whereas either the programmer or the PSoC 5 device drives the data line. Host programmer and PSoC 5 device communicate in packet format through the SWD interface. Write packet refers to the SWD packet transaction in which the host writes data to PSoC 5. Read packet refers to the SWD packet transaction in which the host reads data from PSoC 5. The format of the write packet and read packet are illustrated in Figure 41-2 and Figure 41-3, respectively

Figure 41-2. SWD “Write Packet” Timing Diagram



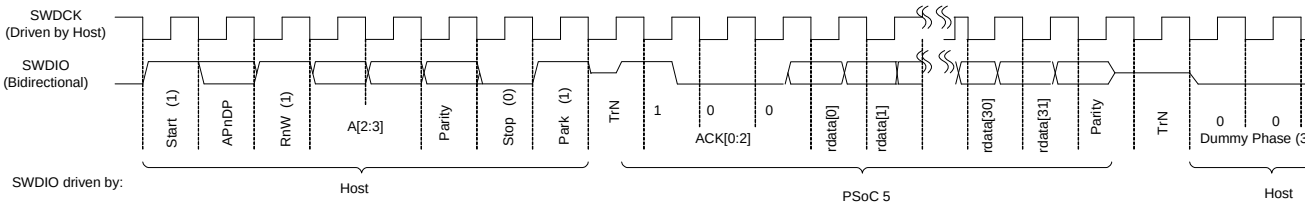
a.) Host Write Operation: Host sends data on the SWDIO line on falling edge of SWDCK and PSoC 5 reads that data on the next SWDCK rising edge (for example, 8-bit header data, Write data(wdata[31:0]), Dummy phase (3'b000))

b.) Host Read Operation: PSoC 5 sends data on the SWDIO line on the rising edge of SWDCK and the host should read that data on the next SWDCK falling edge (Ex: ACK data (ACK[2:0])

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 5 starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 5 and the Host will not drive the line during the entire second TrN phase (indicated as ‘z’). Host should start sending the Write data (wdata) on the next falling edge of SWDCK after second TrN phase.

d.) “DUMMY” phase is three SWD clock cycles with SWDIO line low. This DUMMY phase is not part of SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Figure 41-3. SWD “Read Packet” Timing Diagram



a.) Host Write Operation: Host sends data on the SWDIO line on falling edge of SWDCK and PSoC 5 reads that data on the next SWDCK rising edge (for example, 8-bit header data, dummy phase (3'b000))

b.) Host Read Operation: PSoC 5 sends data on the SWDIO line on rising edge of SWDCK and the host should read that data on the next SWDCK falling edge (for example, ACK data (ACK[2:0], Read data (rdata[31:0]))

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 5 starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 5 and the host will not drive the line during the entire second TrN phase (indicated as ‘z’). Host should start sending the Dummy phase (3'b000) on the next falling edge of SWDCK after second TrN phase.

d.) “DUMMY” phase is three SWD clock cycles with SWDIO line low. This phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

A complete data transfer requires 46 clocks (not including the optional three dummy clock cycles in Figure 41-2 and Figure 41-3). Each data transfer consists of three phases:

- **Packet request** – External host programmer issues a request to the PSoC 5 device.
- **Acknowledge response** – PSoC 5 sends an acknowledgement to the host.
- **Data** – Data is valid only when a packet request is followed by a valid (OK) acknowledge response.

The data transfer is either:

- PSoC 5 to host, following a read request – RDATA
- Host to PSoC 5, following a write request – WDATA

In Figure 41-2 and Figure 41-3, the following sequence occurs:

1. The start bit initiates a transfer; it is always logic ‘1’.
2. The APnDP bit determines whether the transfer is an AP access, ‘1’, or a DP access, ‘0’.
3. The next bit is RnW, which is ‘1’ for a read from the PSoC 5 device or ‘0’ for a write to the PSoC 5 device.

4. The ADDR bits (A[3:2]) are register select bits for the access port or debug port. See Table 41-2 for address bit definitions.
5. The parity bit has the parity of APnDP, RnW, and ADDR. This is an even parity bit. If the number of logical 1s in these bits is odd, then parity must be ‘1’, otherwise it is ‘0’.

   If the parity bit is not correct, the header is ignored by the target device; there is no ACK response. For the host implementation, the programming operation should be stopped and tried again by doing a device reset.
6. The stop bit is always logic ‘0’.
7. The park bit is always logic ‘1’ and should be driven high by the host.
8. The ACK bits are the device-to-host response.

   Possible values are shown in Table 41-1. Note that the ACK in the current SWD transfer reflects the status of the previous transfer. OK ACK means the previous packet is successful. WAIT response indicates that the previous packet transaction is not yet complete. For a Fault operation, the programming operation should be aborted immediately.

Table 41-1. ACK Response for SWD Transfers

ACK[2:0]	SWD
OK	001
WAIT	010
FAULT	100

   a. For a WAIT response, if it is a read transaction, the host should ignore the data read in the data phase. PSoC 5 does not drive the line and the host must not check the parity bit.
   b. For a WAIT response, if it is a write transaction, the data phase is ignored by the PSoC 5 device. But the host must still send the data to be written from an implementation standpoint. The parity data corresponding to the data should also be sent by the host.
   c. For a WAIT response, it means that the PSoC 5 device 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, failing which, it can abort the programming operation and try again.
   d. For a FAULT response, the programming operation should be aborted and retried by doing a device reset.
9. The data phase includes a parity bit (even parity, similar to the packet request phase).
   a. For a read data packet, if the host detects a parity error, then it must abort the programming operation and restart.
   b. For a write data packet, if the PSoC 5 detects a parity error in the data packet sent by the host, it generates a FAULT ACK response in the next packet.
10. Turnaround (TrN) phase: According to the SWD protocol, the TrN phase is used both by the host and the PSoC 5 device to change the Drive modes on their respective SWDIO line. There are two TrN phases in each SWD packet. During the first TrN phase after packet request, PSoC 5 drives the ACK data on the SWDIO line on the rising edge of SWDCK in TrN phase. This ensures that the host can read the ACK data on the next falling edge. Thus, the first TrN cycle is only for half cycle duration, as shown in Figure 41-2 and Figure 41-3. The location of the second TrN phase is different for read and write packets. The second TrN phase of the SWD packet is one-and-a-half cycle long. Neither the host nor PSoC 5 should drive SWDIO line during both the TrN phases as indicated by ‘z’ in Figure 41-2 and Figure 41-3.
11. The address, ACK, and read and write data are always transmitted least significant bit (lsb) first.
12. At the end of each SWD packet in Figure 41-2 and Figure 41-3, there is a “DUMMY” phase, which is three SWD clock cycles with SWDIO line held low. This DUMMY phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Note The SWD interface can be reset anytime during programming by clocking 51 or more cycles with SWDIO high. To return to the idle state, SWDIO must be clocked low for three or more cycles. The host programmer can begin a new SWD packet transaction from the idle state.

41.5 How It Works

The PSoC 5 SWD interface complies with standard specifications and offers extensions unique to PSoC 5 architecture.

41.5.1 Clocking

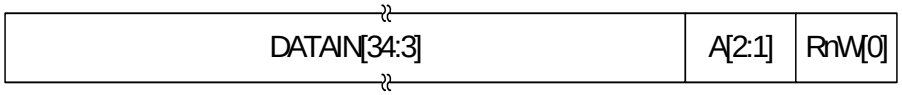
The data signal clock (SWDCK) for SWD interface shares the same I/O pin (P1[1]). (An alternate SWDCK can be input on the USB D- pin, P15[7].) Clocking limits apply to both clocks in either mode.

