PSoC® 3 Architecture TRM

(Technical Reference Manual)

Document No. 001-50235 Rev. *I

October 1, 2013

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

PSoC and CapSense are registered trademarks of Cypress Semiconductor Corporation. PSoC Designer and PSoC Creator are trademarks 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 **21**

1. Introduction 23
2. Getting Started 29
3. Document Construction 31

Section B: CPU System **35**

4. 8051 Core 37
5. PSoC 3 Cache Controller 67
6. PHUB and DMAC 73
7. Interrupt Controller 89

Section C: Memory **97**

8. Nonvolatile Latch 99
9. SRAM 103
10. Flash Program Memory 107
11. EEPROM 109
12. EMIF 111
13. Memory Map 119

Section D: System Wide Resources **121**

14. Clocking System 123
15. Power Supply and Monitoring 137
16. Low-Power Modes 145
17. Watchdog Timer 149
18. Reset 151
19. I/O System 159
20. Flash, Configuration Protection 175

Section E: Digital System **181**

21. Universal Digital Blocks (UDBs) 183
22. UDB Array and Digital System Interconnect 225
23. Controller Area Network (CAN) 233
24. USB 249
25. Timer, Counter, and PWM 265
26. I^2C 281
27. Digital Filter Block (DFB) 295

Section F: Analog System **311**
28. Switched Capacitor/Continuous Time ... 313
29. Analog Routing ... 327
30. Comparators ... 343
31. Opamp ... 347
32. LCD Direct Drive ... 351
33. CapSense ... 365
34. Temperature Sensor ... 371
35. Digital-to-Analog Converter ... 375
36. Precision Reference ... 379
37. Delta Sigma Converter ... 383

Section G: Program and Debug **403**
38. Test Controller ... 405
39. 8051 Debug on-Chip ... 417
40. Nonvolatile Memory Programming ... 427

Glossary **433**

Index **449**

Contents

Section A: Overview 21

Document Revision History 21

1. Introduction 23

1.1 Top Level Architecture 23
1.2 Features 25
1.3 CPU System 25
1.3.1 Processor 25
1.3.2 Interrupt Controller 25
1.3.3 DMA Controller 25
1.4 Memory 25
1.5 System Wide Resources 26
1.5.1 I/O Interfaces 26
1.5.2 Internal Clock Generators 26
1.5.3 Power Supply 26
1.5.3.1 Boost Converter 26
1.5.3.2 Sleep Modes 26
1.6 Digital System 26
1.7 Analog System 27
1.7.1 Delta Sigma ADC 27
1.7.2 Digital Filter Block 27
1.7.3 Digital-to-Analog Converters 27
1.7.4 Additional Analog Subsystem Components 27
1.8 Program and Debug 27

2. Getting Started 29

2.1 Support 29
2.2 Product Upgrades 29
2.3 Development Kits 29

3. Document Construction 31

3.1 Major Sections 31
3.2 Documentation Conventions 31
3.2.1 Register Conventions 31
3.2.2 Numeric Naming 31
3.2.3 Units of Measure 32
3.2.4 Acronyms 32

Section B: CPU System 35

Top Level Architecture 35

4. 8051 Core 37

4.1 Features 37
4.2 Block Diagram 37

4.3 How It Works 38
4.3.1 Memory Spaces 38
4.3.2 Instruction Set 38
4.3.3 8051 Core Enhancements 38
4.3.4 Interrupt Controller Interface 38
4.4 CY 8051 Wrapper 39
4.4.1 SFR–I/O Interface 39
4.5 8051 Instructions 39
4.5.1 Internal Data Space Map 39
4.5.2 Addressing Modes 39
4.5.3 Arithmetic Logic Unit Functions 40
4.5.3.1 Arithmetic Instructions 40
4.5.3.2 Logical Instructions 41
4.5.3.3 Data Transfer Instructions 42
4.5.3.4 Boolean Instructions1 43
4.5.3.5 Program Branching Instructions 43
4.5.3.6 Instruction Set Details 44
4.6 8051 Special Function Registers (SFRs) 62
4.6.1 SFRs 62
4.6.2 Dual Data Pointer SFRs 63
4.6.3 24-Bit Data Pointer SFRs 64
4.6.4 I/O Port Access SFRs 65
4.6.5 Interrupt Enable (IE) 65
4.7 Program and External Data Spaces 66
4.7.1 Program Space 66
4.7.2 External Data Space 66
4.8 CPU Halt Mechanisms 66

5. PSoC 3 Cache Controller 67

5.1 Features 67
5.2 Block Diagram 67
5.3 Cache Memory Organization and Addressing 68
5.3.1 Cache Operation 69
5.3.2 Cache Line Locking: 69
5.3.3 Cache Line Loading in Firmware 69
5.3.4 Cache Line Replacement Policy 69
5.3.5 Background Fill (BFILL) 70
5.3.6 ECC (Error Correction Code) 70
5.3.6.1 Interrupts 70
5.3.7 Flash Low-Power Mode 71

6. PHUB and DMAC 73

6.1 PHUB 73
6.1.1 Features 73
6.1.2 Block Diagram 73
6.1.3 How It Works 74
6.1.4 Arbiter 75
6.2 DMA Controller 75
6.2.1 Local Memory 75
6.2.2 How the DMAC Works 75
6.2.2.1 Interspoke Transfers 76
6.2.2.2 Intraspoke Transfer 77
6.2.2.3 Handling Multiple DMA Channels 78

6.2.2.4 DMA Channel Priority 78
6.2.2.5 DMA Latency in case of Nonideal Conditions 80
6.2.2.6 Request per Burst Bit 85
6.2.2.7 Work Sep Bit 86
6.3 DMA Transaction Modes 86
6.3.1 Simple DMA 86
6.3.2 Auto Repeat DMA 86
6.3.3 Ping Pong DMA 86
6.3.4 Circular DMA 86
6.3.5 Indexed DMA 86
6.3.6 Scatter Gather DMA 87
6.3.7 Packet Queuing DMA 87
6.3.8 Nested DMA 87
6.4 Register List 88

7. Interrupt Controller 89

7.1 Features 89
7.2 Block Diagram 89
7.3 How It Works 90
7.3.1 Enabling Interrupts 90
7.3.2 Pending Interrupts 91
7.3.3 Interrupt Priority 91
7.3.4 Level versus Pulse Interrupt 92
7.3.5 Interrupt Execution 92
7.4 PSoC 3 Interrupt Controller Features 94
7.4.1 Active Interrupts 94
7.4.2 Interrupt Nesting 94
7.4.3 Interrupt Vector Addresses 95
7.4.4 Sleep Mode Behavior 95
7.5 Interrupt Controller and Power Modes 96

Section C: Memory 97

Top Level Architecture 97

8. Nonvolatile Latch 99

8.1 Features 99
8.2 Device Configuration NV Latch 99
8.2.1 PRTxRDM[1:0] 99
8.2.2 XRESMEN 100
8.2.3 DEBUG_EN 100
8.2.4 CFGSPEED 100
8.2.5 DPS[1:0] 100
8.2.6 ECCEN 100
8.2.7 DIG_PHS_DLY[3:0] 100
8.3 Write Once NV Latch 100
8.4 Programming NV Latch 102
8.5 Sleep Mode Behavior 102

9. SRAM 103

9.1 Features 103
9.2 Block Diagram 103
9.3 How It Works 105

10. Flash Program Memory 107
10.1 Features 107
10.2 Block Diagram 107
10.3 How It Works 108
10.4 Flash Memory Access Arbitration 108
10.5 ECC Error Detection and Interrupts 108

11. EEPROM 109
11.1 Features 109
11.2 Block Diagram 109
11.3 How It Works 110

12. EMIF 111
12.1 Features 111
12.2 Block Diagram 111
12.3 How It Works 112
12.3.1 List of EMIF Registers 112
12.3.2 External Memory Support 112
12.3.3 Sleep Mode Behavior 115
12.4 EMIF Timing 116
12.5 Using EMIF with Memory-Mapped Peripherals 118
12.6 Additional Configuration Guidelines 118
12.6.1 Address Bus Configuration 118
12.6.2 Data Bus Configuration 118
12.6.3 16-bit Memory Transfers 118
12.6.4 8-bit Memory Transfers 118

13. Memory Map 119
13.1 Features 119
13.2 Block Diagram 119
13.3 How It Works 119
13.3.1 PSoC 3 Memory Map 120

Section D: System Wide Resources 121
Top Level Architecture 121

14. Clocking System 123
14.1 Features 123
14.2 Block Diagram 124
14.3 Clock Sources 124
14.3.1 Internal Oscillators 124
14.3.1.1 Internal Main Oscillator 124
14.3.1.2 Internal Low-Speed Oscillator 125
14.3.2 External Oscillators 126
14.3.2.1 MHz Crystal Oscillator 126
14.3.2.2 32.768 kHz Crystal Oscillator 127
14.3.3 Oscillator Summary 128
14.3.4 DSI Clocks 128
14.3.5 Phase-Locked Loop 128
14.4 Clock Distribution 129
14.4.1 Master Clock Mux 130
14.4.2 USB Clock 130
14.4.3 Clock Dividers 131

14.4.3.1 Single Cycle Pulse Mode 132
14.4.3.2 50% Duty Cycle Mode 132
14.4.3.3 Early Phase Option 132
14.4.4 Clock Synchronization 132
14.4.5 Phase Selection and Control 133
14.4.6 Divider Update 134
14.4.7 Power Gating of Clock Outputs 134
14.4.8 System Clock 134
14.4.9 Asynchronous Clocks 134
14.5 Low-Power Mode Operation 135
14.6 Clock Naming Summary 135

15. Power Supply and Monitoring 137

15.1 Features 137
15.2 Block Diagram 138
15.3 How It Works 139
15.3.1 Regulator Summary 139
15.3.1.1 Internal Regulators 139
15.3.1.2 Sleep Regulator 139
15.3.1.3 Hibernate Regulator 139
15.3.2 Boost Converter 139
15.3.2.1 Operating Modes 140
15.3.2.2 Status Monitoring 140
15.3.3 Voltage Monitoring 141
15.3.3.1 Low-Voltage Interrupt 141
15.3.3.2 High Voltage Interrupt 142
15.3.3.3 Processing a Low/High Voltage Detect Interrupt 142
15.3.3.4 Reset on a Voltage Monitoring Interrupt 142
15.4 Register Summary 143

16. Low-Power Modes 145

16.1 Features 145
16.2 Active Mode 146
16.2.1 Entering Active Mode 146
16.2.2 Exiting Active Mode 146
16.3 Alternative Active Mode 147
16.3.1 Entering Alternative Active Mode 147
16.3.2 Exiting Alternative Active Mode 147
16.4 Sleep Mode 147
16.4.1 Entering Sleep Mode 147
16.4.2 Exiting Sleep Mode 147
16.5 Hibernate Mode 147
16.5.1 Entering Hibernate Mode 147
16.5.2 Exiting Hibernate Mode 148
16.6 Timewheel 148
16.6.1 Central Timewheel (CTW) 148
16.6.2 Fast Timewheel (FTW) 148
16.7 Register List 148

17. Watchdog Timer 149

17.1 Features 149
17.2 Block Diagram 149
17.3 How It Works 150

17.3.1 Enabling and Disabling the WDT 150
17.3.2 Setting the WDT Time Period and Clearing the WDT 150
17.3.3 Operation in Low-Power Modes 150
17.3.4 Watchdog Protection Settings 150
17.4 Register List 150

18. Reset 151

18.1 Reset Sources 151
18.1.1 Power-on-Reset 151
18.1.2 Low-Voltage Reset and High-Voltage Reset 151
18.1.3 Watchdog Reset 151
18.1.4 Software Initiated Reset 152
18.1.5 External Reset 152
18.1.6 Identifying Reset Sources 152
18.1.6.1 Preservation of Reset Status 152
18.2 Reset Diagram 152
18.3 Reset Summary 154
18.4 Boot Process and Timing 155
18.4.1 Manufacturing Configuration NV Latch 156
18.4.1.1 Device Configuration NV Latch 156
18.4.2 Boot Phase 156
18.4.3 User Mode 157
18.5 Register List 157

19. I/O System 159

19.1 Features 159
19.2 Block Diagrams 160
19.3 How It Works 162
19.3.1 Usage Modes and Configuration 162
19.3.2 I/O Drive Modes 162
19.3.2.1 Drive Mode on Reset 163
19.3.2.2 High Impedance Analog 163
19.3.2.3 High Impedance Digital 163
19.3.2.4 Resistive Pull Up or Resistive Pull Down 163
19.3.2.5 Open Drain, Drives High and Drives Low 164
19.3.2.6 Strong Drive 164
19.3.2.7 Resistive Pull Up and Pull Down 164
19.3.3 Slew Rate Control 164
19.3.4 Digital I/O Controlled by Port Register 164
19.3.4.1 Port Configuration Registers 164
19.3.4.2 Pin Wise Configuration Register Alias 164
19.3.4.3 Port Wide Configuration Register Alias 165
19.3.5 SFR to GPIO 166
19.3.6 Digital I/O Controlled Through DSI 166
19.3.6.1 DSI Output 166
19.3.6.2 DSI Input 167
19.3.6.3 DSI for Output Enable Control 167
19.3.7 Analog I/O 168
19.3.8 LCD Drive 168
19.3.9 CapSense 168
19.3.10 External Memory Interface (EMIF) 169
19.3.11 SIO Functions and Features 169
19.3.11.1 Regulated Output Level 169

19.3.11.2 Adjustable Input Level 169
19.3.11.3 Hot Swap 170
19.3.12 Special Functionality 170
19.3.13 I/O Port Reconfiguration 171
19.3.14 Power Up I/O Configuration 171
19.3.15 Overvoltage Tolerance 171
19.3.16 I/O Power Supply 172
19.3.17 Sleep Mode Behavior 172
19.3.18 Low-power Behavior 172
19.4 Port Interrupt Controller Unit 172
19.4.1 Features 172
19.4.2 Interrupt Controller Block Diagram 172
19.4.3 Function and Configuration 173
19.5 Register Summary 174

20. Flash, Configuration Protection 175

20.1 Flash Protection 175
20.2 Device Security 176
20.3 Configuration Segment Protection 176
20.3.1 Locking/Unlocking Segment Configuration Register 177
20.3.2 Locking and Protecting Segments 177
20.3.3 Example 179
20.4 Frequently Asked Questions About Flash Protection and Device Security 179

Section E: Digital System 181

Top Level Architecture 182

21. Universal Digital Blocks (UDBs) 183

21.1 Features 183
21.2 Block Diagram 183
21.3 How It Works 184
21.3.1 PLDs 184
21.3.1.1 PLD Macrocells 185
21.3.1.2 PLD Carry Chain 187
21.3.1.3 PLD Configuration 187
21.3.2 Datapath 187
21.3.2.1 Overview 189
21.3.2.2 Datapath FIFOs 190
21.3.2.3 FIFO Status 197
21.3.2.4 Datapath ALU 197
21.3.2.5 Datapath Inputs and Multiplexing 200
21.3.2.6 CRC/PRS Support 200
21.3.2.7 Datapath Outputs and Multiplexing 203
21.3.2.8 Datapath Parallel Inputs and Outputs 205
21.3.2.9 Datapath Chaining 205
21.3.2.10 Dynamic Configuration RAM 206
21.3.3 Status and Control Module 207
21.3.3.1 Status and Control Mode 208
21.3.3.2 Control Register Operation 210
21.3.3.3 Parallel Input/Output Mode 211
21.3.3.4 Counter Mode 212
21.3.3.5 Sync Mode 213
21.3.3.6 Status and Control Clocking 213

21.3.3.7 Auxiliary Control Register ... 213
21.3.3.8 Status and Control Register Summary ... 214
21.3.4 Reset and Clock Control Module ... 214
21.3.4.1 Clock Control ... 215
21.3.4.2 Reset Control ... 217
21.3.4.3 UDB POR Initialization ... 219
21.3.5 UDB Addressing ... 220
21.3.5.1 Working Register Address Space ... 220
21.3.5.2 Configuration Register Address Space ... 222
21.3.5.3 UDB Configuration Address Space ... 222
21.3.5.4 Routing Configuration Address Space ... 222
21.3.6 System Bus Access Coherency ... 223
21.3.6.1 Simultaneous System Bus Access ... 223
21.3.6.2 Coherent Accumulator Access (Atomic Reads and Writes) ... 223
21.4 UDB Working Register Reference ... 224

22. UDB Array and Digital System Interconnect 225

22.1 Features ... 225
22.2 Block Diagram ... 225
22.3 How It Works ... 226
22.4 UDB Array System Interface ... 228
22.4.1 UDB Array POR Initialization ... 228
22.4.2 UDB POR Configuration Sequence ... 229
22.4.2.1 Quadrant Route Disable ... 230
22.4.3 UDB Sleep and Power Control ... 230
22.4.4 UDB Register References and Address Mapping ... 230

23. Controller Area Network (CAN) 233

23.1 Features ... 233
23.2 Block Diagram ... 234
23.3 CAN Message Frames ... 234
23.3.1 Data Frames ... 234
23.3.1.1 Standard Data Frame ... 234
23.3.1.2 Extended Data Frame ... 235
23.3.2 Remote Frame ... 236
23.3.3 Error Frame ... 236
23.3.4 Overload Frame ... 236
23.4 Transmitting Messages in CAN ... 236
23.4.1 Message Arbitration ... 237
23.4.2 Message Transmit Process ... 237
23.4.3 Message Abort ... 238
23.4.4 Transmitting Extended Data Frames ... 238
23.5 Receiving Messages in CAN ... 238
23.5.1 Message Receive Process ... 239
23.5.2 Acceptance Filter ... 239
23.5.2.1 Example ... 239
23.5.3 DeviceNet Filtering ... 241
23.5.4 Filtering of Extended Data Frames ... 241
23.5.5 Receiver Message Buffer Linking ... 242
23.6 Remote Frames ... 242
23.6.1 Transmitting a Remote Frame by the Requesting Node ... 243
23.6.2 Receiving a Remote Frame ... 243
23.6.3 RTR Auto Reply ... 243

23.6.4 Remote Frames in Extended Format 243
23.7 Bit Time Configuration 243
23.7.1 Allowable Bit Rates and System Clock (CLK_BUS) 243
23.7.2 Setting Bit Rate TSEG1 and TSEG2 244
23.7.2.1 Example 245
23.8 Error Handling and Interrupts in CAN 245
23.8.1 Types of Errors 245
23.8.1.1 BIT Error 245
23.8.1.2 FORM Error 245
23.8.1.3 ACKNOWLEDGE Error 245
23.8.1.4 CRC Error 245
23.8.1.5 STUFF Error 246
23.8.2 Error States in CAN 246
23.8.3 Interrupt Sources in CAN 246
23.9 Operating Modes in CAN 247
23.9.1 Listen Only Mode 247
23.9.2 Run/Stop Mode 247

24. USB 249

24.1 Features 249
24.2 Block Diagram 250
24.2.1 Serial Interface Engine (SIE) 251
24.2.2 Arbiter 252
24.2.2.1 SIE Interface Module 252
24.2.2.2 CPU Interface Block 252
24.2.2.3 Memory Interface 252
24.2.2.4 DMA Interface 252
24.2.2.5 Arbiter Logic 252
24.2.2.6 Synchronization Block 253
24.3 How it Works 253
24.3.1 Operating Frequency 253
24.3.2 Operating Voltage 253
24.3.3 Transceiver 253
24.3.4 Endpoints 254
24.3.5 Transfer Types 254
24.3.6 Interrupts 254
24.4 Logical Transfer Modes 255
24.4.1 Store and Forward Mode 256
24.4.1.1 No DMA Access 256
24.4.1.2 Manual DMA Access 257
24.4.2 Cut Through Mode 259
24.4.3 Control Endpoint Logical Transfer 261
24.5 PS/2 and CMOS I/O Modes 262
24.6 Register List 263

25. Timer, Counter, and PWM 265

25.1 Features 265
25.2 Block Diagram 265
25.3 How It Works 266
25.3.1 Clock Selection 266
25.3.2 Enabling and Disabling Block 267
25.3.3 Input Signal Characteristics 267
25.3.3.1 Enable Signal 268

25.3.3.2 Capture Signal 268
25.3.3.3 Timer Reset Signal 270
25.3.3.4 Kill Signal 270
25.3.4 Operating Modes 271
25.3.4.1 Timer Mode – Free Run Mode 271
25.3.4.2 Gated Timer Mode 272
25.3.4.3 Pulse-width Modulator Mode 276
25.3.4.4 One Shot Mode 279
25.3.5 Interrupt Enabling 279
25.3.6 Sleep Mode Behavior 280
25.4 Register Listing 280

26. I²C 281

26.1 Features 281
26.2 Block Diagram 281
26.3 Background Information 283
26.3.1 I²C Bus Description 283
26.3.2 Typical I²C Data Transfer 283
26.4 How It Works 283
26.4.1 Bus Stalling (Clock Stretching) 284
26.4.2 System Management Bus 284
26.4.3 Pin Connections 284
26.4.4 I2C Interrupts 284
26.4.5 Control by Registers 284
26.4.6 Operating the I²C Interface 284
26.4.6.1 Slave Mode 286
26.4.6.2 Master Mode 287
26.4.6.3 Multi-Master Mode 288
26.5 Hardware Address Compare 288
26.6 Wake from Sleep 288
26.7 Slave Mode Transfer Examples 289
26.7.1 Slave Receive 289
26.7.2 Slave Transmit 290
26.8 Master Mode Transfer Examples 291
26.8.1 Single Master Receive 291
26.8.2 Single Master Transmit 292
26.9 Multi-Master Mode Transfer Examples 293
26.9.1 Multi-Master, Slave Not Enabled 293
26.9.2 Multi-Master, Slave Enabled 294

27. Digital Filter Block (DFB) 295

27.1 Features 295
27.2 Block Diagram 295
27.3 How It Works 296
27.3.1 Controller 296
27.3.1.1 FSM RAM 297
27.3.1.2 Program Counter 298
27.3.1.3 Control Store 298
27.3.1.4 Next State Decoder 298
27.3.2 Datapath 299
27.3.2.1 MAC 300
27.3.2.2 ALU 300
27.3.2.3 Shifter and Rounder 300

27.3.3 Address Calculation Unit....301
27.3.4 Bus Interface and Register Descriptions....301
27.3.4.1 Streaming Mode....301
27.3.4.2 Block Transfer Modes....302
27.3.4.3 Result Handling....303
27.3.4.4 Data Alignment....305
27.3.4.5 DMA and Semaphores....305
27.3.4.6 DSI Routed Inputs and Outputs....305
27.4 DFB Instruction Set....306
27.5 Usage Model....309

Section F: Analog System 311

Top Level Architecture....311

28. Switched Capacitor/Continuous Time 313

28.1 Features....313
28.2 Block Diagram....313
28.3 How it Works....315
28.3.1 Operational Mode of Block is Set....315
28.4 Naked Opamp....315
28.4.1 Bandwidth/Stability Control....315
28.4.1.1 BIAS_CONTROL....315
28.4.1.2 SC_COMP[1:0]....316
28.4.1.3 SC_REDC[1:0]....316
28.5 Continuous Time Unity Gain Buffer....317
28.6 Continuous Time Programmable Gain Amplifier....317
28.7 Continuous Time Transimpedance Amplifier....318
28.8 Continuous Time Mixer....320
28.9 Sampled Mixer....321
28.10 Delta Sigma Modulator....323
28.10.1 First-Order Modulator, Incremental Mode....324
28.11 Track and Hold Amplifier....325

29. Analog Routing 327

29.1 Features....327
29.2 Block Diagram....327
29.3 How it Works....330
29.3.1 Analog Globals (AGs)....330
29.3.2 Analog Mux Bus (AMUXBUS)....330
29.3.3 Liquid Crystal Display Bias Bus (LCDBUS)....330
29.3.4 Analog Local Bus (abus)....332
29.3.5 Switches and Multiplexers....332
29.3.5.1 Control of Analog Switches....332
29.4 Analog Resource Blocks – Routing and Interface....334
29.4.1 Digital-to-Analog Converter (DAC)....335
29.4.2 Comparator....336
29.4.3 Delta Sigma Modulator (DSM)....337
29.4.4 Switched Capacitor....338
29.4.5 Opamp....339
29.4.6 Low-Pass Filter (LPF)....339
29.5 Low-Power Analog Routing Considerations....340
29.5.1 Mitigating Analog Routes with Degraded Low-power Signal Integrity....340
29.6 Analog Routing Register Summary....341

30. Comparators 343
30.1 Features 343
30.2 Block Diagram 343
30.3 How it Works 344
30.3.1 Input Configuration 344
30.3.2 Power Configuration 344
30.3.3 Output Configuration 344
30.3.4 Hysteresis 345
30.3.5 Wake Up from Sleep 345
30.3.6 Comparator Clock 345
30.3.7 Offset Trim 345
30.3.8 Register Summary 346

31. Opamp 347
31.1 Features 347
31.2 Block Diagram 348
31.3 How it Works 348
31.3.1 Input and Output Configuration 348
31.3.2 Power Configuration 348
31.3.3 Buffer Configuration 349
31.3.4 Register Summary 349

32. LCD Direct Drive 351
32.1 Features 351
32.2 LCD System Operational Modes 351
32.3 LCD Always Active 352
32.3.1 Functional Description 353
32.3.1.1 LCD DAC 353
32.3.1.2 LCD Driver Block 354
32.3.1.3 UDB 357
32.3.1.4 DMA 357
32.4 LCD Low-Power Mode 357
32.4.1 Functional Description 358
32.4.1.1 LCD Timer 358
32.4.1.2 UDB 358
32.4.1.3 DMA 359
32.4.1.4 LCD DAC and Driver: Low Power Feature 359
32.4.2 Timing Diagram for LCD Low-Power Mode 361
32.5 LCD Usage Models 363

33. CapSense 365
33.1 Features 365
33.2 Block Diagram 365
33.3 How It Works 366
33.3.1 Reference Driver 366
33.3.2 Low-pass Filter 366
33.3.3 Analog Mux Bus 366
33.3.4 GPIO Configuration for CapSense 366
33.3.5 Other Resources 367
33.4 CapSense Delta Sigma Algorithm 368

34. Temperature Sensor 371
34.1 Features 371

34.2 Block Diagram 371
34.3 How It Works 372
34.4 Command and Status Interface 372
34.4.1 Status Codes 372
34.4.2 Temperature Sensor Commands 372
34.4.2.1 Get Temperature 372
34.4.2.2 Setup Temperature Sensor 373
34.4.2.3 Disable Temperature Sensor 374

35. Digital-to-Analog Converter 375

35.1 Features 375
35.2 Block Diagram 375
35.3 How It Works 376
35.3.1 Current DAC 376
35.3.2 Voltage DAC 376
35.3.3 Output Routing Options 376
35.3.4 Making a Higher Resolution DAC 377
35.4 Register List 378

36. Precision Reference 379

36.1 Block Diagram 379
36.2 How It Works 379

37. Delta Sigma Converter 383

37.1 Features 383
37.2 Block Diagram 383
37.3 How It Works 384
37.3.1 Input Buffer 384
37.3.2 Delta Sigma Modulator 385
37.3.2.1 Clock Selection 386
37.3.2.2 Capacitance Configuration 386
37.3.2.3 Gain Configuration 387
37.3.2.4 Power Configuration 388
37.3.2.5 Other Configuration Options 392
37.3.2.6 Quantizer 392
37.3.2.7 Reference Options 392
37.3.2.8 Reference for DSM: Usage Guidelines 395
37.3.3 Analog Interface 396
37.3.3.1 Conversion of Thermometric Code to Two's Complement 397
37.3.3.2 Modulation Input 397
37.3.3.3 Clock Selection and Synchronization 397
37.3.4 Decimator 397
37.3.4.1 Shifters 397
37.3.4.2 CIC Filter 398
37.3.4.3 Post Processing Filter 398
37.3.5 ADC Conversion Time 399
37.3.6 Coherency Protection 399
37.3.6.1 Protecting Writes (Gain/Offset) with Coherency Checking 399
37.3.6.2 Protecting Reads (Output Sample) with Coherency Checking 400
37.3.7 Modes of Operation 400

Section G: Program and Debug 403

Top Level Architecture 403

38. Test Controller 405

38.1 Features 405
38.2 Block Diagram 405
38.3 Background Information 406
38.3.1 JTAG Interface 406
38.3.2 Serial Wire Debug Interface 409
38.4 How It Works 411
38.4.1 Clocking 411
38.4.2 PSoC 3 JTAG Instructions 411
38.4.3 DP/AP Access Register 412
38.4.4 JTAG/SWD Addresses (PSoC 3) 412
38.4.5 Debug Port and Access Port Registers (PSoC 3) 413
38.4.6 Register Access Examples 413
38.4.7 Boundary Scan Pin Order 414
38.5 Test Controller Interface Pins 414
38.6 Test Controller Acquisition 414
38.6.1 Changing Interface from SWD to JTAG 415
38.6.2 Changing Interface from JTAG to SWD 415
38.6.3 Boundary Scan 415
38.6.4 Functional Test 415
38.6.5 Programming Flash/EEPROM 415
38.6.6 Program Debug/Trace 415

39. 8051 Debug on-Chip 417

39.1 Features 417
39.2 Block Diagram 418
39.3 How it Works 419
39.3.1 Enabling and Activating 419
39.3.2 Halting, Stepping 419
39.3.3 Accessing PSoC Memory And Registers 419
39.3.4 Breakpoints 420
39.3.4.1 Program Address Breakpoints 420
39.3.4.2 Memory Access Breakpoint 420
39.3.4.3 Watchdog Trigger Breakpoint 420
39.3.4.4 Breakpoint Chaining 420
39.3.5 CPU Reset 421
39.3.6 Tracing Program Execution 421
39.3.6.1 Reading Traces 422
39.3.6.2 Trace Time Stamp 422
39.3.7 DOC Registers 422
39.4 Serial Wire Viewer 423
39.4.1 SWV Protocols 423
39.4.1.1 Manchester Encoding 424
39.4.1.2 UART Encoding 424
39.4.2 SWV Registers 425

40. Nonvolatile Memory Programming 427

40.1 Features 427
40.2 Block Diagram 427
40.3 How It Works 428
40.3.1 Commands 428
40.3.1.1 Command Code Descriptions 429
40.3.1.2 Command Failure Codes 430

40.3.2 Register Summary 430
40.3.3 Flash Protection Settings 431

Glossary **433**

Index **449**

Section A: Overview

This document encompasses the PSoC® 3 family of devices. In conjunction with the device datasheet, it contains complete and detailed information about how to design with the IP blocks that construct a PSoC 3 device. This document describes the analog and digital architecture, and helps to better understand the features of the device.

This section consists of the following chapters:

- Introduction chapter on page 23
- Getting Started chapter on page 29
- Document Construction chapter on page 31

See the *PSoC® 3 Registers TRM (Technical Reference Manual)* for complete register sets.

Document Revision History

Table 1-1. PSoC® 3 Architecture Technical Reference Manual (TRM) Revision History

Revision	Issue Date	Origin of Change	Description of Change
**	12.15.2008	HMT	Preliminary release of the PSoC 3: CY8C38 Family Technical Reference Manual.
*A	02.12.2009	HMT	Release for ES10.
*B	06.22.2009	HMT	Initial silicon release.
*C	07.14.2009	HMT	Initial non NDA release.
*D	09.08.2009	DSG	Addressed many issues, changes throughout document.
*E	12.23.2010	DSG	Document rewrite to reflect product development.
*F	06.05.2012	SHEA	Extensive content updates to reflect product development
*G	09.26.2012	SHEA	Modified reference to TMR_CFG field in "Operating Modes" on page 271. Updated Figure 12-1. Updated details on boost interface timing in 15.3.2 Boost Converter. Updated information on VDDA being greater than or equal to VDDD/ VDDIO in 15.2 Block Diagram. Added information on software reset during LVI in 15.3.3.3 Processing a Low/High Voltage Detect Interrupt. Updated 15.3.2.1 Operating Modes
*H	06/18/2013	SHEA	Added 18.1.2 Low-Voltage Reset and High- Voltage Reset and 18.1.6.1 Preservation of Reset Status; updates to 15.3.3. Voltage Monitoring. Added information on effect of changing pin modes in section 19.3.2 I/O Modes. Updated section 25.3.4.1 (Period register setting to EN = 1). Added a note to sections 10.3 and 11.3. Updated Tables 20-3 to 20-6. Made a correction in Figure 14-1. Modified the Datapath Top Level Diagram Updated IMO frequency to 62 MHz. Added DEBUG_EN parameter. Updates to PHUB and DMAC chapters
*I	09/26/2013	SHEA	Updated Drive Modes diagram in the I/O System chapter on page 159. Corrected section 14.3.2.2 32.768 kHz Crystal Oscillator to mention the active mode operating current. Removed the comparator as a wakeup source from hibernate in section 16.5.2 Exiting Hibernate Mode. Corrected FTW register name in section 16.6.2 Fast Timewheel (FTW).