41.5.2 DP/AP Access Register

PSoC 5 architecture has a DP/AP Access register that is 35 bits wide. This register is used to transfer data between the SWD interface and the Debug Port and Access Port registers. The SWD interface enables direct reads and writes of the DP/AP Access register.

When writing the register from the SWD interface, the structure is as shown in the following figure.

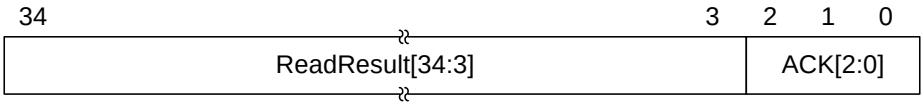
Figure 41-4. Writing the Register



- **Bits 34 to 3** – 32 bits of data – If the Port register is less than 32 bits wide, only the N LS bits are transferred, where N is the width of the Port register.
- **Bits 2 to 1** – 2-bit address for Debug or Access Port register select – In PSoC 5 devices, it is transferred to bits [3:2] of the register select; bits [1:0] are 0.
- **Bit 0 – RnW** – 1 = read (from device to debug host); 0 = write (to device from debug host)

When reading the register from the SWD interface, the structure is as shown in the following figure.

Figure 41-5. Reading the Register



- **Bits 34 to 3** – 32 bits of data – If the Port register is less than 32 bits wide, only the N LS bits are transferred, where N is the width of the port register.
- **Bits 2 to 0** – ACK response code – The ACK response is as indicated in Table 41-1.

41.5.3 Debug Port and Access Port Registers (PSoC 5)

The registers listed in Table 41-3 are part of the ARM Cortex-M3 Debug Access Port (DAP). In the PSoC 5 Cortex-M3, the DAP consists of the SWD Debug Port (SW-DP) and the AHB Access Port (AHB-AP).

For further information about these ports and their registers, see the *ARM Debug Interface Architecture Specification* (for the SWJ-DP), and the *ARM Cortex-M3 Technical Reference Manual* (for the AHB-AP), both available at http://www.arm.com.

Table 41-2. Debug Port and Access Port Registers (PSoC 5)

Name	Instruction (AP/DP)	Address (Register Select)	Function
DP CTRL/STAT	DPACC	01	Debug Port control/status register
SELECT	DPACC	10	Access port select – The MS byte of the SELECT register selects which Access Port (AP) is used on AP accesses. To select the Cortex-M3 AHB-AP, this byte should always be 0x03. Bits [7:4] select which register in the AHB-AP is accessed.
RDBUFF	DPACC	11	Returns the result of the last AP read access, without the need to start a new AP access operation.
AP Control Status	APACC	00 (SELECT[7:4] = 0)	AHB-AP control/status register
AP Transfer Address	APACC	01 (SELECT[7:4] = 0)	AHB-AP transfer address register
AP Data Read/Write	APACC	11 (SELECT[7:4] = 0)	AHB-AP data read/write register

Table 41-2. Debug Port and Access Port Registers (PSoC 5) *(continued)*

Name	Instruction (AP/DP)	Address (Register Select)	Function
AP Banked Data 0	APACC	00 (SELECT[7:4] = 1)	AHB-AP banked data register
AP Banked Data 1	APACC	01 (SELECT[7:4] = 1)	AHB-AP banked data register
AP Banked Data 2	APACC	10 (SELECT[7:4] = 1)	AHB-AP banked data register
AP Banked Data 3	APACC	11 (SELECT[7:4] = 1)	AHB-AP banked data register
AP Debug ROM Address	APACC	10 (SELECT[7:4] = 0xF)	AHB-AP debug ROM address register (read only)
AP Identification Register	APACC	11 (SELECT[7:4] = 0xF)	AHB-AP ID register (read only)

41.6 Test Controller Acquisition

The only way to gain debug access to the part is to enter a valid port acquire key within a key window period of 8 µs after reset (8 µs is only the initial window, it extends to 400 µs if 8 clocks are sampled in 8 µs). The port acquire key must be transmitted over one of the two SWD pin pairs, as indicated in the following table.

Table 41-3. SWD Pin Pairs

SWD Pin Pair	SWDIO	SWDCK
Standard	P1[0]	P1[1]
Alternate	P15[6] (USB D+)	P15[7] (USB D-)

The SWD packet request phase consists of:

- APnDP = 0
- RnW = 0
- ADDR = 11
- WDATA = 0x7B0C06DB with WDATA Parity = 0

The SWD frame should be transmitted at least twice, ignoring the ACK on the first transmit; it should be transmitted until the ACK response is OK. The SWD interface will be in the idle state, ready for the next write. Refer to the PSoC 5 Device Programming Specifications for detailed timing diagrams on test controller acquisition.

41.6.1 Functional Test

To perform a functional test:

1. At reset, assume that the pins state is unknown.
2. Do a port acquire within the key window, which enables the SWD interface.
3. Use the SWD interface.
4. Start writing or reading PSoC 5 devices to or from registers and memory for functional tests.

41.6.2 Programming Flash/EEPROM

To program Flash or EEPROM:

1. At reset, assume that the pins state is unknown.
2. Do a port acquire within the key window, which enables the SWD interface.
3. Use the SWD interface.
4. Start writing or reading to or from PSoC 5 SPC registers to program Flash/EEPROM. Refer to the Nonvolatile Memory Programming chapter on page 357 for details.

41.6.3 Program Debug/Trace

To perform a debug or trace:

1. At any time during normal operation, use the SWD interface.
2. Start writing or reading to or from PSoC 5 Cortex-M3 DAP debug module registers to do program debug/trace operations.

42. Cortex-M3 Debug and Trace

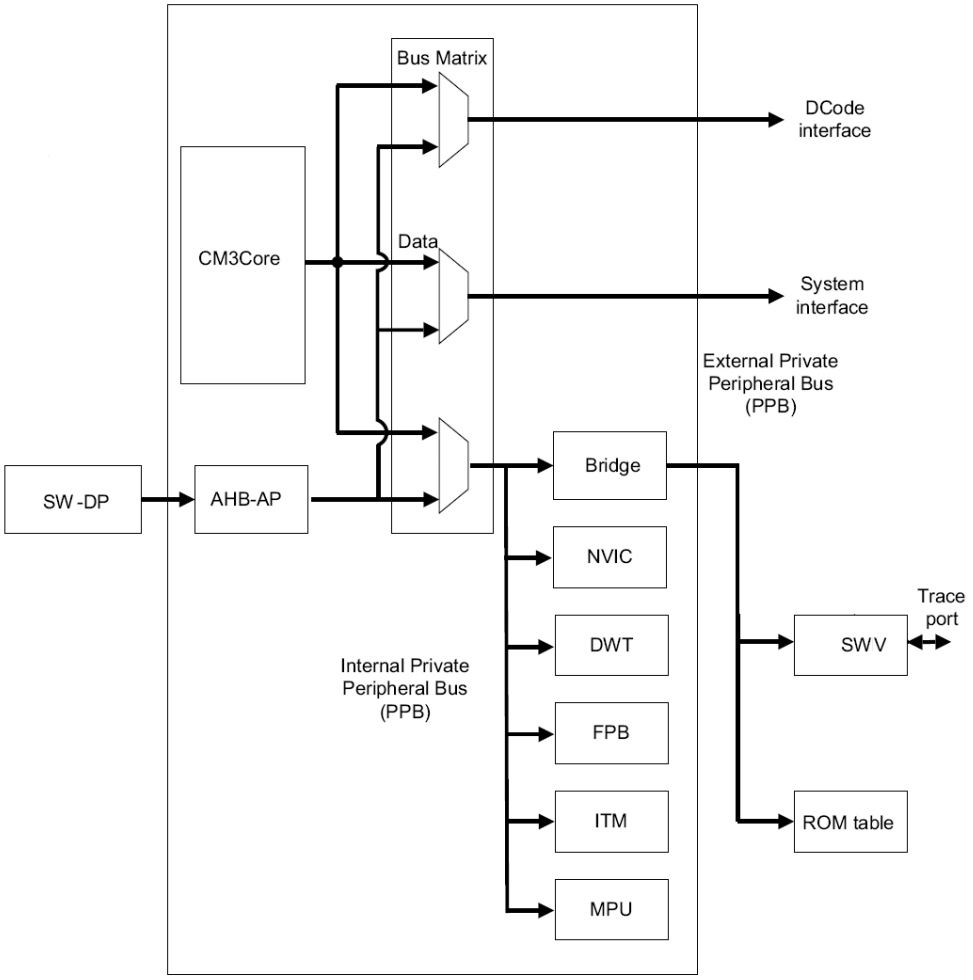
The PSoC® platform provides extensive support for programming, testing, debugging, and tracing both hardware and firmware. PSoC 5 supports two interfaces: SWD and SWV. Cortex-M3 debug and trace functionality enables full device debugging in the final system using the standard production device.