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 3 (CY8C38xxxx, CY8C36xxx, CY8C34xxx, CY8C32xxx) families are fully scalable 8-bit PSoC platform devices that have these characteristics:

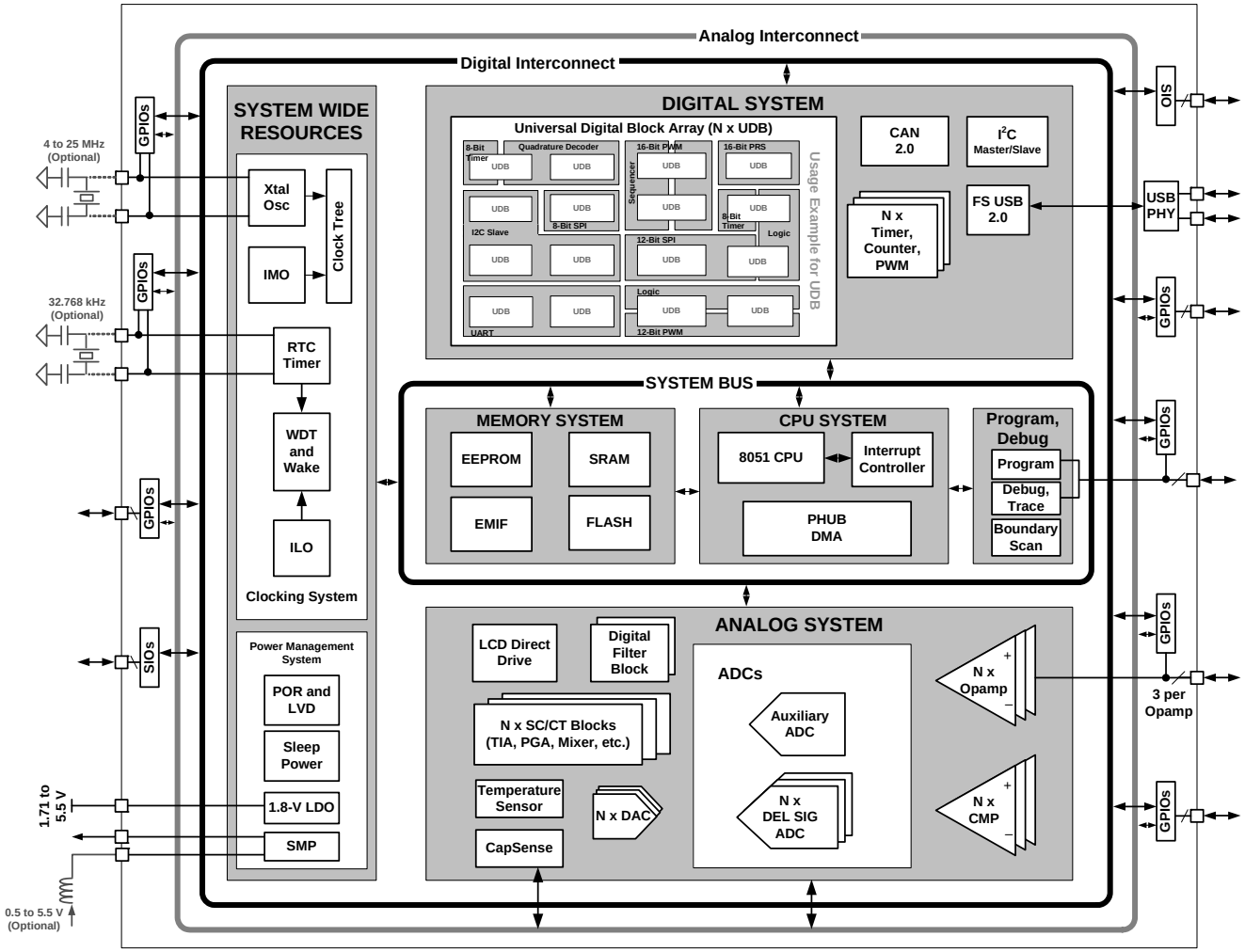
- High-performance, configurable digital system that supports a wide range of communication interfaces, such as USB, I2C, and CAN
- High-precision, high-performance analog system with up to 20-bit ADC, DACs, comparators, opamps, and programmable blocks to create PGAs, TIAs, mixers, and so on
- Easily configurable logic array
- Flexible routing to all pins
- High-performance, 8-bit single-cycle 8051 core
- PSoC Creator, an integrated development environment software

This document describes PSoC 3 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 3 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 24 shows the major components of the PSoC 3 architecture. The PSoC 3 device uses the 8051 core.

Figure 1-1. Top Level Architecture for PSoC 3 Devices



1.2 Features

PSoC 3 devices have these major components. See Figure 1-1 on page 24.

- 8051 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 CAN and 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, comparators, PGAs, and more
- Programming and debug system through JTAG, serial wire debug (SWD), and single wire viewer (SWV)

1.3 CPU System

1.3.1 Processor

PSoC 3 CPU subsystem is built around an 8-bit single cycle pipelined 8051 processor, running up to 67 MHz. The single cycle 8051 CPU runs ten times faster than a standard 8051 processor. The PSoC 3 instruction set is compatible with the original MCS-51 instruction set.

1.3.2 Interrupt Controller

The CPU subsystem includes a programmable Nested Vectored Interrupt Controller (NVIC), DMA (Direct Memory Access) controller, flash cache ECC, and RAM. The NVIC of PSoC 3 devices provide 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.

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.

1.4 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. An Error Correcting Code (ECC) can enable high-reliability applications.

A powerful and flexible protection model allows you 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. Additionally, selected configuration options, such as boot speed and pin drive mode, are stored in nonvolatile memory, allowing settings to become active immediately after power-on-reset (POR).

1.5 System Wide Resources

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

1.5.1 I/O Interfaces

PSoC 3 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. All I/Os have a large number of drive modes that are set at POR. PSoC 3 devices also provide up to four individual I/O voltage domains through the VDDIO pins.
- **Special Input/Output (SIO)** – The SIOs on PSoC 3 devices allow setting 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 ideal for use on an I^2C bus where the PSoC 3 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.

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 1% 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 3 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 3 devices support extensive supply operating ranges from 1.7 V to 5.5 V, allowing operation from regulated supplies such as 1.8 ± 5%, 2.5 V ± 10%, 3.3 V ± 10%, 5.0 V ± 10%, or directly from a wide range of battery types.

1.5.3.1 Boost Converter

The PSoC platform provides an integrated high-efficiency synchronous boost converter that is used to power the device from supply voltages as low as 0.5 V. This converter enables the device to power directly from a single battery or solar cell. You can employ the boost converter to generate other voltages required by the device, such as a 3.3-V supply for LCD glass drive. The boost output is available on the VBOOST pin, allowing other devices in the application to draw power from the PSoC device.

1.5.3.2 Sleep Modes

The PSoC platform supports five low-power sleep modes, from the lowest current RAM retention mode (hibernation) to the full function active mode. A 1.0-µA RTC mode runs the optional 32.768-kHz watch crystal continuously to drive the RTC timer that is used to maintain RTC. 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 3 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, Full Speed USB, and CAN 2.0b. See the device datasheet 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
- Transimpedance 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 62 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 nonlinearity (INL) less than 1 LSB, differential nonlinearity (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 Digital Filter Block

The ADC output 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.

1.7.3 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.

1.7.4 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 transimpedance amplifiers, programmable gain amplifiers, and mixers.

1.8 Program and Debug

JTAG (4-wire) or serial wire debugger (SWD) (2-wire) interfaces are 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, you can implement a full debugging interface with just three pins.

These standard interfaces enable debugging or programming the PSoC device with a variety of hardware solutions from Cypress or third party vendors.

PSoC 3 devices support on-chip break points, and an instruction and data trace memory for debug. JTAG also supports standard JTAG scan chains for board level test and chaining multiple JTAG devices.

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 (IDE) software. This technical reference manual helps to understand the details of the PSoC 3 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 option. 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 following sections include these topics:

- Section B: CPU System on page 35
- Section C: Memory on page 97
- Section D: System Wide Resources on page 121
- Section E: Digital System on page 181
- Section F: Analog System on page 311
- Section G: Program and Debug on page 403

3.1 Major Sections

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

- 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® 3 Registers TRM (Technical Reference Manual)* – Supply all device register details summarized in the technical reference manual. These are additional documents.

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® 3 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

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
BINC	bit implemented no connection
BOM	bill of materials
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CAN	controller area network
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
DC	direct current
DFB	digital filter block
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
EMIF	external memory interface
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
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

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
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
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
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

Table 3-2. Acronyms *(continued)*

Symbol	Unit of Measure
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

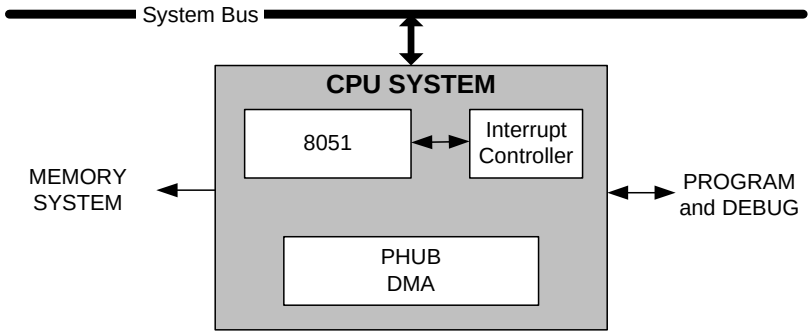
The PSoC 3 8051 CPU subsystem is built around a single cycle pipelined 8051 8-bit processor, running up to 67 MHz. The single cycle 8051 CPU runs ten times faster than a standard 8051 processor. The PSoC 3 instruction set is compatible with the original MCS-51 instruction set.

This section includes the following chapters:

- 8051 Core chapter on page 37
- PHUB and DMAC chapter on page 73
- Interrupt Controller chapter on page 89

Top Level Architecture

CPU System Block Diagram



4. 8051 Core

The PSoC® 3 8051 core is a high-performance, speed-optimized 8-bit central processing unit (CPU). It is 100% binary compatible with the industry standard 8051. The PSoC 3 family includes wrapper logic around the 8051 core. This wrapper includes internal data random access memory (RAM), an external data space interface, a Special Function Register–Input/Output (SFR–I/O) interface, and a CPU clock divider.

4.1 Features

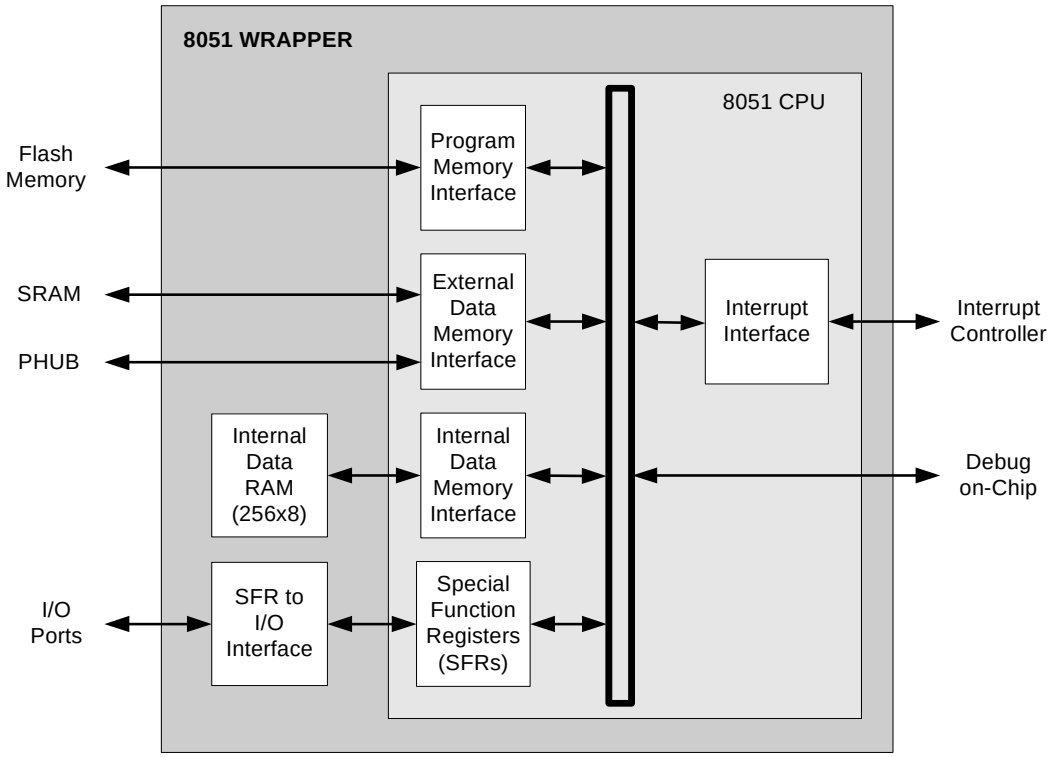
The PSoC 3 8051 has the following features:

- Pipelined RISC architecture that executes ten times faster than the industry standard 8051
- 100% binary compatible with the industry-standard 8051 instruction set
- Most instructions executed in one or two cycles
- 256 bytes of internal data RAM
- Dual DPTR extension to the standard 8051 architecture
- 24-bit external data space that enables access to on-chip memory and registers, and to off-chip memory
- New interrupt interface that enables direct interrupt vectoring. See 4.3.4 Interrupt Controller Interface on page 38
- New special function registers (SFRs) enable fast access to PSoC 3 I/O ports

4.2 Block Diagram

Figure 4-1 on page 38 shows a diagram of the wrapper around the 8051 and its interface to different blocks on the device.

Figure 4-1. 8051 Wrapper Block Diagram



4.3 How It Works

The PSoC 3 8051 core is fully compatible with the standard 8051 microcontroller, maintaining all ***instruction mnemonics*** and binary compatibility. The 8051 core incorporates enhancements that allow it to execute instructions with high performance.

4.3.1 Memory Spaces

The PSoC 3 8051 memory map is very similar to the standard 8051 memory map. There are separate address spaces for program and data memory. The data space is further divided into internal and external data spaces. See 4.5.1 Internal Data Space Map on page 39, and 4.7 Program and External Data Spaces on page 66.

4.3.2 Instruction Set

The PSoC 3 8051 core instruction set is highly optimized for 8-bit processing and Boolean operations. Types of instructions supported include:

- Arithmetic
- Logical
- Data Transfer
- Boolean
- Program Branching

See 4.5.3 Arithmetic Logic Unit Functions on page 40.

4.3.3 8051 Core Enhancements

The PSoC 3 8051 core has several enhancements:

- Dual DPTRs – see 4.6.2 Dual Data Pointer SFRs on page 63
- 24-bit external data space with DPTR extension SFRs – see 4.6.3 24-Bit Data Pointer SFRs on page 64
- Vectored interrupt controller interface removes the need for hard interrupt vectors in the program space – see 4.3.4 Interrupt Controller Interface on page 38

4.3.4 Interrupt Controller Interface

With the PSoC 3 8051, there is no need to place a JMP instruction at address zero because the interrupt controller interface jumps directly to ISRs with dynamic vector addresses. When an interrupt occurs, the current instruction is completed and the program counter pushed onto the stack. Code execution then jumps to the program address provided by the vector.

After the ISR has completed, an RETI instruction returns execution to the instruction following the previously inter-

rupted instruction, by popping the program counter from the stack.

4.4 CY 8051 Wrapper

The wrapper logic around the 8051 core provides an interface to the rest of the PSoC 3 device. See Figure 4-1 on page 38. The wrapper has the following features:

- The 8051 is one of two bus masters, the other is the DMA controller – see the PHUB and DMAC chapter on page 73
- The two bus slaves are the on-chip SRAM and the PHUB:
  - Accessed within the 8051 external data space
  - Enables access to all PSoC 3 registers and to external memory
- An SFR–I/O interface allows direct access to some I/O port registers using SFRs – see 4.4.1 SFR–I/O Interface on page 39

4.4.1 SFR–I/O Interface

Each I/O port supports two interfaces:

- SFRs in the 8051 – allow faster access to a limited set of I/O port registers
- PHUB – allows boot configuration and access to all I/O port registers

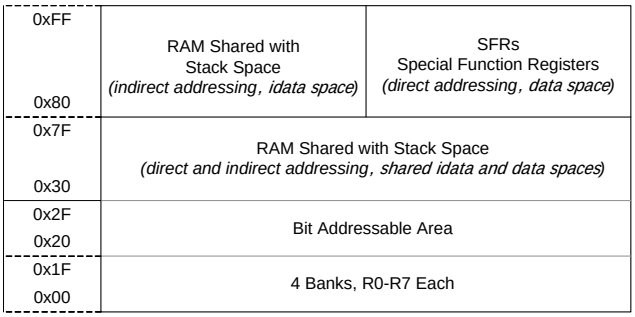
4.5 8051 Instructions

The 8051 has a full-featured set of instructions that supports a number of flexible addressing modes.

4.5.1 Internal Data Space Map

A diagram of the 8051 internal data space is shown in Figure 4-2.

Figure 4-2. 8051 Internal Data Space



4.5.2 Addressing Modes

The following addressing modes are supported by the 8051:

- Direct Addressing – The operand is specified by a direct 8-bit address field. Only the lower 128 bytes of internal RAM, and the SFRs, are accessed using this mode.
- Indirect Addressing – The instruction specifies the register that contains the address of the operand. Registers R0, R1, or SP are used to specify the 8-bit address for all 256 bytes of internal RAM. The data pointer (DPTR) register is used to specify the 16-bit address for the program and external data spaces.
- Register Addressing – Certain instructions access one of the registers in the specified register bank. These instructions are more efficient because there is no need for an address field.
- Register Specific Instructions – Some instructions are specific to certain registers. For example, some instructions always act on the accumulator. In this case, there is no need to specify the operand.
- Immediate Constants – The instruction contains a constant value.
- Bit Addressing – The operand is one of 256 bits.

4.5.3 Arithmetic Logic Unit Functions

The Arithmetic Logic Unit (ALU) section of the processor performs extensive data manipulation (see 4.5 8051 Instructions on page 39) and includes:

- 8-bit ALU – performs:
  - Typical arithmetic operations – such as addition, subtraction, multiplication, and division
  - Additional operations – such as increment, decrement, BCD decimal adjust, and compare
  - AND, OR, Exclusive OR, complement, and rotation
  - Bit operations – such as set, clear, complement, jump-if-not-set, jump-if-set-and-clear, and move to/from carry
- ACC (SFR 0xE0) register – accumulator for results of most instructions
- B (SFR 0xF0) register – used during multiply and divide operations. In other cases, it is used as a general purpose register, directly addressable
- PSW (SFR 0xD0) register – used as the program status word, which contains several bits that reflect the current state of the CPU

PSW 0xD0 (Program Status Word SFR)

	7	6	5	4	3	2	1	0
Access: POR	RW: 00							
Bit Name	CY	AC	F0	RS1	RS0	OV	F1	P

Bits	Name	Description
7	CY	Carry flag – affected by arithmetic and bit instructions
6	AC	Auxiliary carry – affected by ADD, ADDC, SUBB instructions
5	F0	General purpose flag 0
4, 3	RS[1:0]	Register bank select bits: 00 – Bank 0, data address 0x00-0x07 01 – Bank 1, data address 0x08-0x0F 10 – Bank 2, data address 0x10-0x17 11 – Bank 3, data address 0x18-0x1F
2	OV	Overflow flag – affected by ADD, ADDC, SUBB, MUL, DIV instructions
1	F1	General purpose flag 1
0	P	Parity flag – set/cleared after each instruction to indicate an odd/even number of 1s in the accumulator

4.5.3.1 Arithmetic Instructions

Arithmetic instructions support the direct, indirect, register, immediate, and register-specific addressing modes. They support addition, subtraction, multiplication, division, increment, and decrement operations. Standard 8051 8 × 8 multiplications and divisions are done in just 2 and 6 cycles, respectively. Table 4-1 lists the various arithmetic instructions.

Table 4-1. Arithmetic Instructions

Mnemonic	Description	Bytes	Cycles
ADD A,Rn	Add register to accumulator	1	1
ADD A,Direct	Add direct byte to accumulator	2	2
ADD A,@Ri	Add indirect RAM to accumulator	1	2
ADD A,#data	Add immediate data to accumulator	2	2
ADDC A,Rn	Add register to accumulator with carry	1	1
ADDC A,Direct	Add direct byte to accumulator with carry	2	2
ADDC A,@Ri	Add indirect RAM to accumulator with carry	1	2

Table 4-1. Arithmetic Instructions *(continued)*

Mnemonic	Description	Bytes	Cycles
ADDC A,#data	Add immediate data to accumulator with carry	2	2
SUBB A,Rn	Subtract register from accumulator with borrow	1	1
SUBB A,Direct	Subtract direct byte from accumulator with borrow	2	2
SUBB A,@Ri	Subtract indirect RAM from accumulator with borrow	1	2
SUBB A,#data	Subtract immediate data from accumulator with borrow	2	2
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	2
INC Direct	Increment direct byte	2	3
INC @Ri	Increment indirect RAM	1	3
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	2
DEC Direct	Decrement direct byte	2	3
DEC @Ri	Decrement indirect RAM	1	3
INC DPTR	Increment data pointer	1	1
MUL	Multiply accumulator and B	1	2
DIV	Divide accumulator by B	1	6
DAA	Decimal adjust accumulator	1	3

4.5.3.2 Logical Instructions

Logical instructions perform Boolean operations such as AND, OR, XOR, rotate, and swap of nibbles. The Boolean operations on the bytes are performed on the bit-by-bit basis. Table 4-2 lists logical instructions and their descriptions.

Table 4-2. Logical Instructions

Mnemonic	Description	Bytes	Cycles
ANL A,Rn	AND register to accumulator	1	1
ANL A,Direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2
ANL A,#data	AND immediate data to accumulator	2	2
ANL Direct, A	AND accumulator to direct byte	2	3
ANL Direct, #data	AND immediate data to direct byte	3	3
ORL A,Rn	OR register to accumulator	1	1
ORL A,Direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL A,#data	OR immediate data to accumulator	2	2
ORL Direct, A	OR accumulator to direct byte	2	3
ORL Direct, #data	OR immediate data to direct byte	3	3
XRL A,Rn	XOR register to accumulator	1	1
XRL A,Direct	XOR direct byte to accumulator	2	2
XRL A,@Ri	XOR indirect RAM to accumulator	1	2
XRL A,#data	XOR immediate data to accumulator	2	2
XRL Direct, A	XOR accumulator to direct byte	2	3
XRL Direct, #data	XOR immediate data to direct byte	3	3
CLR A	Clear accumulator	1	1

Table 4-2. Logical Instructions *(continued)*

Mnemonic	Description	Bytes	Cycles
CPL A	Complement accumulator	1	1
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right though carry	1	1
SWAP	Swap nibbles within accumulator	1	1

4.5.3.3 Data Transfer Instructions

There are three types of data transfer instructions:

- Internal data – includes transfer between any two internal RAM locations or SFRs. These instructions use direct, indirect, register, and immediate addressing.
- External data – includes only the transfer between the accumulator and the external address. It uses only the MOVX instruction.
- Lookup tables – includes only the transfer between the accumulator and the program memory address. It can use only the MOVC instruction.

Table 4-3 lists the available data transfer instructions.

Table 4-3. Data Transfer Instructions

Mnemonic	Description	Bytes	Cycles
MOV A,Rn	Move register to accumulator	1	1
MOV A,Direct	Move direct byte to accumulator	2	2
MOV A,@Ri	Move indirect RAM to accumulator	1	2
MOV A,#data	Move immediate data to accumulator	2	2
MOV Rn,A	Move accumulator to register	1	1
MOV Rn,Direct	Move direct byte to register	2	3
MOV Rn, #data	Move immediate data to register	2	2
MOV Direct, A	Move accumulator to direct byte	2	2
MOV Direct, Rn	Move register to direct byte	2	2
MOV Direct, Direct	Move direct byte to direct byte	3	3
MOV Direct, @Ri	Move indirect RAM to direct byte	2	3
MOV Direct, #data	Move immediate data to direct byte	3	3
MOV @Ri, A	Move accumulator to indirect RAM	1	2
MOV @Ri, Direct	Move direct byte to indirect RAM	2	3
MOV @Ri, #data	Move immediate data to indirect RAM	2	2
MOV DPTR, #data16	Load data pointer with 16 bit constant	3	3
MOVC A, @A+DPTR	Move code byte relative to DPTR to accumulator	1	5
MOVC A, @A + PC	Move code byte relative to PC to accumulator	1	4
MOVX A,@Ri	Move a byte from external data space to accumulator	1	4
MOVX A, @DPTR	Move a byte from external data space to accumulator	1	3
MOVX @Ri, A	Move a byte from accumulator to external RAM	1	5
MOVX @DPTR, A	Move a byte from accumulator to external RAM	1	4
PUSH Direct	Push direct byte onto stack	2	3
POP Direct	Pop direct byte from stack	2	2

Table 4-3. Data Transfer Instructions *(continued)*

Mnemonic	Description	Bytes	Cycles
XCH A, Rn	Exchange register with accumulator	1	2
XCH A, Direct	Exchange direct byte with accumulator	2	3
XCH A, @Ri	Exchange indirect RAM with accumulator	1	3
XCHD A, @Ri	Exchange low order indirect digit RAM with accumulator	1	3

4.5.3.4 Boolean Instructions1

The 8051 core has a bit addressable memory with 128 bits of bit addressable RAM (at internal data addresses 0x20 - 0x2F), and a set of SFRs that are bit addressable. The instruction set includes move, set, clear, toggle, and OR and AND instructions, as well as conditional jump instructions. An abundance of bit-level instructions makes the 8051 an excellent bit processor.

Table 4-4 lists the available Boolean instructions.

Table 4-4. Boolean Instructions

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CLR bit	Clear direct bit	2	3
SETB C	Set carry	1	1
SETB bit	Set direct bit	2	3
CPL C	Complement carry	1	1
CPL bit	Complement direct bit	2	3
ANL C, bit	AND direct bit to carry	2	2
ANL C, /bit	AND complement of direct bit to carry	2	2
ORL C, bit	OR direct bit to carry	2	2
ORL C, /bit	OR complement of direct bit to carry	2	2
MOV C, bit	Move direct bit to carry	2	2
MOV bit, C	Move carry to direct bit	2	3
JC rel	Jump if carry is set	2	3
JNC rel	Jump if no carry is set	2	3
JB bit, rel	Jump if direct bit is set	3	5
JNB bit, rel	Jump if direct bit is not set	3	5
JBC bit, rel	Jump if direct bit is set and clear bit	3	5

4.5.3.5 Program Branching Instructions

The 8051 supports a set of conditional and unconditional jump instructions to modify the program execution flow.

Table 4-5 shows the list of jump instructions.

Table 4-5. Jump Instructions

Mnemonic	Description	Bytes	
ACALL addr11	Absolute subroutine call	2	4
LCALL addr16	Long subroutine call	3	4
RET	Return from subroutine	1	4
RETI	Return from interrupt	1	4
AJMP addr11	Absolute jump	2	3

Table 4-5. Jump Instructions *(continued)*

Mnemonic	Description	Bytes	
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (relative address)	2	3
JMP @A + DPTR	Jump Indirect relative to DPTR	1	5
JZ rel	Jump if accumulator is zero	2	4
JNZ rel	Jump if accumulator is non zero	2	4
CJNE A,Direct, rel	Compare direct byte to accumulator and jump if not equal	3	5
CJNE A, #data, rel	Compare immediate data to accumulator and jump if not equal	3	4
CJNE Rn, #data, rel	Compare immediate data to register and jump if not equal	3	4
CJNE @Ri, #data, rel	Compare immediate data to indirect RAM and jump if not equal	3	5
DJNZ Rn,rel	Decrement register and jump if non zero	2	4
DJNZ Direct, rel	Decrement direct byte and jump if non zero	3	5
NOP	No operation	1	1

4.5.3.6 Instruction Set Details

The following instructions are supported by the 8051 and are listed in alphabetical order.

ACALL addr11

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ACALL addr11	Absolute call	(PC) ← (PC) + 2 (SP) ← (SP) + 1 ((SP)) ← (PC7-0) (SP) ← (SP) + 1 ((SP)) ← (PC15-8) (PC10-0) ← page address	None	0x11, 0x31, 0x51, 0x71, 0x91, 0xB1, 0xD1, 0xF1	2	4

Unconditionally calls a subroutine located at the indicated address. The destination address is obtained by concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must start within the same 2K block of program memory as the first byte of the instruction following the ACALL.

ADD A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADD A, Rn	Add a register to ACC	(PC) ← (PC) + 1 (A) ← (A) + (Rn)	C, AC, OV	0x28 – 0x2F	1	1

Adds the register indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the overflow (OV) flag indicates an overflow occurred. OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

ADD A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADD A, direct	Add a direct byte to ACC	(PC) ← (PC) + 2 (A) ← (A) + (direct)	C, AC, OV	0x25	2	2

Adds the direct byte indicated to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

ADD A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADD A, @Ri	Add an indirect byte to ACC	(PC) ← (PC) + 1 (A) ← (A) + ((Ri))	C, AC, OV	0x26, 0x27	1	2

Adds a byte pointed to by R0 or R1 to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

ADD A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADD A, #data	Add an immediate byte to ACC	(PC) ← (PC) + 2 (A) ← (A) + data	C, AC, OV	0x24	2	2

Adds an immediate byte (the second byte of the instruction) to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

ADDC A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADDC A, Rn	Add a register and C to ACC	(PC) ← (PC) + 1 (A) ← (A) + (C) + (Rn)	C, AC, OV	0x38 – 0x3F	1	1

Adds the register indicated, and the carry flag, to the accumulator, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the overflow flag indicates an overflow occurred. OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

ADDC A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADDC A, direct	Add a direct byte and C to ACC	(PC) ← (PC) + 2 (A) ← (A) + (C) + (direct)	C, AC, OV	0x35	2	2

Adds the direct byte indicated, and the carry flag, to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

ADDC A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADDC A, @Ri	Add an indirect byte and C to ACC	(PC) ← (PC) + 1 (A) ← (A) + (C) + ((Ri))	C, AC, OV	0x36, 0x37	1	2

Adds a byte pointed to by R0 or R1, and the carry flag, to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

ADDC A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ADDC A, #data	Add an immediate byte and C to ACC	(PC) ← (PC) + 2 (A) ← (A) + (C) + data	C, AC, OV	0x34	2	2

Adds an immediate byte (the second byte of the instruction), and the carry flag, to the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

AJMP addr11

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
AJMP addr11	Absolute jump	(PC) ← (PC) + 2 (PC10-0) page address	None	0x01, 0x21, 0x41, 0x61, 0x81, 0xA1, 0xC1, 0Xe1	2	3

Unconditionally transfers program control to the indicated address. The address is obtained by concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The destination must be within the same 2K block of program memory as the first byte of the instruction following the AJMP.

ANL A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL A, Rn	Logical AND for byte operands	(PC) ← (PC) + 1 (A) ← (A) and (Rn)	None	0x58 – 0x5F	1	1

Performs a bitwise logical AND operation between the accumulator and a register, leaving the result in the accumulator.

ANL A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL A, direct	Logical AND for byte operands	PC) ← (PC) + 2 (A) ← (A) and (direct)	None	0x55	2	2

Performs a bitwise logical AND operation between the accumulator and a direct byte, leaving the result in the accumulator.

ANL A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL A, @Ri	Logical AND for byte operands	(PC) ← (PC) + 1 (A) ← (A) and ((Ri))	None	0x56, 0x57	1	2

Performs a bitwise logical AND operation between the accumulator and a byte pointed to by R0 or R1, leaving the result in the accumulator.