Cortex-M3 debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, data watchpoints, register value access, and ROM-based debugging. Noninvasive debugging includes memory access, data trace, and software trace.

42.1 Features

- Debug access to all memory and registers in the system, including Cortex-M3 register bank when the core is running or halted.
- SWD access.
- Flash Patch and Breakpoint (FPB) block for implementing breakpoints and code patches.
- Data Watchpoint and Trace (DWT) block for implementing watchpoints, trigger resources, and system profiling.
- Instrumentation Trace Macrocell (ITM) for support of printf style debugging.
- Support for six breakpoints and four watchpoints.

Figure 42-1. Debug and Trace Block Diagram



Debug control and data access occurs through the Advanced High-performance Bus-Access Port (AHB-AP) interface. This interface is driven by the Serial Wire Debug Port (SW-DP) component.

Through internal PPB, the debugger can access:

- Nested Vectored Interrupt Controller (NVIC). Debug access to the processor core is made through the NVIC.
- DWT
- FPB
- ITM

Through external PPB, the debugger can access:

- Serial Wire Viewer for printf style debugging

Through the DCode bus, the debugger can access memory located in the code space. The system bus provides access to bus, memory, and peripherals located in the system bus space.

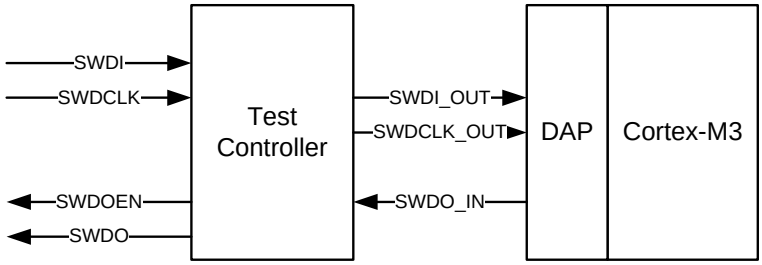
42.2 How It Works

The PSoC 5 SWD interface complies with standard specifications and offer extensions unique to PSoC 5 architecture.

42.2.1 Test Controller (TC)

The Test Controller is used to access the device memory and registers (via the PHUB) through PSoC 5 Cortex-M3 Debug Access Port (DAP) for functional testing, device programming, and program debugging.

Figure 42-2. PSoC 5 Test Controller interface



42.2.1.1 Debug Port and Access Port Registers

The registers are part of the ARM Cortex-M3 Debug Access Port (DAP). In the PSoC 5 Cortex-M3, the DAP consists of the SWD Debug Port (SW-DP) and the AHB Access Port (AHB-AP). The registers are listed in Table 41-3 on page 349.

For further information about these ports and their registers, see the ARM Debug Interface Architecture Specification (for the SW-DP), and the ARM Cortex-M3 Technical Reference Manual (for the AHB-AP), both available at http://www.arm.com.

42.2.2 SWV Interface

The SWV interface provides trace data to a debug host via the Cypress MiniProg3 or an external trace port analyzer.

42.3 Core Debug

Core debug allows users to exercise features such as enabling debug, halting, stepping, and accessing the PSoC memory and registers. Core debug is accessed through the core debug registers. The main core debug registers are:

- Debug Halting Control and Status Register (DHCSR)
- Debug Exception and Monitor Control Register (DEMCR)
- Debug Core Register Data Register (DCRDR)
- Debug Core Register Selector Register (DCRSR)

Among these the Debug Halting Control and Status Register allows enabling the core debug, providing status information about the state of the processor and halting and stepping the processor. More details about these registers can be found in the ARM Cortex-M3 Technical Reference Manual, available at http://www.arm.com.

42.3.1 Enabling the Debug

The core debug can be enabled by setting the C_DEBUGEN bit of the Debug Halting Control and Status Register.

Note that when the debug controller is enabled, it can read the entire flash memory regardless of the flash protection setting. Therefore, if flash protection is required, the debug controller also needs to be disabled.

42.3.2 Halting

The debugger can halt the core by setting the C_DEBUGEN and C_HALT bits of the Debug Halting Control and Status Register. The core acknowledges when halted by setting the S_HALT bit of the Debug Halting Control and Status Register.

42.3.3 Stepping

The core can be single stepped by halting the core, setting the C_STEP bit to ‘1’, and then clearing the C_HALT bit to ‘0’. The core acknowledges completion of the step and rehalts by setting the S_HALT bit of the Debug Halting Control and Status Register.

The core can exit halting debug by clearing the C_DEBUGEN bit in the Debug Halting Control and Status Register.

42.3.4 Accessing PSoC Memory and Registers

The Debug Core Register Data Register (DCRDR) and Debug Core Register Selection Register (DCRSR) are used for accessing the PSoC memory and registers. The register and memory access are 32 bits wide.

To use the registers to read the contents of a register, the following steps should be performed:

1. Set the C_DEBUGEN and C_HALT bits of the Debug Halting Control and Status Register. This enables the debug and halts the core.

2. Wait for the S_HALT bit of the Debug Halting and Status Register to be set. This indicates that the core is halted.
3. Write to the DCRSR with bit 16 set to ‘0’, indicating it is a read operation.
4. Poll until the S_REGRDY bit in DHCSR is ‘1’.
5. Write the register number to be read into the Debug Core Register Selector Register.
6. Read the value from the Debug Core Register Data Register.

To write to a register, the following steps should be performed:

1. Make sure the processor is halted by following steps 1 and 2 mentioned above.
2. Write data value to the DCRDR.
3. Write to the DCRSR with bit 16 set to ‘1’, indicating it is a write operation.
4. Write the register number that you want to write to into the DCRSR.
5. Poll until the S_REGRDY bit in DHCSR is ‘1’. When the bit becomes ‘1’, the write operation is complete.

The Memory Access Port (MEM-AP) provides access to the memory through the DAP. All accesses to a MEM-AP are made through the MEM-AP registers. All registers are 32 bits wide. The important registers required for memory access include:

- Control/Status Word Register (CSW) - The CSW Register configures and controls accesses through the MEM-AP to or from a connected memory system.
- Transfer Address Register (TAR) - The TAR holds the memory address to be accessed.
- Data Read/Write Register (DRW) - The DRW holds a 32-bit data value. In write mode, the DRW holds the value to write for the current transfer to the address specified in TAR[31:0]. In read mode, the DRW holds the value read in the current transfer from the address specified in TAR[31:0].
- Configuration Register (CFG) - The CFG Register provides information about the configuration of the MEM-AP implementation. It indicates whether memory accesses by the MEM-AP are big-endian or little-endian.
- Debug Base Address Register (BASE) - The BASE Register provides an index into the connected memory mapped resource. This index value points to one of the following, the start of a set of debug registers or a ROM table that describes the connected debug components.