ANL A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL A, #data	Logical AND for byte operands	(PC) ← (PC) + 2 (A) ← (A) and data	None	0x54	2	2

Performs a bitwise logical AND operation between the accumulator and an immediate byte (the second byte of the instruction), leaving the result in the accumulator.

ANL direct A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL direct A	Logical AND for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) and (A)	None	0x52	2	3

Performs a bitwise logical AND operation between a direct byte and the accumulator, leaving the result in the direct byte.

ANL direct, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL direct, #data	Logical AND for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) and data	None	0x53	3	3

Performs a bitwise logical AND operation between a direct byte and an immediate byte (the third byte of the instruction), leaving the result in the direct byte.

ANL C, bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL C, bit	Logical AND for bit operands	(PC) ← (PC) + 2 (C) ← (C) and (bit)	C	0x82	2	2

Performs a bitwise logical AND operation between the carry flag and a bit, leaving the result in the carry flag.

ANL C, /bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ANL C, /bit	Logical AND for bit operands	(PC) ← (PC) + 2 (C) ← (C) and / (bit)	C	0xB0	2	2

Performs a bitwise logical AND operation between the carry flag and the inversion of a bit, leaving the result in the carry flag.

CJNE A, direct, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles

CJNE A, direct, rel	Compare and jump if not equal	(PC) ← (PC) + 3
		If (A) ≠ (direct) then
		  (PC) ← (PC) + rel
		If (A) < (direct) then
		  (C) ← 1
		Else
		  (C) ← 0
			C	0xB5	3	5

Compares the magnitudes of the accumulator and the direct byte, and branches if their values are not equal. The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of the accumulator is less than the unsigned integer value of the direct byte; otherwise, the carry is cleared. Neither operand is affected.

CJNE A, #data, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles

CJNE A, #data, rel	Compare and jump if not equal	(PC) ← (PC) + 3
		If (A) ≠ data then
		  (PC) ← (PC) + rel
		If (A) < data then
		  (C) ← 1
		Else
		  (C) ← 0
			C	0xB4	3	4

Compares the magnitudes of the accumulator and the immediate byte (the second byte of the instruction), and branches if their values are not equal. The branch destination and carry flag are set as described above. The accumulator is not affected.

CJNE Rn, #data, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles

CJNE Rn, #data, rel	Compare and jump if not equal	(PC) ← (PC) + 3
		If (Rn) ≠ data then
		(PC) ← (PC) + rel
		If (Rn) < data then
		(C) ← 1
		Else
		(C) ← 0
			C	0xB8 – 0xBF	3	4

Compares the magnitudes of the indicated register and the immediate byte (the second byte of the instruction), and branches if their values are not equal. The branch destination and carry flag are set as described above. The register is not affected.

CJNE @Ri, #data, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles

CJNE @Ri, #data, rel	Compare and jump if not equal	(PC) ← (PC) + 3
		If ((Ri)) ≠ data then
		(PC) ← (PC) + rel
		If ((Ri)) < data then
		(C) ← 1
		Else
		(C) ← 0
			C	0xB6, 0xB7	3	5

Compares the magnitudes of the byte pointed to by R0 or R1 and the immediate byte (the second byte of the instruction), and branches if their values are not equal. The branch destination and carry flag are set as described above. The byte pointed to by R0 or R1 is not affected.

CLR A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CLR A	Clear accumulator	(PC) ← (PC) + 1 (A) ← 0	None	None	1	1

All bits of the accumulator are cleared (set to zero).

CLR bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CLR bit	Clear bit	(PC) ← (PC) + 2 (bit) ← 0	None	0xC2	2	3

The indicated bit is cleared (set to zero).

CLR C

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CLR C	Clear carry	(PC) ← (PC) + 1 (C) ← 0	None	0xC3	1	1

The carry flag is cleared (set to zero).

CPL A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CPL A	Complement accumulator	(PC) ← (PC) + 1 (A) ← /(A)	None	0xF4	1	1

Each bit of the accumulator is logically complemented (one's complement). Bits that previously contained a one are changed to zero and vice versa.

CPL bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CPL bit	Complement bit	(PC) ← (PC) + 2 (bit) ← /(bit)	None	0xB2	2	3

The bit variable specified is complemented. A bit that was a one is changed to zero and vice versa. CPL can operate on the carry or any directly addressable bit. When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

CPL C

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
CPL C	Complement carry	(PC) ← (PC) + 1 (C) ← /(C)	C	0xB3	1	1

The carry flag is complemented. A bit that was a one is changed to zero and vice versa.

DAA

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles

DA A	Decimal adjust accumulator for addition	(PC) ← (PC) + 1
		if [[(A3-0) > 9] ^ [(AC) = 1]] then
		(A3-0) ← (A3-0) + 6
		next
		if [[(A7-4) > 9] ^ [(C) = 1]] then
		(A7-4) ← (A7-4) + 6
			C	0xD4	1	3

Adjusts the value in the accumulator resulting from the earlier addition of two variables (each in packed BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition. If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition sets the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but does not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this sets the carry flag if there was a carry-out of the high-order bits, but does not clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially; this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions. DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

DEC A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DEC A	Decrement accumulator	(PC) ← (PC) + 1 (A) ← (A) – 1	None	0x14	1	1

The accumulator is decremented by 1. An original value of 0 will underflow to 0xFF.

DEC Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DEC Rn	Decrement register	(PC) ← (PC) + 1 (Rn) ← (Rn) – 1	None	0x18 – 0x1F	1	2

The indicated register is decremented by 1. An original value of 0 will underflow to 0xFF.

DEC direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DEC direct	Decrement direct byte	(PC) ← (PC) + 2 (direct) ← (direct) – 1	None	0x15	2	3

The indicated direct byte is decremented by 1. An original value of 0 will underflow to 0xFF. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

DEC @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DEC @Ri	Decrement indirect byte	(PC) ← (PC) + 1 ((Ri)) ← ((Ri)) – 1	None	0x16, 0x17	1	3

The byte pointed to by R0 or R1 is decremented by 1. An original value of 0 will underflow to 0xFF. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

DIV

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DIV	Divide	(PC) ← (PC) + 1 (A15-8) ← (A) / (B) result's bits 15..8 (B7-0) ← (A) / (B) result's bits 7..0	C, OV	0x84	1	6

Divides the unsigned integer in the accumulator by the unsigned integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. If B had originally contained 0, the values returned in the accumulator and register B are undefined and the overflow flag is set. Otherwise the overflow flag is cleared. The carry flag is cleared.

DJNZ Rn, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DJNZ Rn, rel	Decrement and jump if not zero	(PC) ← (PC) + 2 (Rn) ← (Rn) - 1 if (Rn) ≠ 0 then (PC) ← (PC) + rel	None	0xD8 – 0xDF	2	4

Decrements the register indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 0 will underflow to 0xFF. The branch destination is computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

DJNZ direct, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
DJNZ direct, rel	Decrement and jump if not zero	(PC) ← (PC) + 3 (direct) ← (direct) - 1 if (direct) ≠ 0 then (PC) ← (PC) + rel	None	0xD5	3	5

Decrements the direct byte indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 0 will underflow to 0xFF. The branch destination is computed as described above. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

INC A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
INC A	Increment accumulator	(PC) ← (PC) + 1 (A) ← (A) + 1	None	0x04	1	1

The accumulator is incremented by 1. An original value of 0xFF will overflow to 0.

INC Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
INC Rn	Increment register	(PC) ← (PC) + 1 (Rn) ← (Rn) + 1	None	0x08 – 0x0F	1	2

The indicated register is incremented by 1. An original value of 0xFF will overflow to 0.

INC direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
INC direct	Increment direct byte	(PC) ← (PC) + 2 (direct) ← (direct) + 1	None	0x05	2	3

The indicated direct byte is incremented by 1. An original value of 0xFF will overflow to 0. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

INC @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
INC @Ri	Increment indirect byte	(PC) ← (PC) + 1 ((Ri)) ← ((Ri)) + 1	None	0x06, 0x07	1	3

The byte pointed to by R0 or R1 is incremented by 1. An original value of 0xFF will overflow to 0. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

INC DPTR

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
INC DPTR	Increment data pointer	(PC) ← (PC) + 1 (DPTR) ← (DPTR) + 1	None	0xA3	1	1

Increment the 16-bit data pointer by 1. A 16-bit increment is performed; an overflow of the low-order byte of the data pointer (DPL) from 0xFF to 0 will increment the high-order byte (DPH). This is the only 16-bit register that can be incremented.

JB bit, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JB bit, rel	Jump if bit is set	(PC) ← (PC) + 3 if (bit) = 1 then (PC) ← (PC) + rel	None	0x20	3	5

If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified.

JBC bit, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JBC bit, rel	Jump if bit is set and clear bit	(PC) ← (PC) + 3 if (bit) = 1 then bit ← 0 (PC) ← (PC) + rel	None	0x10	3	5

If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed as described above. When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

JC rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JC rel	Jump if carry is set	(PC) ← (PC) + 2 if (C) = 1 then (PC) ← (PC) + rel	None	0x40	2	3

If the carry flag is set, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed as described above.

JMP @A + DPTR

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JMP @A + DPTR	Jump indirect	(PC) ← (A) + (DPTR)	None	0x73	1	5

Add the 8-bit unsigned contents of the accumulator with the 16-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. 16-bit addition is performed: a carry-out from the low-order 8 bits propagates through the high-order bits. Neither the accumulator nor the data pointer is altered.

JNB bit, rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JNB bit, rel	Jump if bit is not set	(PC) ← (PC) + 3 if (bit) = 0 then (PC) ← (PC) + rel	None	0x30	3	5

If the indicated bit is a zero, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified.

JNC rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JNC rel	Jump if carry is not set	(PC) ← (PC) + 2 if (C) = 0 then (PC) ← (PC) + rel	None	0x50	2	3

If the carry flag is set, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed as described above.

JNZ rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JNZ rel	Jump if accumulator is not zero	(PC) ← (PC) + 2 if (A) ≠ 0 then (PC) ← (PC) + rel	None	0x70	2	4

If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed as described above. The accumulator is not modified.

JZ rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
JZ rel	Jump if accumulator is zero	(PC) ← (PC) + 2 if (A) = 0 then (PC) ← (PC) + rel	None	0x60	2	4

If all bits of the accumulator are zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed as described above. The accumulator is not modified.

LCALL addr16

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
LCALL addr16	Long call	(PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC7-0) (SP) ← (SP) + 1 ((SP)) ← (PC15-8) (PC) ← addr15-0	None	0x12	3	4

Calls a subroutine located at the indicated address. The high-order and low-order bytes of the PC are loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64 KB program memory address space.

LJMP addr16

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
LJMP addr16	Long jump	(PC) ← addr15…addr0	None	0x02	3	4

Does an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space.

MOV A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV A, Rn	Copy a register to ACC	(PC) ← (PC) + 1 (A) ← (Rn)	None	0xE8 – 0xEF	1	1

Copies the register indicated to the accumulator. The register is not affected.

MOV A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV A, direct	Copy a direct byte to ACC	(PC) ← (PC) + 2 (A) ← (direct)	None	0xE5	2	2

Copies the direct byte indicated to the accumulator. The direct byte is not affected.

MOV A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV A, @Ri	Copy an indirect byte to ACC	(PC) ← (PC) + 1 (A) ← ((Ri))	None	0xE6, 0xE7	1	2

Copies a byte pointed to by R0 or R1 to the accumulator. The byte pointed to by R0 or R1 is not affected.

MOV A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV A, #data	Load ACC with immediate data	(PC) ← (PC) + 2 (A) ← data	None	0xE4	2	2

Loads the accumulator with an immediate byte (the second byte of the instruction).

MOV Rn, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV Rn, A	Copy ACC to a register	(PC) ← (PC) + 1 (Rn) ← (A)	None	0xF8 – 0xFF	1	1

Copies the accumulator to the register indicated. The accumulator is not affected.

MOV Rn, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV Rn, direct	Copy a direct byte to a register	(PC) ← (PC) + 2 (Rn) ← (direct)	None	0xA8 – 0xAF	2	3

Copies the direct byte indicated to the register indicated. The direct byte is not affected.

MOV Rn, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV Rn, #data	Load a register with immediate data	(PC) ← (PC) + 2 (Rn) ← data	None	0x78 – 0x7F	2	2

Loads the register indicated with an immediate byte (the second byte of the instruction).

MOV direct, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV direct, A	Copy ACC to a direct byte	(PC) ← (PC) + 2 (direct) ← (A)	None	0xF5	2	2

Copies the accumulator to the direct byte indicated. The accumulator is not affected.

MOV direct, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV direct, Rn	Copy a register to a direct byte	(PC) ← (PC) + 2 (direct) ← (Rn)	None	0x88 – 0x8F	2	2

Copies the register indicated to the direct byte indicated. The register is not affected.

MOV direct, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV direct, direct	Copy a direct byte to a direct byte	(PC) ← (PC) + 3 (direct) ← (direct)	None	0x85	3	3

Copies the direct source byte indicated to the direct destination byte indicated. The direct source byte is not affected.

MOV direct, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV direct, @Ri	Copy an indirect byte to a direct byte	(PC) ← (PC) + 2 (direct) ← ((Ri))	None	0x86, 0x87	2	3

Copies the byte pointed to by R0 or R1 to the direct byte indicated. The byte pointed to by R0 or R1 is not affected.

MOV direct, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV direct, #data	Load a direct byte with immediate data	(PC) ← (PC) + 3 (direct) ← data	None	0x75	3	3

Loads the direct byte indicated with an immediate byte (the third byte of the instruction).

MOV @Ri, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV @Ri, A	Copy ACC to an indirect byte	(PC) ← (PC) + 1 ((Ri)) ← (A)	None	0xF6, 0xF7	1	2

Copies the accumulator to a byte pointed to by R0 or R1. The accumulator is not affected.

MOV @Ri, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV @Ri, direct	Copy a direct byte to an indirect byte	(PC) ← (PC) + 2 ((Ri)) ← (direct)	None	0xA6, 0xA7	2	3

Copies the direct byte indicated to a byte pointed to by R0 or R1. The direct byte is not affected.

MOV @Ri, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV @Ri, #data	Load an indirect byte with immediate data	(PC) ← (PC) + 2 ((Ri)) ← data	None	0x76, 0x77	2	2

Loads a byte pointed to by R0 or R1 with an immediate byte (the second byte of the instruction).

MOV C, bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV C, bit	Copy a bit to C	(PC) ← (PC) + 2 (C) ← (bit)	C	0xA2	2	2

The Boolean variable indicated (directly addressable bit) is copied into the carry flag.

MOV bit, C

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV bit, C	Copy C to a bit	(PC) ← (PC) + 2 (bit) ← (C)	None	0x92	2	3

The carry flag is copied into the Boolean variable indicated (directly addressable bit).

MOV DPTR, #data16

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOV DPTR, #data16	Load DPTR with immedi-ate data	(PC) ← (PC) + 3 DPH ← immediate data15...8 DPL ← immediate data7...0	None	0x85	3	3

Loads the data pointer with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. This is the only instruction that moves 16 bits of data at once.

MOVC A, @A + DPTR

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVC A, @A + DPTR	Load ACC with a code byte	(PC) ← (PC) + 1 (A) ← ((A) + (DPTR))	None	0x93	1	5

Loads the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned accumulator contents and the contents of the 16-bit DPTR. A 16-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits.

MOVC A, @A + PC

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVC A, @A + PC	Load ACC with a code byte	(PC) ← (PC) + 1 (A) ← ((A) + (PC))	None	0x83	1	4

Loads the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned accumulator contents and the contents of the 16-bit PC. The PC is incremented to the address of the following instruction before being added to the accumulator. 16-bit addition is performed so a carry-out from the low-order eight bits may propagate through high-order bits.

MOVX A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVX A, @Ri	Copy an external byte to ACC	(PC) ← (PC) + 1 (A) ← (MXAX : P2 : (Ri))	None	0xE2, 0xE3	1	3

Copies a byte of external data memory to the accumulator. The 24-bit external address is formed by concatenating the MXAX register (SFR address 0xEA), the P2AX register (SFR address 0xA0), and the contents of R0 or R1. The external byte is not affected.

MOVX A, @DPTR

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVX A, @DPTR	Copy an external byte to ACC	(PC) ← (PC) + 1 (A) ← (DPX : DPTR)	None	0xE0	1	2

Copies a byte of external data memory to the accumulator. The 24-bit external address is formed by concatenating the DPX register (SFR address 0x93 or 0x95) and the contents of DPTR. The external byte is not affected.