For more details about the Memory Access Port and registers, refer to the ARM Debug Interface Architecture Specification, available at http://www.arm.com.

42.4 System Debug

The processor contains several system debug components that facilitate low cost debug, trace and profiling, breakpoints, watchpoints and code patching.

The system debug components are:

- Flash Patch and Breakpoint (FPB) unit to implement breakpoints and code patches.
- Data Watchpoint and Trace (DWT) unit to implement watchpoints, trigger resources, and system profiling.
- Instrumentation Trace Macrocell (ITM) for application-driven trace source that supports printf style debugging.

42.4.1 Flash Patch and Breakpoint (FPB) Unit

The main functions of the FPB are:

- Implement hardware breakpoint (generates a breakpoint event to the processor to invoke debug modes such as halt or debug monitor).
- Patch instruction or data from code memory space to SRAM.

The FPB unit contains:

- Two comparators for matching against literal loads from code space, and remapping to a corresponding area in system space.
- Six instruction comparators for matching against instruction fetches from code space, and remapping to a corresponding area in system memory space. Alternatively, it is possible to individually configure the comparators to return a Breakpoint Instruction (BKPT) to the processor core upon a match, providing hardware breakpoint capability.

The FPB has a flash patch control register that contains an enable bit to enable the FPB. In addition, each comparator comes with a separate enable bit in its comparator control register. Both of the enable bits must be set to ‘1’ for a comparator to operate. If the comparison for an entry matches, the address is remapped to the address set in the remap register plus an offset corresponding to the comparator that matched, or is remapped to a BKPT instruction, if that feature is enabled.

42.4.2 Data Watchpoint and Trace (DWT)

The DWT has a number of debugging functionalities. It has four comparators, each of which can be configured as follows:

- Hardware watchpoint (generates a watchpoint event to processor to invoke debug modes such as halt or debug monitor)
- PC sampler event trigger
- Data address sampler trigger
- The first comparator can also be used to compare against the clock cycle counter instead of comparing to a data address

The DWT also has counters for counting:

- Clock cycles (CYCCNT)
- Folded Instructions: A folded instruction is one that does not incur even one cycle to execute
- Load Store Unit (LSU) Operations: LSU counts include all LSU costs after the initial cycle for the instruction
- Sleep cycles
- Cycles per instruction (CPI)
- Interrupt overhead
- PC sampling at regular intervals to count the number of core cycles
- Applications and debuggers can use the counter to measure elapsed execution time
- Interrupt events trace

When used as a hardware watchpoint, the comparator can be programmed to compare either data addresses or program counters. Otherwise, it compares the data addresses.

42.4.3 Instrumentation Trace Macrocell (ITM)

The ITM is a an application driven trace source that supports printf style debugging to trace Operating System (OS) and application events, then emit diagnostic system information. The ITM emits trace information as packets. There are three sources that can generate packets. If multiple sources generate packets at the same time, the ITM arbitrates the order in which packets are output. The three sources in decreasing order of priority are:

- Software Trace. Software can write directly to ITM stimulus registers. This emits packets.
- Hardware Trace. The DWT generates these packets, and the ITM emits them.
- Time Stamping. The ITM can generate timestamp packets that are inserted into a trace stream to help the debugger find out the timing of events. The ITM contains a 21-bit counter to generate the timestamp. The Cortex-M3 clock or the bit clock rate of the Serial Wire Viewer (SWV) output clocks the counter.

One of the main uses of the ITM is to support printf style debugging. The ITM contains 32 stimulus ports, allowing different software processes to output to different ports, and messages that can be separated later at the debug host. Each port can be enabled or disabled by the Trace Enable Register (SWV_ITM_TER) and can be programmed (in groups of eight ports) to allow or disallow user processes to write to it. The output messages can be collected at the trace port interface or the Serial Wire Viewer (SWV) on the TPIU.

The ITM is used in output of hardware trace packets. The packets are generated from the DWT and the ITM acts as a trace packet merging unit. To use DWT trace, you need to enable the DWTEn bit in the ITM Control Register (SWV_ITM_CR).

ITM has a timestamp feature that allows trace capture tools to find out timing information by inserting delta timestamp packets into the traces when a new trace packet enters the FIFO inside the ITM. The timestamp packet is also generated when the timestamp counter overflows.

The timestamp packets provide the time difference (delta) with previous events. Using the delta timestamp packets, the trace capture tools can then establish the timing of when each packet is generated and hence reconstruct the timing of various debug events.

42.4.4 Single Wire Viewer (SWV)

Single Wire viewer (SWV) allows target resident code to communicate diagnostic information to the outside world through a single pin. The Serial Wire Viewer block is a combination of the Instrumentation Trace Macrocell (ITM) and the Serial Wire Output (SWO). ITM is a software application trace source.

The SWV's trace output (TRACESWO) is channeled through the Test Controller, so that the Test Controller can output the trace data over the SWO pin when SWD is enabled.

SWV can only be used when the Serial Wire Debug (SWD) is enabled.

42.4.4.1 Enabling SWV

The Trace Enable Register (SWV_ITM_TER) is used to enable the stimulus ports so that trace data can be written into the stimulus port registers. Each bit in the Trace Enable Register is set to enable the corresponding stimulus port register. Also, the ITM should be enabled using the global enable bit, ITMEn, in the Control Register (SWV_ITM_CR).

42.4.4.2 Communicating with SWV

Trace data is written into the stimulus port registers (SWV_ITM_SPR_DATA [0…31]). Each of the 32 stimulus ports has its own address. A write to one of these locations causes data to be written into the FIFO if the corresponding bit in the Trace Enable Register is set. Reading from any of the stimulus ports returns the FIFO status. A '0' is returned if the FIFO is full and a '0' is returned otherwise, only if the bit in the Trace Enable Register is set.

42.4.5 Using Multiple Interfaces Simultaneously

If debugging and tracing are done at the same time, then SWD may be used with SWV

43. Nonvolatile Memory Programming

PSoC® 5 devices have three types of nonvolatile memory: Flash, Electronically Erasable Programmable Read Only Memory (EEPROM), and Write Once Nonvolatile Latch (WO NVL). These can all be programmed by either the CPU running a boot loader program or by an external system via the SWD interface. See AN64359 - PSoC 5 Programming Specifications, for details about device programming and programming specifications.