MOVX @Ri, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVX @Ri, A	Copy ACC to an external byte	(PC) ← (PC) + 1 (MXAX : P2 : (Ri)) ← (A)	None	0xF2, 0xF3	1	4

Copies the accumulator to the external data memory address indicated. The 24-bit external address is formed as described in MOVX A, @Ri above. The accumulator is not affected.

MOVX @DPTR, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MOVX @DPTR, A	Copy ACC to an external byte	(PC) ← (PC) + 1 (DPX : DPTR) ← (A)	None	0xF0	1	3

Copies the accumulator to the external data memory address indicated. The 24-bit external address is formed as described above. The accumulator is not affected.

MUL

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
MUL	Multiply	(PC) ← (PC) + 1 (A) ← (A) x (B) result's bits 7…0 (B) ← (A) x (B) result's bits 15…8	C, OV	0xA4	1	2

Multiplies the unsigned 8-bit integers in the accumulator and register B. The low-order byte of the 16-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0xFF) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

NOP

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
NOP	No operation	(PC) ← (PC) + 1	None	0x00	1	1

No operation. Execution continues at the following instruction.

ORL A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL A, Rn	Logical OR for byte operands	(PC) ← (PC) + 1 (A) ← (A) or (Rn)	None	0x48 – 0x4F	1	1

Performs a bitwise logical OR operation between the accumulator and a register, leaving the result in the accumulator.

ORL A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL A, direct	Logical OR for byte operands	(PC) ← (PC) + 2 (A) ← (A) or (direct)	None	0x45	2	2

Performs a bitwise logical OR operation between the accumulator and a direct byte, leaving the result in the accumulator.

ORL A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL A, @Ri	Logical OR for byte operands	(PC) ← (PC) + 1 (A) ← (A) or ((Ri))	None	0x46, 0x47	1	2

Performs a bitwise logical OR operation between the accumulator and a byte pointed to by R0 or R1, leaving the result in the accumulator.

ORL A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL A, #data	Logical OR for byte operands	(PC) ← (PC) + 2 (A) ← (A) or data	None	0x44	2	2

Performs a bitwise logical OR operation between the accumulator and an immediate byte (the second byte of the instruction), leaving the result in the accumulator.

ORL direct, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL direct, A	Logical OR for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) or (A)	None	0x42	2	3

Performs a bitwise logical OR operation between a direct byte and the accumulator, leaving the result in the direct byte. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

ORL direct, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL direct, #data	Logical OR for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) or data	None	0x43	3	3

Performs a bitwise logical OR operation between a direct byte and an immediate byte (the third byte of the instruction), leaving the result in the direct byte. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

ORL C, bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL C, bit	Logical OR for bit operands	(PC) ← (PC) + 2 (C) ← (C) or (bit)	C	0x72	2	2

Performs a bitwise logical OR operation between the carry flag and a bit, leaving the result in the carry flag.

ORL C, /bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
ORL C, /bit	Logical OR for bit operands	(PC) ← (PC) + 2 (C) ← (C) or / (bit)	C	0xA0	2	2

Performs a bitwise logical OR operation between the carry flag and the inversion of a bit, leaving the result in the carry flag.

POP direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
POP direct	Pop from stack	(PC) ← (PC) + 2 (direct) ← ((SP)) (SP) ← (SP) – 1	None	0xD0	2	2

The contents of the internal RAM location addressed by the stack pointer are read, and the stack pointer is decremented by one. The value read is copied to the direct byte indicated.

PUSH direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
PUSH direct	Push to stack	(PC) ← (PC) + 2 (SP) ← (SP) + 1 ((SP)) ← (direct)	None	0xC0	2	3

The stack pointer is incremented by one. The contents of the direct byte indicated are then copied into the internal RAM location addressed by the stack pointer.

RET

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RET	Return from subroutine	(PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) - 1	None	0x22	1	4

Pops the high and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL.

RETI

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RETI	Return from interrupt	(PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) - 1	None	0x32	1	4

Pops the high and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower or same-level interrupt is pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

RL A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RL A	Rotate ACC left	(PC) ← (PC) + 1 (An + 1) ← (An) n = 0-6 (A0) ← (A7)	None	0x23	1	1

The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position.

RLC A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RLC A	RLC A	(PC) ← (PC) + 1 (An + 1) ← (An) n = 0-6 (A0) ← (C) (C) ← (A7)	C	0x33	1	1

The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position.

RR A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RR A	Rotate ACC right	(PC) ← (PC) + 1 (An) ← (An + 1) n = 0-6 (A7) ← (A0)	None	0x03	1	1

The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position.

RRC A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
RRC A	Rotate ACC right through C	(PC) ← (PC) + 1 (An) ← (An + 1) n = 0-6 (A7) ← (C) (C) ← (A0)	C	0x13	1	1

The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original state of the carry flag moves into the bit 7 position.

SETB bit

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SETB bit	Set bit	(PC) ← (PC) + 2 (bit) ← 1	None	0xD2	2	3

The indicated bit is set (to one).

SETB C

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SETB C	Set carry	(PC) ← (PC) + 1 (C) ← 1	None	0xD3	1	1

The carry flag is set (to one).

SJMP rel

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SJMP rel	Short jump	(PC) ← (PC) + 2 (PC) ← (PC) + rel	None	0x80	2	3

Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it. Note the an SJMP with a displacement of 0xFE is a one-instruction infinite loop.

SUBB A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SUBB A, Rn	Subtract a register and C from ACC	(PC) ← (PC) + 1 (A) ← (A) - (C) - (Rn)	C, AC, OV	0x98 – 0x9F	1	1

Subtracts the register indicated, and the carry flag, from the accumulator, leaving the result in the accumulator. The carry (borrow) flag is set if a borrow is needed for bit 7, and otherwise C is cleared. (If C was set before executing the instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6 but not into bit 7, or into bit 7 but not bit 6. When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

SUBB A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SUBB A, direct	Subtract a direct byte and C from ACC	(PC) ← (PC) + 2 (A) ← (A) - (C) - (direct)	C, AC, OV	0x95	2	2

Subtracts the direct byte indicated, and the carry flag, from the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

SUBB A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SUBB A, @Ri	Subtract an indirect byte and C from ACC	(PC) ← (PC) + 1 (A) ← (A) - (C) - ((Ri))	C, AC, OV	0x96, 0x97	1	2

Subtracts a byte pointed to by R0 or R1, and the carry flag, from the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

SUBB A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SUBB A, #data	Subtract an immediate byte and C from ACC	(PC) ← (PC) + 2 (A) ← (A) - (C) - data	C, AC, OV	0x94	2	2

Subtracts an immediate byte (the second byte of the instruction), and the carry flag, from the accumulator, leaving the result in the accumulator. The carry, auxiliary carry, and overflow flags are set as described above.

SWAP

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
SWAP	Swap nibbles within ACC	(PC) ← (PC) + 1 (A3-0) ← (A7-4), (A7-4) ← (A3-0)	None	0xC4	1	1

SWAP A interchanges the low and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction.

XCH A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XCH A, Rn	Exchange a register with ACC	(PC) ← (PC) + 1 (A) ← (Rn)	None	0xC8 – 0xCF	1	2

Exchanges the register indicated with the accumulator.

XCH A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XCH A, direct	Exchange a direct byte with ACC	(PC) ← (PC) + 2 (A) ← (direct)	None	0xC5	2	3

Exchanges the direct byte indicated with the accumulator.

XCH A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XCH A, @Ri	Exchange an indirect byte with ACC	(PC) ← (PC) + 1 (A) ← ((Ri))	None	0xC6, 0xC7	1	3

Exchanges a byte pointed to by R0 or R1 with the accumulator.

XCHD A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XCHD A, @Ri	Exchange a digit	(PC) ← (PC) + 1 (A3-0) ← ((Ri)3-0)	None	0xD6, 0xD7	1	3

XCHD exchanges the low-order nibble of the accumulator (bits 3-0, generally representing a hexadecimal or BCD digit), with that of the byte pointed to by R0 or R1. The high-order nibbles (bits 7-4) are not affected.

XRL A, Rn

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL A, Rn	Logical exclusive OR for byte operands	(PC) ← (PC) + 1 (A) ← (A) xor (Rn)	None	0x68 – 0x6F	1	1

Performs a bitwise logical exclusive OR operation between the accumulator and a register, leaving the result in the accumulator.

XRL A, direct

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL A, direct	Logical exclusive OR for byte operands	(PC) ← (PC) + 2 (A) ← (A) xor (direct)	None	0x65	2	2

Performs a bitwise logical exclusive OR operation between the accumulator and a direct byte, leaving the result in the accumulator.

XRL A, @Ri

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL A, @Ri	Logical exclusive OR for byte operands	(PC) ← (PC) + 1 (A) ← (A) xor ((Ri))	None	0x66, 0x67	1	2

Performs a bitwise logical exclusive OR operation between the accumulator and a byte pointed to by R0 or R1, leaving the result in the accumulator.

XRL A, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL A, #data	Logical exclusive OR for byte operands	(PC) ← (PC) + 2 (A) ← (A) xor data	None	0x64	2	2

Performs a bitwise logical exclusive OR operation between the accumulator and an immediate byte (the second byte of the instruction), leaving the result in the accumulator.

XRL direct, A

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL direct, A	Logical exclusive OR for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) xor (A)	None	0x62	2	3

Performs a bitwise logical exclusive OR operation between a direct byte and the accumulator, leaving the result in the direct byte. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

XRL direct, #data

Mnemonic	Function	Operation	Flags	Opcodes	Bytes	Cycles
XRL direct, #data	Logical exclusive OR for byte operands	(PC) ← (PC) + 2 (direct) ← (direct) xor data	None	0x63	3	3

Performs a bitwise logical exclusive OR operation between a direct byte and an immediate byte (the third byte of the instruction), leaving the result in the direct byte. When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

4.6 8051 Special Function Registers (SFRs)

The Special Function Registers (SFRs) provide access to I/Os and other functions. All 8051 registers except the PC - ACC, B, PSW, SP, DPTR - can also be accessed as SFRs.

4.6.1 SFRs

Table 4-6 shows the map for the SFRs space.

Table 4-6. SFR Map

	0/8 (Bit Addressable)	1/9	2/A	3/B	4/C	5/D	6/E	7/F
0xF8	SFRPRT15DR	SFRPRT15PS	SFRPRT15SEL					
0xF0	B		SFRPRT12SEL					
0xE8	SFRPRT12DR	SFRPRT12PS	MXAX					
0xE0	ACC							
0xD8	SFRPRT6DR	SFRPRT6PS	SFRPRT6SEL					
0xD0	PSW							
0xC8	SFRPRT5DR	SFRPRT5PS	SFRPRT5SEL					
0xC0	SFRPRT4DR	SFRPRT4PS	SFRPRT4SEL					
0xB8								
0xB0	SFRPRT3DR	SFRPRT3PS	SFRPRT3SEL					
0xA8	IE							
0xA0	P2AX		SFRPRT1SEL					
0x98	SFRPRT2DR	SFRPRT2PS	SFRPRT2SEL					
0x90	SFRPRT1DR	SFRPRT1PS		DPX0		DPX1		
0x88		SFRPRT0PS	SFRPRT0SEL					
0x80	SFRPRT0DR	SP	DPL0	DPH0	DPL1	DPH1	DPS	

4.6.2 Dual Data Pointer SFRs

Dual data pointer (DPTR) SFRs are implemented to speed up data block copying. DPTR0 and DPTR1 are located at four SFR addresses. The active DPTR register is selected by the SEL bit (0x86.0). If the SEL bit is equal to 0, DPTR0 (SFRs 0x83:0x82) is selected; if not, DPTR1 (SFRs 0x85:0x84) is used.

4.6.2.1 DPTR0 (Data Pointer 0 SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	DPH0 (0x83)							
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Access: POR	R/W:00							
Name	DPL0 (0x82)							

Bits	Name	Description
7:0	DPH0[7:0]	Upper byte of DPTR0 register
7:0	DPL0[7:0]	Lower byte of DPTR0 register

4.6.2.2 DPTR1 (Data Pointer 1 SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	DPH1 (0x85)							
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Access: POR	R/W:00							
Name	DPL1 (0x84)							

Bits	Name	Description
7:0	DPH1[7:0]	Upper byte of DPTR1 register
7:0	DPL1[7:0]	Lower byte of DPTR1 register

4.6.2.3 DPS 0x86 (Data Pointer Select SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name								SEL

Bits	Name	Description
7:1		Reserved
0	SEL	Select current DPTR

The data pointer select register is used in the following instructions:

- MOVX @DPTR,A
- MOVX A, @DPTR
- MOVC A, @A+DPTR
- JMP @A+DPTR
- INC DPTR
- MOV DPTR, #data16

4.6.3 24-Bit Data Pointer SFRs

Extended data pointer SFRs DPX0, DPX1, MXAX, and P2AX hold the most significant parts of memory addresses during access to the external data memory space. After reset, each of these registers have 0x00 values.

4.6.3.1 DPX0 0x93 (Data Pointer 0 eXtended SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	DPX0							

Bits	Name	Description
7:0	DPX0[7:0]	During MOVX instruction using DPTR0 register, the most significant part of address XRAMADDR[23:16] is always equal to the contents of DPX0 (SFR 0x93).

4.6.3.2 DPX1 0x95 (Data Pointer 1 eXtended SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	DPX1							

Bits	Name	Description
7:0	DPX1[7:0]	During MOVX instruction using DPTR1 register, the most significant part of address XRAMADDR[23:16] is always equal to the contents of DPX1 (SFR 0x95).

4.6.3.3 MXAX 0xEA (MOVX @Ri eXtended SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	MXAX							

Bits	Name	Description
7:0	MXAX[7:0]	During MOVX using R0 or R1 register, XRAMADDR[23:16] is always equal to contents of MXAX (SFR 0xEA).

4.6.3.4 P2AX 0xA0 (P2 Read-Write SFR)

Bits	7	6	5	4	3	2	1	0
Access: POR	R/W:00							
Name	P2AX							

Bits	Name	Description
7:0	P2AX[7:0]	During MOVX using R0 or R1 register, XRAMADDR[15:8] is always equal to the contents of P2AX (SFR 0xA0).

During a MOVX instruction using the DPTR0/DPTR1 register, XRAMADDR[23:16] is always equal to the contents of DPX0 (SFR 0x93) / DPX1 (SFR 0x95).

During a MOVX instruction using the R0 or R1 register, XRAMADDR[23:16] is always equal to the contents of MXAX (SFR 0xEA), and XRAMADDR[15:8] is always equal to the contents of P2AX (SFR 0xA0).

4.6.4 I/O Port Access SFRs

Each I/O port supports two interfaces:

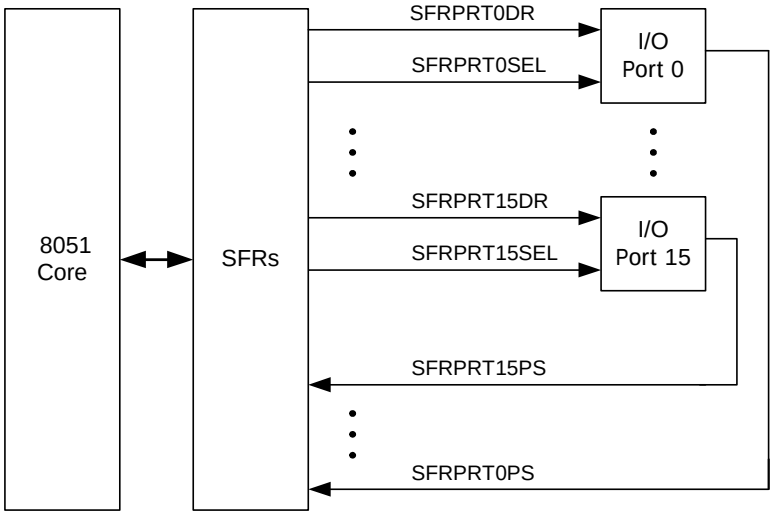
- PHUB bus – allows boot configuration and access to all I/O port registers
- SFR bus – allows faster access to a limited set of I/O port registers

SFR registers contain three registers for each I/O port, making a total of 27 registers for 9 I/O ports. The registers function in this manner:

- SFRPRTxDR – sets the output data state of the port (where x is port number and includes ports 0-6, 12, and 15)
- SFRPRTxSEL – selects each SFRPRTxDR register bit to set the output state of corresponding pin:
  - If the SFRPRTxSEL[y] bit is high, the SFRPRTxDR[y] bit sets the output state for the pin
  - If the SFRPRTxSEL[y] bit is low, PRTxDR[y] of port logic sets the output state of the pin (where y varies from 0 to 7)
- SFRPRTxPS – a read-only register that contains pin state values of the port pins

Figure 4-3 shows the connections between the 8051 and the I/O ports.