43.1 Features

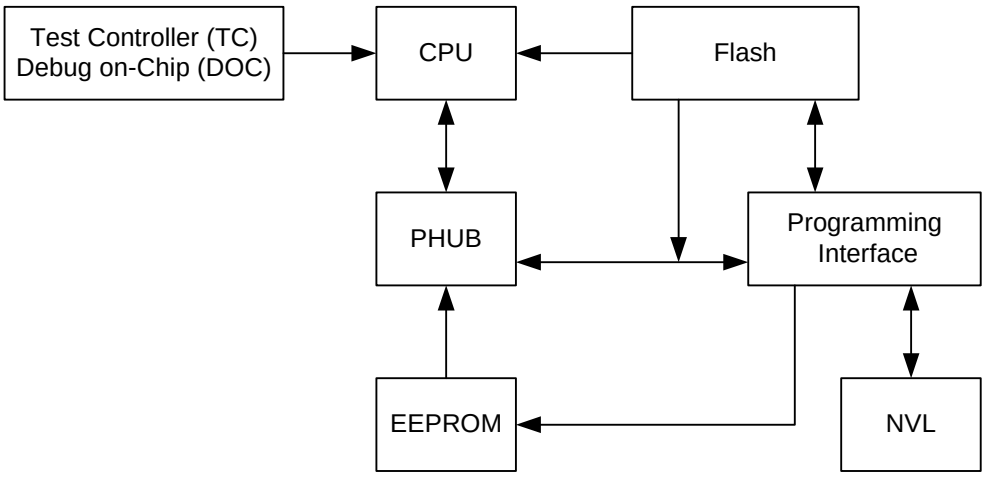
The nonvolatile memory programming system has the following features:

- Simple command/status register interface
- Flash can be written at the 288-byte row level
- Each row of Flash has 256 bytes of data plus an additional 32 bytes for configuration
- EEPROM can be written both at the byte level and at the 16-byte row level
- WO NVL can be programmed under limited programming voltage and temperature conditions

43.2 Block Diagram

Figure 43-1 is a block diagram of the Flash programming system.

Figure 43-1. Flash Block Diagram



43.3 How It Works

All programming operations are done through a simple command/status register interface summarized in Table 43-1.

Table 43-1. Command and Status Register Summary

Register	Size (Bits)	Description
SPC_CPU_DATA	8	Data to/from the CPU
SPC_DMA_DATA	8	Data to/from the DMAC
SPC_SR	8	Status – ready, data available, status code

Commands and data are sent as a series of bytes to either SPC_CPU_DATA or SPC_DMA_DATA, depending on the source of the command. Response data is read via the same register to which the command is sent. The status register, SPC_SR, indicates whether a new command can be accepted, when data is available for the most recent command, and a success/failure response for the most recent command.

43.3.1 Commands

Before sending a command to the SPC_CPU_DATA or SPC_DMA_DATA register, the SPC_Idle bit in SPC_SR[1] must be '1'. SPC_Idle will go to '0' when the first byte of a command (0xB6) is written to a data register, and go back to '1' when command execution is complete or an error is detected. Commands sent to either data register while SPC_Idle is '0' are ignored. All commands must adhere to the following format:

- Key byte #1 – always 0xB6
- Key byte #2 – 0xD3 plus the command code (ignore overflow)
- Command code byte
- Command parameter bytes
- Command data bytes

The command codes are shown in Table 43-2. See 43.3.1.1 Command Code Descriptions on page 359 for details.

Table 43-2. Command Codes

Command Code	Command Name	Memory Type	Access	Description
0x00	Load byte	NVL	Any	Loads a single byte of data into the volatile latch
0x01	Load multi bytes	Flash, EEPROM	Any	Loads 1 to 32 bytes of data into the row latch
0x02	Load row	Flash, EEPROM	Any	Loads a row of data
0x03	Read byte	NVL	Any	Read a byte from NV memory
0x04	Read multi bytes	Flash, EEPROM	TC only	Reads 1 – 256 data bytes, does not cross row boundaries
0x05	Write row	Flash, EEPROM	Any	Erases then programs a row with data in row latch
0x06	Write NVL	NVL	TC only	Programs all of user NVL with data in the volatile latch
0x08	Erase sector	Flash, EEPROM	Any	Erases a 64-row sector
0x09	Erase all	Flash	TC only	Erases all Flash, including row protection bytes
0x0B	Protect	Flash	TC only	Program Flash protection bits with data in row latch
0x0C	Get Checksum	Flash	Any	Computes 4 byte checksum for given memory locations

Some commands are available only when the device is being controlled by an external system via the SWD interface and the test controller (see the Test Controller chapter on page 345).

Some commands require an array ID as a parameter. Array ID codes are shown in Table 43-3.

Table 43-3. Array ID Codes

Array ID Code	Memory Type
0x00 – 0x3E	Single Flash array
0x3F	All Flash arrays (used by the Erase All command)
0x40	Single EEPROM array
0xF8	Write Once NVL array

A Flash array has, at most, 64 KB plus 8 KB configuration bytes. PSoC 5 architecture has one or more arrays, where each array is 64K plus 8 KB configuration bytes. For example, if a PSoC 5 device has 256 KB Flash, there are four arrays, and the only valid array IDs are 0x00 – 0x03.

An EEPROM array has, at most, 2 KB. PSoC 5 devices have one EEPROM array, the size of which is 512 bytes, 1 KB, or 2 KB.

PSoC 5 devices have one write once NVL array.

For commands operating on Flash or EEPROM, all array IDs within the number of Flash and EEPROM arrays are valid. If a non-existent array is selected, the array ID wraps. For example, if a device has two Flash arrays (IDs = 0 and

1) and a command is sent with array ID = 3 then the upper bits of the ID are truncated and so array ID 1 is selected.

Some commands require an address as a parameter. As with array IDs, any address is valid for a Flash or EEPROM array. Upper address bits are truncated to allow only addressing of valid locations. For example, if a device has 512 bytes EEPROM and address 0x202 (514) is passed as a parameter, the operation takes place on address 0x002.

Array IDs and addresses do not wrap for NVL accesses.

Some commands use the row latch size for Flash and EEPROM. Row latch sizes are shown in the following table.

Table 43-4. Row Latch Sizes

Array Type	Size (Bytes)
Flash	288 (256 data bytes plus 32 configuration bytes)
EEPROM	16

43.3.1.1 Command Code Descriptions

The following are descriptions of the command codes listed in Table 43-2 on page 358.

- Command 0x00 – Load Byte

  Command Parameter Bytes – Array ID, Address, Data

  This command loads the given data byte into the volatile latch for the selected NVL array (in accordance with the array ID) at the given address. Only addresses within the selected NVL array are valid.

- Command 0x01 – Load Multiple Bytes

  Command Parameter Bytes – Array ID, Start address high, Start address low, Number of bytes (N), Data0, …, DataN

  This command loads N + 1 given data bytes into a row latch for Flash or EEPROM. N may range from 0 to 31 for Flash or 0 to 15 for EEPROM. The given start address + N must be less than the array row latch size. See Table 43-4.

- Command 0x02 – Load Row

  Command Parameter Bytes – Array ID, Data0, …, Data(row latch size -1)

  This command loads the given data bytes into a row latch for Flash or EEPROM. The number of data bytes expected equals the row latch size. See Table 43-4.

- Command 0x03 – Read Byte

  Command Parameter Bytes – Array ID, Address

  This command returns a data byte from the selected NVL array (per the array ID), at the given address. Only addresses within the selected NVL array are valid. Note that when this command is executed all of the data bytes are transferred from the nonvolatile cells to the volatile latch portion of the NVL.

- Command 0x04 – Read Multiple Bytes

  Command Parameter Bytes – Array ID, Start address high, Start address mid, Start address low, Number of bytes (N)

  This command returns N + 1 data bytes from Flash or EEPROM, starting at the given address.

  In Flash arrays, two address spaces exist – data and configuration. Bit 7 of the Address high parameter selects which of the two address spaces is addressed. If the bit is 0 then the data space is selected, otherwise the configuration space is selected. For example, if the address is 0x80000B and N is 0x08, the command reads 9 configuration bytes starting at address 0x00000B.

  The address plus N must not cross a row boundary – 256 for the Flash data space, 32 for the Flash configuration space, and 16 for EEPROM.

  Address wrapping applies; if the address is greater than the Flash or EEPROM size, the upper bits are then ignored. For example, 16 bits of address are needed to access the data space in a 64 KB Flash array, so the seven LS bits of the Address high parameter are ignored. Address 0x045A8B actually addresses 0x005A8B.

  Similarly, 13 address bits are needed to access the 8 KB configuration space associated with a 64 KB Flash array, and 11 address bits are needed to access a 2 KB EEPROM. For example, for a 64 KB Flash array (which also has 8 KB configuration bytes), valid address ranges are:

  - Data space – 0x000000 – 0x00FFFF (64 KB)
  - Configuration space – 0x800000 – 0x801FFF (8 KB)

- Command 0x05 – Write Row

  Command Parameter Bytes – Array ID, Row ID high, Row ID low, Temperature sign, Temperature magnitude

  This command erases the addressed Flash/EEPROM row and then programs it with the data in the row latch. If the row ID is greater than the array size (in rows), then the row ID wraps (the upper bits are ignored). The internal die temperature parameters (sign, magnitude) need to be passed while performing a write row operation. When this command is used for doing a complete device programming including the NVL, a fixed value of +25 °C is passed for this command. This is explained in the PSoC 5 Device Programming Specification document at http://www.cypress.com/?rID=46790. When using this command to wirte to flash/EEPROM rows as part of the application code, PSoC Creator APIs should be used for measuring the temperature. The measured temperature (sign, magnitude) should be passed as command parameters.

  For Flash, data bytes and configuration bytes are both programmed.

- Command 0x06 – Write User NVL

Command Parameter Bytes – Array ID

This command writes all of the bytes in the volatile latch for the selected NVL array (per the array ID) to that NVL array. All Flash protection bits must be cleared (no Flash protection) or the command fails. The only user NVL in PSoC 5 is WO NVL. There are conditions on the device operating voltage and temperature for programming WO NVL. Refer to the device data sheet for more details.

- Command 0x08 – Erase Sector

  Command Parameter Bytes – Array ID, Sector ID

  This command erases a sector of Flash/EEPROM. A sector is a block of 64 contiguous rows that starts at a 64-row boundary. For Flash arrays, all configuration bytes are also erased. The sector ID wraps if it exceeds the number of sectors.

- Command 0x09 – Erase All

  Command Parameter Bytes – None

  This command erases all Flash data and configuration bytes, all Flash protection rows, and all row latches in all Flash arrays on the device.

- Command 0x0B – Protect

  Command Parameter Bytes – Array ID, Row Select

  This command programs a Flash protection row with data in the Flash row latch (see 43.3.3 Flash Protection Settings on page 361). This command can be executed only if none of the protection bits are currently set – no Flash protection. Any bytes of the protection row that are marked as unused space are programmed with 0x00. This occurs regardless of what values are loaded into the row latches prior to sending this command.

  The Row Select parameter is used for Flash arrays that have a row size less than 256 bytes. Because all Flash arrays have 256-byte rows, this parameter should always be 0x00.

  When the Flash protection data is programmed, this command cannot be sent again until an Erase All command is sent first.

  For devices with multiple Flash arrays, the All Flash array ID (0x3F) may be used with this command. This causes each Flash array to have its protection row programmed with its row latch contents, simultaneously with the other arrays, reducing the overall Flash programming time.

- Command 0x0C – Get Checksum

  Command Parameter Bytes – Array ID, Start row high, Start row low, Number of rows high, Number of rows low

  This command computes a 4-byte checksum for the given number of Flash rows + 1, starting at the given row. The checksum is computed by a running simple addition of all values in the rows. The computation includes all data from the user space and the configuration space.

The Array ID parameter in this command can only take the value of the Single Flash array, as listed in Table 43-3. This parameter cannot take the value of “All Flash arrays” (0x3F). Passing the “All Flash arrays” as parameter value for ArrayID will result in command failure. To calculate the checksum value of the entire PSoC 5 flash memory, the “Get Checksum” command needs to be called separately for each flash array. The running sum of the checksum values returned for each flash array gives the checksum for the entire flash memory of PSoC 5.

The checksum value is returned MS byte first.

43.3.1.2 Command Failure Codes

In response to commands, a success/failure code is returned in the SPC_SR register: These codes are described in Table 43-5.

Table 43-5. Command Failure Codes

Success/Failure Code (Bits[7:2] in SPC_SR register)	Meaning
0x00	Command successfully executed
0x01	Invalid array ID
0x02	Invalid key
0x03	Array is asleep
0x04	External access failure: command must be sent via test controller
0x05	Invalid ‘N’ value
0x07, 0x08	Program/Erase failure
0x09	Protection check failure: protection settings are in a state that prevents the command from executing
0x0A	Invalid address
0x0B	Invalid command code
0x0C	Invalid row ID

43.3.2 Register Summary

All programming operations are done through a simple command/status register interface, shown in Table 43-1 on page 358.

43.3.3 Flash Protection Settings

Each row of Flash has its own protection settings. For each Flash array, Flash protection bits are stored in a “hidden” row in that array. A hidden row is one that is not readable by the CPU, and contains no CPU program or data bytes. In the hidden row, two bits per Flash row are packed into a byte; therefore, each byte in the hidden row has protection settings for four Flash rows. As shown in Figure 43-2, the Flash rows are ordered so that the first two bits in the hidden row correspond to the protection settings of Flash row 0.

Protection is cumulative in that modes have successively higher protection levels and include the lower protection modes. The following table shows the protection modes.

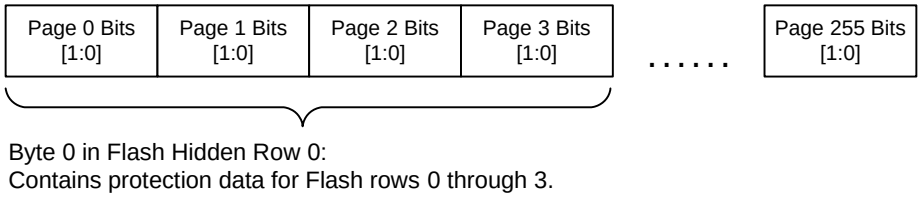
Table 43-6. Protection Modes

Mode	Description	Read[a]	External Write[b]	Internal Write[c]
00	Unprotected	Yes	Yes	Yes
01	Read Protect	No	Yes	Yes
10	Disable External Write	No	No	Yes
11	Disable Internal Write	No	No	No

a. Read – Applies to Test Controller and Read Commands.
b. External Write – Test Controller/third-party programmers.
c. Internal Write – Boot loading or writes due to firmware execution.

When a read/write/erase operation is to be done for a row, the corresponding protection bits are checked. The command is executed only if allowed under the current protection mode. If the command is not allowed, the command then fails.

Figure 43-2. Flash Protection Bits



Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it would be necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PSoC device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.

AND See *Boolean Algebra*.

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

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

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

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

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

B

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

bandwidth

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

bias

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

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

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

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

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

block

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

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

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

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

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

buffer

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

bus

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

byte A digital storage unit consisting of 8 bits.

C

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

D

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

E

emulator Duplicates (provides an emulation of) the functions of one system with a different system, so that the second system appears to behave similar to the first system.

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

F

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

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

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

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

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

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

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

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

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

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

G

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

gate

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

ground

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

H

hardware

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

hardware reset

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

hexadecimal

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

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

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

high time

The amount of time the signal has a value of ‘1’ in one period, for a periodic digital signal.

I

I²C A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). I^2C is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. I^2C uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.

ICE The in-circuit emulator that allows users to test the project in a hardware environment, while viewing the debugging device activity in a software environment (PSoC Designer™).