Figure 4-3. SFR–I/O Connections



4.6.5 Interrupt Enable (IE)

Bit 7 of IE (SFR 0xA8) enables or disables all 8051 interrupts.

4.6.5.1 Interrupt Enable 0xA8

	7	6	5	4	3	2	1	0
Access: POR	RW: 00							
Bit Name	EA							

4.7 Program and External Data Spaces

The 8051 has separate address spaces for program and data memory. The Internal, External, SFRs, and Program memory areas have their own address spaces. Data memory is divided onto 16 MB of external and 256 bytes of internal data memory, with an additional 128 bytes of SFR memory area.

4.7.1 Program Space

Program memory space begins at address 0x0000 and ends at address 0xFFFF. The 16-bit Program Counter register (PC) points to the next instruction to be read. Data can be read from the program space, but only through the MOVC instruction.

On reset, the PC is set to 0x0000. In the standard 8051, the instruction at address 0x0000 is usually a JMP, because interrupt vectors are hard-located at addresses 0x0003, 0x000B, 0x0013, 0x001B, and so on. Because PSoC 3 has a vectored interrupt controller, the 8051 can simply start executing code at address 0x0000.

4.7.2 External Data Space

The 8051 can address up to 16 MB external data memory (see the Memory Map chapter on page 119). This memory is accessed by MOVX instructions only.

4.8 CPU Halt Mechanisms

The CPU halts in the following circumstances:

- Boot Logic – asserted when the part comes out of reset; the CPU remains halted until the boot process completes.
- Miscellaneous Logic – writing a ‘1’ to the stop bit in register MLOGIC.CPU.SCR[0] asserts a halt request to the CPU. The CPU remains stopped until a DMA, reset, or the DoC sets this bit to ‘0’.
- Debug on-Chip (DoC) – when the debugger is enabled, a DoC halt request is asserted by writing to DBG_CTRL[1]. See the 8051 Debug on-Chip chapter on page 417.

5. PSoC 3 Cache Controller

The cache block is an instruction cache only. It is responsible for servicing 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.

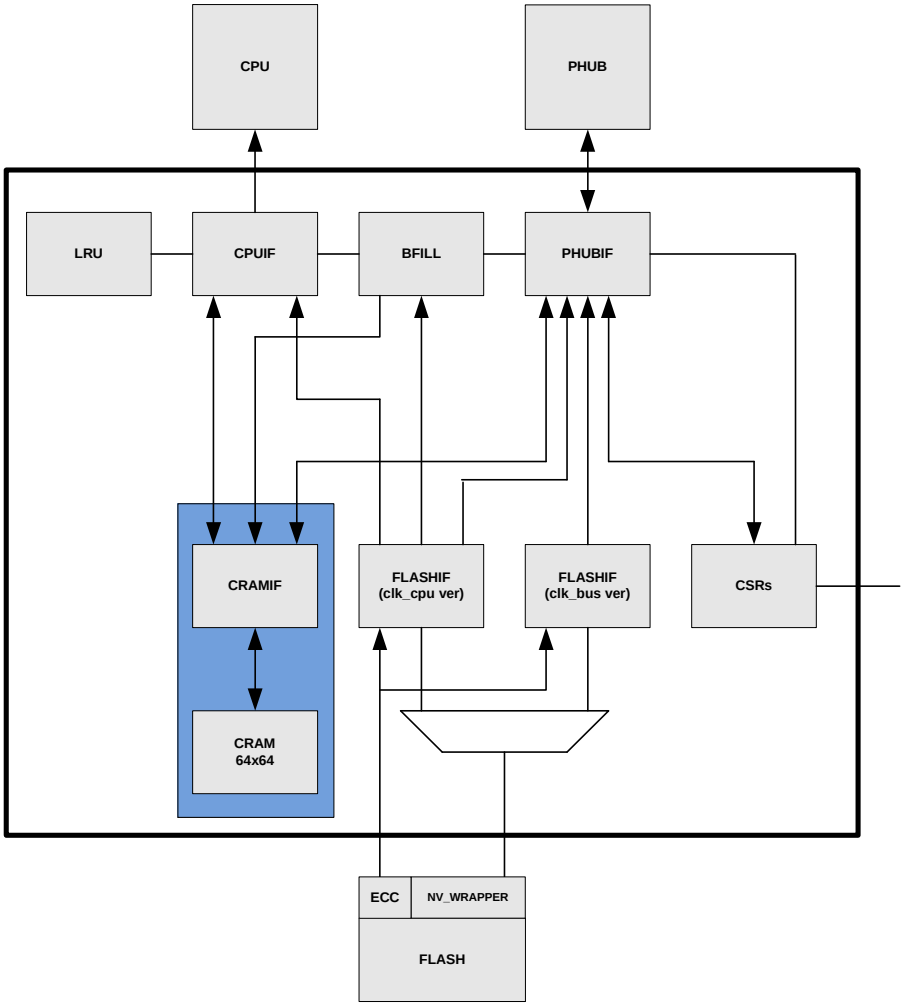
5.1 Features

- Single Port Cache RAM (CRAM) – either one read or one write at a time
- Instruction cache
- Fully associative
- 512 bytes total cache memory in PSoC 3
- Control to enable and disable cache
- Designed to put flash into sleep automatically to save power

5.2 Block Diagram

The PSoC 3 cache controller block diagram is in Figure 5-1.

Figure 5-1. Cache Controller Block Diagram



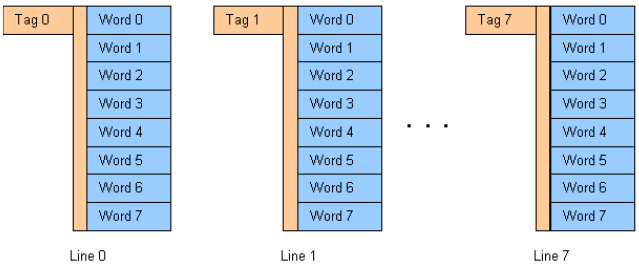
CRAMIF logic handles the communication between the cache and other blocks - CPU, background fill (BFILL), and PHUB requests.

5.3 Cache Memory Organization and Addressing

PSoC 3 has a total of 512 bytes of cache memory, which is divided into eight lines. A line is the cache channel where group of bytes move in or out. Each line has an eight-word capacity and each word contains eight bytes. This gives each cache line a 64-byte capacity.

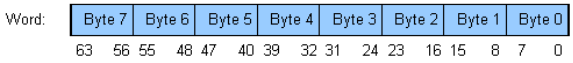
With this structure, a cache location is addressed by addressing the line, the word in the line, and the byte in the word.

Figure 5-2. Cache Lines



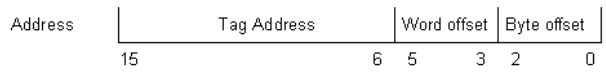
Bytes within a word are organized into a little endian format where byte 0 is least significant byte.

Figure 5-3. Word Byte Order



Each cache line has tag information. The address that is used to look up the cache is broken down into these fields:

Figure 5-4. Cache Byte Format



Instruction fetch access selects the particular line with matching tag value, word offset selects the particular word in the line and byte offset selects a byte in a word. Firmware can access these CRAM bytes and tag information.

Tag information is available using the CACHE_TAG[0..7] register. Cache RAM memory has the base address of 0x30000.

5.3.1 Cache Operation

You enable the cache by setting the CACHE_EN bit of the CACHE_CR register to 1.

The CPU sends out instruction fetch request to CPUIF; CPUIF, which interfaces to cache IF and flash IF, determines whether the instruction that is requested by CPU is already present in Cache (hit). If it is not (miss), then it accesses the instruction from either flash, 8-byte pre-fetch buffer of CPUIF, or 8-byte prefetch buffer of BFILL (Background fill).

On cache miss and if the cache is enabled, CPUIF writes the instruction fetched from the flash into CRAM, so that it is available the next time CPU requests the same location. Moreover, CPUIF has an 8-byte prefetch buffer that is used to save all the instructions and operands each time the CPU requests. Note that each time the flash is requested, it is always an 8-byte fetch. These eight bytes are loaded into the cache (if enabled) and the prefetch buffer of the CPUIF. The CPUIF prefetch buffer contains its own tag information. This forms an alternative to CRAM and allows the data in the prefetch buffer to be immediately returned to the CPU if there is address match rather than fetching from CRAM or flash, thus reducing power and improving performance.

If the cache is disabled, it is never updated by the hardware. However, CPUIF still performs cache look-ups and may produce hits. The instruction fetch request from the CPU is serviced from the cache as a regular hit. In this mode, firmware can update the cache and tag values. This is useful for the ISRs that require minimal latency, which can be loaded into the cache manually in firmware.

5.3.2 Cache Line Locking:

Each cache line can be locked so that it is not replaced at the next flash instruction request. A cache line can be locked by setting TAG_LOCK bit of corresponding tag register CACHE_TAG [0…7] to 1.

5.3.3 Cache Line Loading in Firmware

Firmware can be used to load instructions into the cache directly and set the tags to allow the CPU to fetch instructions from the cache instead of flash.

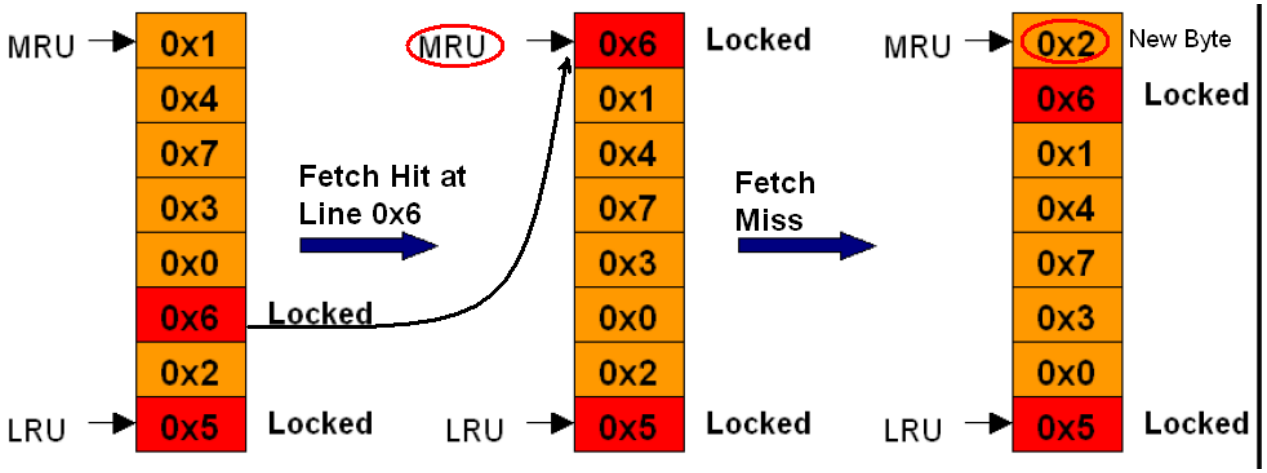
Follow the steps:

1. Clear CACHE_EN bit of CACHE_CR register to prevent instruction fetch from causing writes to CRAM during update.
2. Invalidate lines by clearing the appropriate tag register bits (TAG_VALID bits of CACHE_TAG register) to prevent instruction fetch from causing reads from the lines being loaded.
3. Write CRAM locations corresponding to the bytes invalidated in step 2.
4. Update appropriate tag register bits to mark as valid and lock modified lines. Setting lock bits will ensure that bytes do not get evicted from the cache.
5. Set CACHE_EN to reactivate writes to cache CRAM caused by the cache misses.

5.3.4 Cache Line Replacement Policy

An instruction fetch requested by the CPU may not exist in the cache. If the cache is full, then an existing valid cache line is evicted from the cache to create room for new line. The algorithm to select a line for eviction is the Least Recently Used (LRU) line that is not locked.

Figure 5-5. Replacement Policy



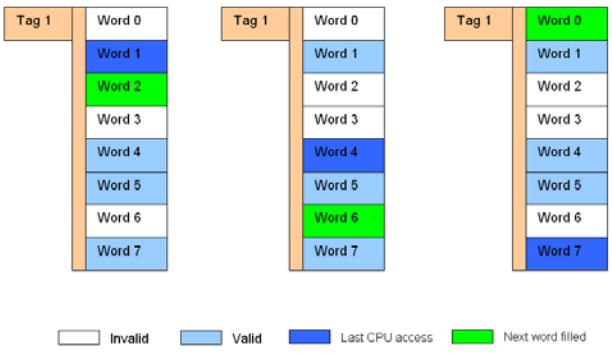
The LRU points to the cache line, which is the oldest of the used cache lines. The evicted cache line can be the line that is pointed by the LRU. However, if the line pointed by LRU is locked, then the unlocked line which is closer is evicted. The process of replacing the cache line can be understood by above figure.

An instruction fetch that produces a hit will mark that line as the Most Recently Used (MRU) regardless of that line is locked or not.

5.3.5 Background Fill (BFILL)

Background fill fetches the data from the flash and writes into CRAM. When the CPU is waiting for an instruction fetch and the cache controller encounter a miss on either a cache line or an invalid word in a valid cache line, the background fill state machine starts requesting the data from flash to fill any invalid words on the MRU line of the cache. When the BFILL reaches the end of the line, it wraps back to the beginning of the MRU line.

Figure 5-6. Background Fill



To enable BFILL, set the BFILL_EN bit of CACHE_CR register. Generally whenever cache is enabled, it is good idea to also turn on BFILL to improve cache performance. However, background fill is a speculative function and can create what may be wasteful reads from the flash. For highly power sensitive applications, it may be desirable to disable the BFILL function to save some power at the expense of performance.

5.3.6 ECC (Error Correction Code)

The ECC block checks the data read from the flash. It is responsible for error detection and correction. The cache gets the error status from the ECC block. The error status is logged into firmware visible registers. If the error is correctable, the ECC block corrects it and updates the CACHE_INT_LOG3 register. If the error is not correctable, it updates the details of the flash location where error occurred, into CACHE_INT_LOG4 register. This block can issue interrupt to the CPU, explained in the “interrupt” section.

ECC data in flash changes every time a write is done to the corresponding flash row. When a flash region is read, the corresponding ECC data will be used for error checking. The error checking is dynamic and happens every time the cache reads from the flash; this means, the comparison is for the latest data written to flash.

5.3.6.1 Interrupts

There are four interrupts related to the cache controller that are issued to the CPU. The interrupts can be enabled by setting appropriate bits of CACHE_INT_MSK register.

- ISR loading violation: If the cache is enabled and if firmware tries to write into Cache tag or CRAM, then this interrupt is issued. The write into the tag or CRAM will not be executed. A log is created (the associated cache line number where the violation happened), in the CACHE_INT_LOG0 register.

- Coherency violation: This interrupt is caused:
  - If the CACHE_EN bit of CACHE_CR register is set to ‘0’ and write to tag register (CACHE_TAG) except bits (23:16) is attempted
  - If the CACHE_EN bit is set to ‘0’ and a write to line in CRAM is attempted whose corresponding TAG_VALID is set

  In both cases, the write is allowed to access the. CACHE_INT_LOG1 register and see in which cache line the error occurred.
- Duplicate tag violation: This interrupt occurs when CACHE_EN bit is set to 0 and if firmware tries to create two tags with the same tag address. CACHE_INT_LOG2 register can be accessed to see which two lines have got the identical tags and the tag address.
- ECC - single bit: This interrupt occurs when a single bit error is encountered during a fill operation and is fixed. CACHE_INT_LOG3 register is updated with the flash location address where error occurred.
- ECC - multiple bits: This interrupt occurs when multiple bit error is encountered during a fill operation. This error cannot be fixed. CACHE_INT_LOG4 register is updated with the flash location address where error occurred.