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

impedance
1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.
2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.

input A point that accepts data, in a device, process, or channel.

input/output (I/O) A device that introduces data into or extracts data from a system.

instruction An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.

instruction mnemonics A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.

integrated circuit (IC) A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.

interface The means by which two systems or devices are connected and interact with each other.

interrupt A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

interrupt service routine (ISR) A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter

1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams.
2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

K

keeper A circuit that holds a signal to the last driven value, even when the signal becomes un-driven.

L

latency The time or delay that it takes for a signal to pass through a given circuit or network.

least significant bit (LSb) The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.

least significant byte (LSB) The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

Linear Feedback Shift Register (LFSR) A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.

load The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).

logic function A mathematical function that performs a digital operation on digital data and returns a digital value.

lookup table (LUT) A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.

low time The amount of time the signal has a value of ‘0’ in one period, for a periodic digital signal.

low voltage detect (LVD) A circuit that senses Vddd and provides an interrupt to the system when Vddd falls below a selected threshold.

M

M8CP An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a PSoC device by interfacing to the Flash, SRAM, and register space.

macro A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.

mask

1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference.
2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.

master device A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.

microcontroller An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.

mnemonic A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode A distinct method of operation for software or hardware. For example, the Digital PSoC block may be in either counter mode or timer mode.

modulation A range of techniques for encoding information about a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux) 1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

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

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise 1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR See *Boolean Algebra*.

NOT See *Boolean Algebra*.

O

OR See *Boolean Algebra*.

oscillator A circuit that may be crystal controlled and is used to generate a clock frequency.

output The electrical signal or signals which are produced by an analog or digital block.

P

parallel The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.

parameter Characteristics for a given block that have either been characterized or may be defined by the designer.

parameter block A location in memory where parameters for the SSC instruction are placed prior to execution.

parity A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).

path 1. The logical sequence of instructions executed by a computer.
2. The flow of an electrical signal through a circuit.

pending interrupts	An interrupt that has been triggered but has not been serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.
phase	The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).
Phase-Locked Loop (PLL)	An electronic circuit that controls an *oscillator* so that it maintains a constant phase angle relative to a reference signal.
pin	A terminal on a hardware component. Also called lead.
pinouts	The pin number assignment: the relation between the logical inputs and outputs of the PSoC device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names.
port	A group of pins, usually eight.
positive edge	A transition from a logic 0 to a logic 1. Also known as a rising edge.
posted interrupts	An interrupt that has been detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.
Power On Reset (POR)	A circuit that forces the PSoC device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*.
program counter	The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.
protocol	A set of rules. Particularly the rules that govern networked communications.
PSoC®	Cypress's Programmable System-on-Chip (PSoC®) devices.
PSoC blocks	*See analog blocks* and *digital blocks*.
PSoC Creator™	The software for Cypress's next generation Programmable System-on-Chip technology.
PSoC Designer™	The software for Cypress's Programmable System-on-Chip technology.
pulse	A rapid change in some characteristic of a signal (for example, phase or frequency), from a baseline value to a higher or lower value, followed by a rapid return to the baseline value.
pulse-width modulator (PWM)	An output in the form of duty cycle which varies as a function of the applied measurand.

R

RAM	An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.
register	A storage device with a specific capacity, such as a bit or byte.
reset	A means of bringing a system back to a know state. See *hardware reset* and *software reset*.
resistance	The resistance to the flow of electric current measured in ohms for a conductor.
revision ID	A unique identifier of the PSoC device.
ripple divider	An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to 2^n - 1.
rising edge	*See positive edge*.
ROM	An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.
routine	A block of code, called by another block of code, that may have some general or frequent use.
routing	Physically connecting objects in a design according to design rules set in the reference library.
runt pulses	In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling	The process of converting an analog signal into a series of digital values or reversed.
schematic	A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.
seed value	An initial value loaded into a linear feedback shift register or random number generator.
serial	1. Pertaining to a process in which all events occur one after the other. 2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.
set	To force a bit/register to a value of logic 1.
settling time	The time it takes for an output signal or value to stabilize after the input has changed from one value to another.

shift	The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.
shift register	A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.
sign bit	The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.
signal	A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.
silicon ID	A unique identifier of the PSoC silicon.
skew	The difference in arrival time of bits transmitted at the same time, in parallel transmission.
slave device	A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.
software	A set of computer programs, procedures, and associated documentation concerned with the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed.
software reset	A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not PSoC blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at Flash address 0x0000.
SRAM	An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value has been loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.
SROM	An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform Flash operations. The functions of the SROM may be accessed in normal user code, operating from Flash.
stack	A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.
stack pointer	A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.
state machine	The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.
sticky	A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The PSoC SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous
1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the PSoC device and reading this manual.

user modules Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital PSoC Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

Vddd A name for a power net meaning "voltage drain." The most positive power supply signal. Usually 5 or 3.3 volts.

volatile Not guaranteed to stay the same value or level when not in scope.

Vss A name for a power net meaning "voltage source." The most negative power supply signal.

W

watchdog timer A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time.

waveform The representation of a signal as a plot of amplitude versus time.

X

XOR See *Boolean Algebra*.

Index

Numerics

32.768 kHz crystal oscillator 94
50% duty cycle mode 100
8051
  how it works 315

A

active interrupt
  PSoC 5 72
active mode
  entering 112
  exiting 112
  low power modes 112
  PSoC 112
alternative active mode 112
  entering 113
  exiting 113
  low power modes 112
analog I/O 134
analog subsystem
  PSoC 25
analog subsystem components
  PSoC 25
analog system 257
  architecture 257
asynchronous clocks 103

B

BASEPRI special register in PSoC 5
  PSoC 5
    BASEPRI special register 75
block diagram
  clock distribution system 97
  clock divider Implementation 100
  clock selection 213
  Cortex-M3 36
  CY8C55 family SRAM accesses 81
  DMA controller 51
  Flash programming system 83
  GPIO 126
  I/O drive mode 129
  internal low speed oscillator 93
  internal SRAM organization for the CY8C55 family 81
  interrupt controller 67
  master clock mux 98
  MHzECO 94
  phase shifter 102
  phase-locked loop 95
  PHUB 51
  port interrupt controller unit 139
  RESET module 120
  resync option 101
  SAR ADC 339
  SIO 127
  USB clock mux 99
  USBIO 128
  voltage monitoring 107
  watchdog timer circuit 117
bypassed clock source 101

C

capture mode
  timing diagram 214
capture signal 214
central processing unit
  cache controller 49
  PHUB and DMA 51
clock
  system 103
clock block
  components 92
clock distribution 97
clock distribution module 100
clock divider 103
  50% duty cycle mode 100
  single cycle pulse mode 100
clock doubler 93
clock frequency
  USB mode operations 99
clock generator
  introduction 91
clock selection
  block diagram 213
clock selection by timer block 212
clock signal naming 104
clock source
  selection by timer block 212
clock sources
  distribution 97
  DSI 95

internal 92
phase-locked loop 95
clock synchronization 101
clock tree
bypassed clock source output 101
phase delayed clk_sync output 101
power gating 103
resynchronized clock output 101
types 97
unsynchronized divided clock output 101
clocks
asynchronous 103
high precision 91
low power mode operation 103
cock dividers
main part of clock distribution module 100
compare types in pulse width mode 221
contention
CY8C38 family 82
CY8C55 family 82
counter mode
register configuration 217
counters
features 211
CPU system 33
architecture 33
crystal oscillator
low power operation 94
CY8C38 family
contention 82
CY8C55 family
contention 82
RAM implementation 82

D

dead band feature 223
debug and trace
introduction 351
delay chain 102
development kits 27
digital I/O
control by DSI 132
digital system 147
architecture 147
digital-to-analog converter 25
DMA controller
block diagram 51
document
glossary 363
overview 3
revision history 19
doubled clock 93
DSI control of digital I/O 132
DSI input 133
DSI output to I/O port 132

E

EEPROM
features 85
EEPROM memory
how it works 86
PSoC 85
enable signal 214
enabling and disabling timer block 213
enabling interrupts
PSoC 69
exceptions supported by PSoC 5 75
external crystal oscillators 94
external reset 119

F

fast start IMO (FIMO) 93
FAULTMASK special register in PSoC 5
PSoC 5
FAULTMASK special register 75
features
counters 211
EEPROM 85
Flash memory 83
I/O system 125
interrupt controller 67
low power modes 111
nonvolatile latch 79
PHUB 51
port interrupt controller unit 139
power supply and monitoring 105
pulse-width modulator 211
SRAM 81
timers 211
watchdog timer 117
Flash memory
features 83
how it works 84
PSoC 83
regions 84
free run mode
timing diagram 216
free run timer mode 215
frequency generation 91

G

gated timer in pulse width
timing diagram 218
general purpose input/output interface
PSoC 24
glossary 363
GPIO
block diagram 126
GPIO pins in creation of buttons and sliders 135

H

help, getting
development kits 27
hibernate mode 113
entering 114
exiting 114
low power modes 113
hibernate regulator 107
high impedance analog drive mode 130
high impedance digital drive mode 130
high voltage interrupt 107, 108
hot swap
SIO pin support 136
how it works
8051 315
EEPROM memory 86
Flash memory 84
I/O system 128
interrupt controller 68
PHUB 52
power regulators 107
PSoC RAM 82
watchdog timer 118

I

I/O drive mode 128
block diagram 129
high impedance analog 130
high impedance digital 130
open drain 130
resistive 130
resistive pull up and pull down 130
strong 130
I/O pins
low power mode behavior 139
overvoltage tolerance 138
I/O power supply 138
I/O system
analog I/O 134
CapSense 135
digital I/O controlled through DSI 132
DSI input 133
DSI output 132
features 125
how it works 128
I/O port reconfiguration 138
I/O power supply 138
introduction 125
LCD drive capabilities 134
low power mode effect on I/O pins 139
open drain modes 130
overvoltage tolerance 138
port interrupt controller unit 139
port interrupt controller unit pin configuration 140
port register of digital I/O 130
power up I/O configuration 138
register summary 141
resistive modes 130
resistive pull up and pull down mode 130
SIO functions 135
slew rate control 130
strong drive mode 130
usage modes and configuration 128
identifying reset sources 119
input signals of timer block 213
internal low speed oscillator 92, 93
internal main oscillator 92
internal regulators 107
interrupt controller
block diagram 67
features 67
how it works 68
interrupt execution 70
introduction 67
level interrupt 70
priority decoding unit 71
pulse interrupt 70
interrupt execution
interrupt controller 70
interrupt masking
PSoC 5 75
interrupt nesting
PSoC 5 72
interrupt service routine
PSoC 5 74
interrupt vector addresses
PSoC 5 74
introduction
clock generator 91
I/O system 125
programming, testing, debugging, tracing 351
reset 119
successive approximation register analog to digital convertor 339
timer 211

K

kill signal 215

L

late arrival interrupts
PSoC 5 74
LCD drive
I/O system capabilities 134
level interrupt
interrupt controller 70
logic diagram
RESET module 120
low power modes 111
active mode 112
alternative active mode 112

features 111
hibernate mode 113
power mode transitions diagram 111
sleep mode 113
timewheel 114
low voltage detect interrupt processing 108
low voltage interrupt 107, 108

M

main stack pointer 72
mapping diagram
digital system input to pad selection 132
master clock mux 98
memory 77
architecture 77
MHz crystal oscillator 94
modes
active mode in low power modes 112

N

nonvolatile latch
features 79
programming 80
sleep mode 80
write once 79

O

on the fly duty cycle update 223
one shot mode 224
oscillators
external crystal 94
internal PSoC 92
summary table 95
overview, document 3, 19
document construction 29
getting started 27
revision history 19
overvoltage tolerance in I/O pins 138

P

phase delayed clk_sync 101
phase select and control 102
phase shifter 102
phase-locked loop 95
PHUB
block diagram 51
features 51
how it works 52
port interrupt controller
features 139
port interrupt controller unit 139
block diagram 139
pin configuration 140
port register control of digital I/O 130
power gating
clock outputs 103
power mode transitions in low power modes 111
power regulators
how it works 107
power supply
PSoC 5 24
power supply and monitoring
features 105
PRIMASK special register in PSoC 5 75
priority decoding unit
interrupt controller 71
process stack pointer 72
product upgrades 27
program and debug 343
architecture 343
PSoC 25
PSoC
active mode 112
analog subsystem 25
analog subsystem components 25
DMA controller 23
EEPROM memory 85
enabling interrupts 69
Flash memory on-chip 83
general purpose input/output interface 24
program and debug 25
special input/output interface 24
SRAM 81
supply pins 105
timer blocks 211
USB input/output interface 24
PSoC 3
8051 features 72
processor 23
PSoC 5
active interrupt 72
exceptions supported 75
interrupt controller 72
interrupt masking 75
interrupt nesting 72
interrupt service routine 74
interrupt vector addresses 74
late arrival interrupts 74
major components 21
power supply 24
PRIMASK special register 75
tail chaining 74
PSoC Ram
how it works 82
PSoC5
interrupt controller 72
pulse interrupt
interrupt controller 70
pulse width mode
comparator mode 221

compare types 221
pulse width modulator
dead band feature 223
on the fly duty cycle update 223
one shot mode 224
register configuration 221
pulse-width modulator
features 211

R

RAM implementation
CY8C55 family 82
real time clock 95
register summary
I/O system 141
registers
counter mode 217
pulse width modulator 221
timer block 225
regulator
hibernate 107
internal 107
reset 119
external 119
identifying sources 119
introduction 119
software initiated 119
watchdog 119
RESET module
logic diagram 120
reset sources
description 119
resynchronized clock 101
revision history 19

S

SAR ADC
introduction 339
single cycle pulse mode 100
SIO
block diagram 127
SIO functions
adjustable input level 135
hot swap 136
regulated output level 135
sleep mode
entering 113
exiting 113
gating off of clock network outputs 103
low power mode 113
low power modes 113
nonvolatile latch behavior 80
timer block behavior 225
slew rate control in I/O system 130
software initiated reset 119
special input/output interface
PSoC 24
SRAM
features 81
PSoC 81
supply pins
PSoC 105
synchronization of clock 101
system clock operation 103
system integration
low power modes 111
reset 119
system wide resources 89
architecture 89

T

tail chaining
PSoC 5 74
timer
introduction 211
timer block
enabling and disabling 213
input signals 213
modes 212
registers 225
selection of clock source 212
sleep mode behavior 225
timer blocks
PSoC 211
timer mode
free run 215
period mode 219
stop on interrupt mode 220
timer reset signal 215
timing diagram 215
timers
features 211
timewheel in low power modes 114
timing diagram
capture mode 214
free run mode 216
gated timer in IRQ mode 220
gated timer in period mode 219
gated timer in pulse width mode 218
timer reset signal 215

U

unsynchronized divided clock 101
update 103
updating of clock divider 103
upgrades 27
usage modes and configuration of I/O system 128
USB clock 99
USB input/output interface
PSoC 24

USB mode operation
clock frequency 99
USBIO
block diagram 128

V

voltage monitoring 107

W

watchdog reset 119
watchdog timer
clearing 118
disabling 118
enabling 118
features 117
how it works 118
setting time period 118
write once latch 79