5.3.7 Flash Low-Power Mode

The cache controller has the ability to send the flash into a low-power mode while continuing to run normally. When the number of cache hits reaches the programmed threshold, cache puts the flash into sleep thus saving power. Threshold value can be set in the CACHE_LP_MODE register. The flash is in low-power mode until the next cache miss occurs.

6. PHUB and DMAC

PSoC® 3 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 3 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 3 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 DMAC.

6.1.1 Features

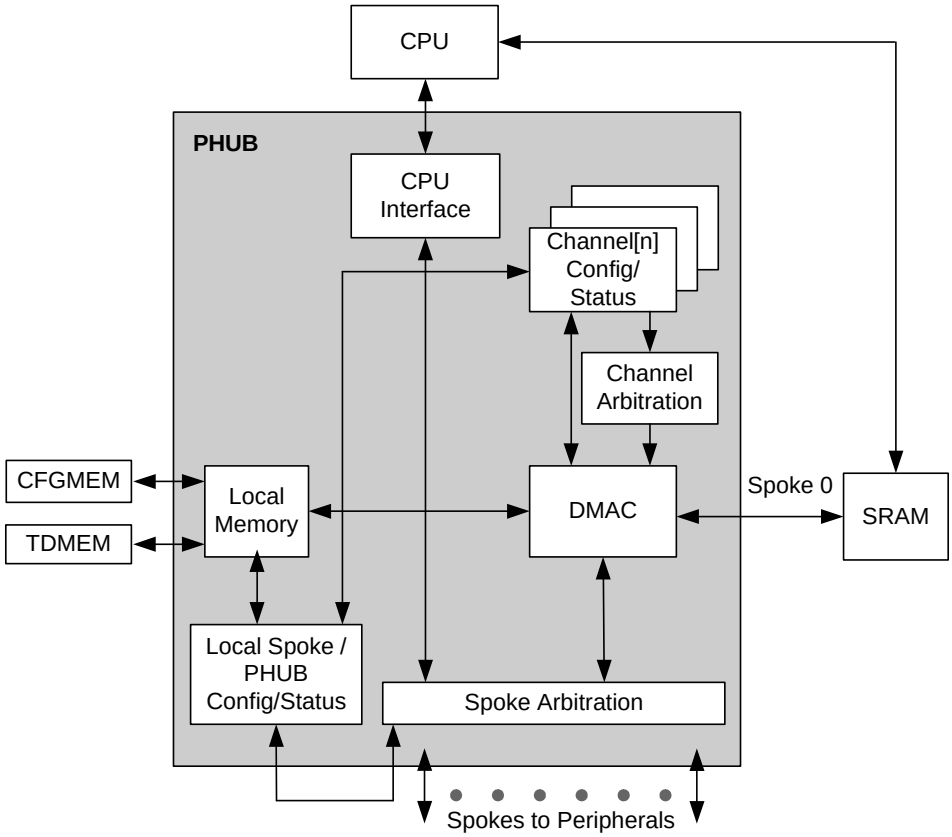
The PHUB has the following features:

- 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 74 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 3 device.

Table 6-1. Spoke Configuration

Spoke	Address Width (in bits)	Data Width (in bits)	Peripheral Names
0	14	32	SRAM
1	9	16	I/O interface, port interrupt control unit (PICU), external memory interface (EMIF)
2	19	32	PHUB local spoke, power management, clock, serial wire viewer (SWV), EEPROM
3	11	16	Delta-sigma ADC, analog interface
4	10	16	USB, CAN, 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.

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 exception is the SRAM, which has direct access by the CPU and PHUB. In this case, there is no arbitration required for SRAM. This helps to reduce the SRAM latency access.

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
- Large transactions can be broken into smaller bursts of 1 to 127 bytes with Intraspoke burst count restricted to ≤16.
- Each channel can be configured to generate an interrupt at the end of transfer
- Supports byte swapping, for conversion between big-endian and little-endian formats
- Handles data-width differences

6.2.1 Local Memory

As shown in Figure 6-1 on page 74, 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 CGFMEM is used to store the DMA channel configuration data. There are two registers: CFGMEMn.CFG0 and CFGMEMn.CFG1 (where n can be from 0 to 23) for each channel. 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.

The local memory is accessed through the local spoke of the PHUB (see Table 6-1 on page 74).

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:

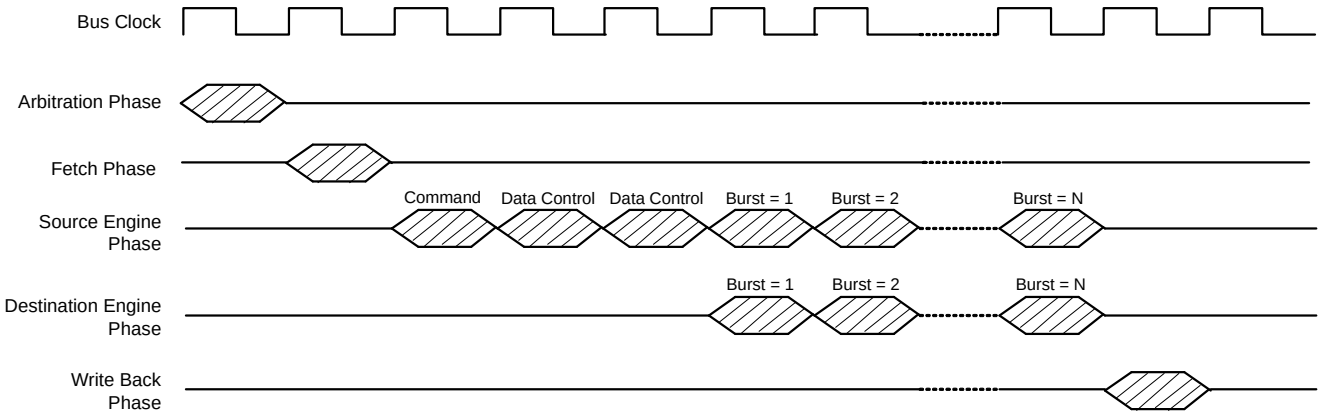
- Single requestor
- CPU doesn't interrupt the fetch phase
- Both source and destination spoke are readily available
- Source spoke and destination spoke are of same width
- Source and destination address start at even addressing
- Transfer count is a multiple of burst count
- Burst count matches the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

You want to move five samples of 16-bit ADC data to memory.

Notes

- 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 the *PSoC 3 Registers TRM*.

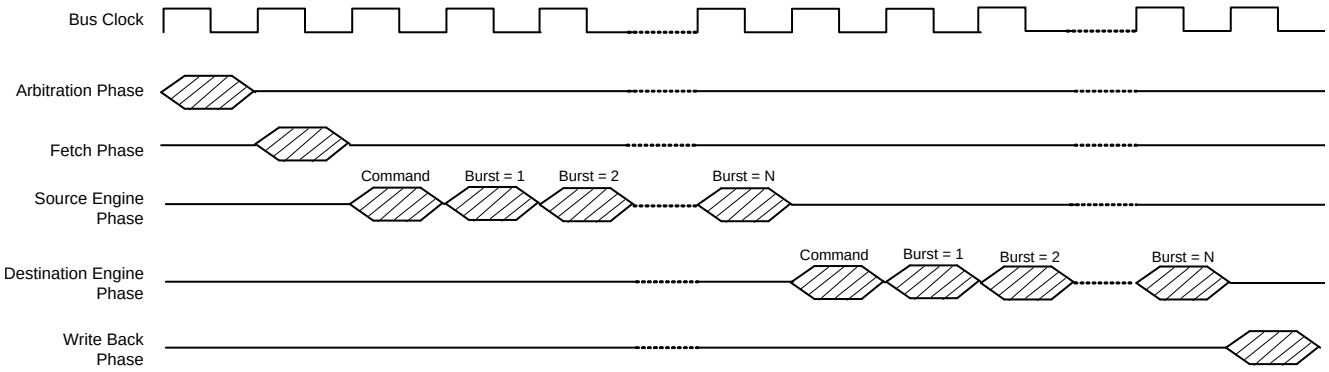
The source engine phase needs N + 3 cycles = 8 cycles.

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)

Intraspoke DMA transfer burst count should be limited to ≤ 16. 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

You want to move four 32-bit data words from one SRAM location to another SRAM location.

Notes

- 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

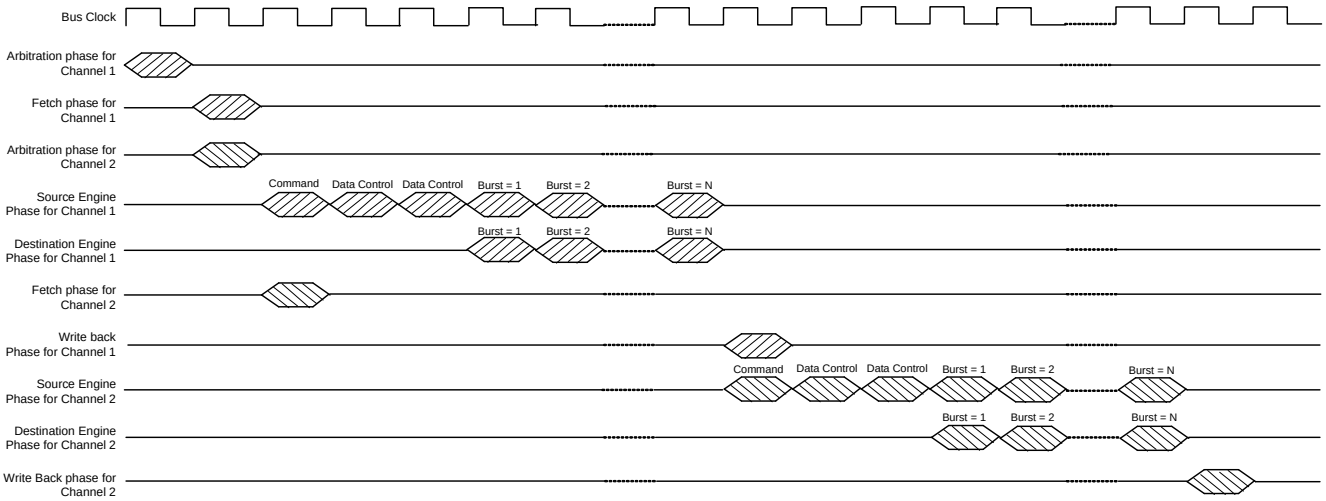
Total cycle time required for intraspoke transfer is 2N + 5 = (2 × 4 + 5) = 13 cycles

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 interspoke 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 different 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 like any normal priority algorithm where high priority channel can interrupt low priority channel
- Grant allocation Fairness algorithm: In this method, the 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%. Rest of the priorities share the bus based on the number of channels requested at that time. Because priority 0 has higher priority than 1, priority 0 can interrupt priority 1.

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

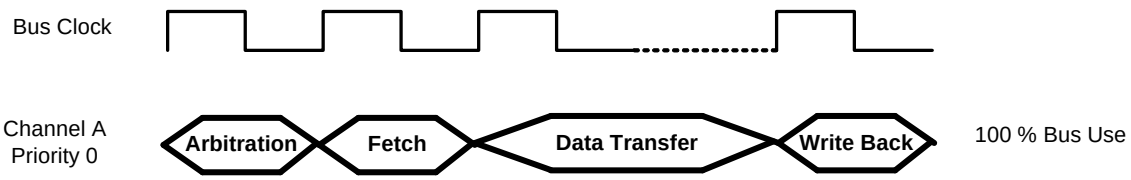
Under ideal conditions the Arbitration phase takes one cycle.

Examples using the Grant allocation Fairness Algorithm

Scenario 1

DMAC is free. Channel A with Priority 0 comes

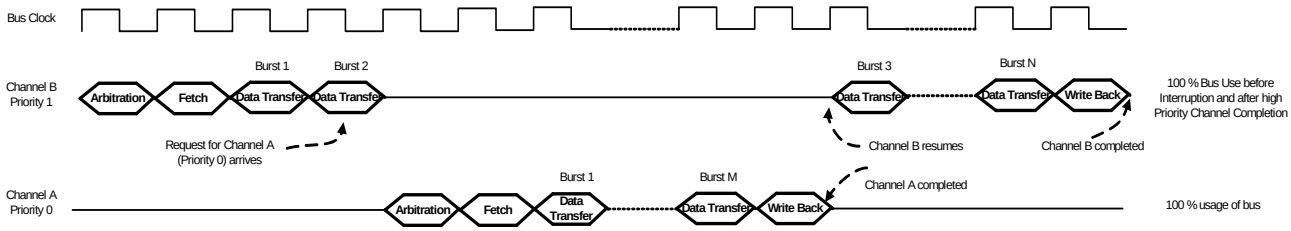
Figure 6-5. Priority 0 and Idle DMAC



Scenario 2

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

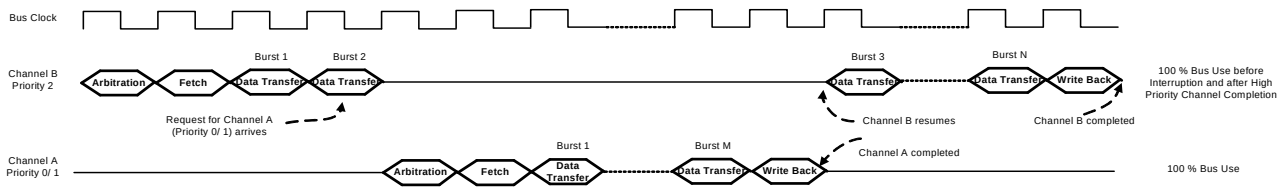
Figure 6-6. Priority 0 and Priority 1



Scenario 3

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

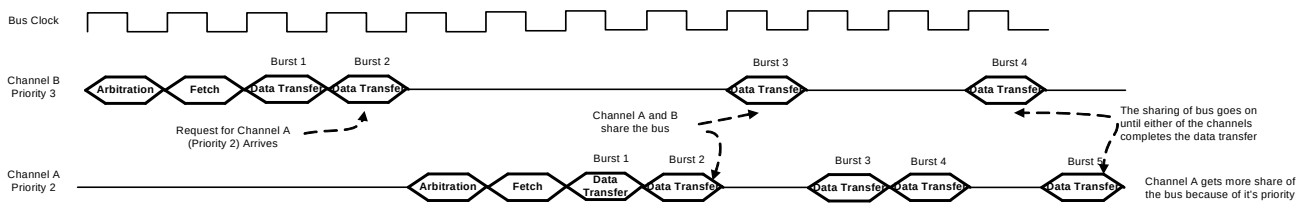
Figure 6-7. Priority 0/1 and Other Low Priority



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



The channels with priorities 2-7 are given access according to Table 6-2.

Table 6-2. Priority Levels and Bus Allocation

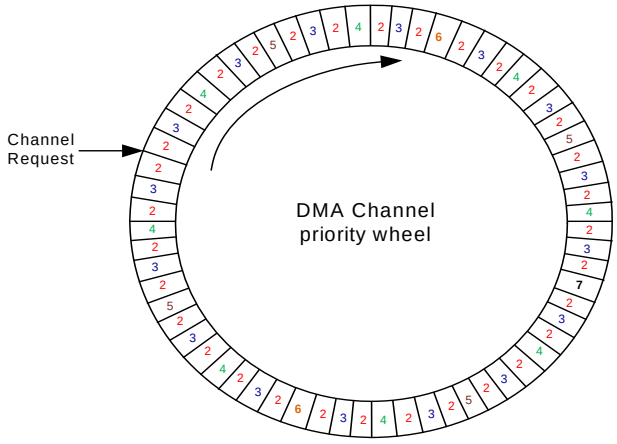
Priority Level	Bus Allocation Percentage
2	50
3	25
4	12.5
5	6.3
6	3.1
7	1.5

When DMA channels of varied priority request for DMAC at a time, 100 percent of bus bandwidth will be allocated for channels of priority 0 or 1.

Table 6-2 applies only if DMA channels with priorities 2 to 7 request simultaneously. Otherwise, the DMA channel with higher priority is given more access than Table 6-2 shows. Figure 6-9 shows a channel priority wheel that describes how the next 63 requests are handled if all channels with priorities 2 to 7 request simultaneously.

If a channel with priority 2 to 7 is NOT requesting, the slots of the missing channel priority are used by the channel with the highest priority. In that case, channels with higher priority get more access than Figure 6-9 shows.

Figure 6-9. DMA Channel Priority Wheel

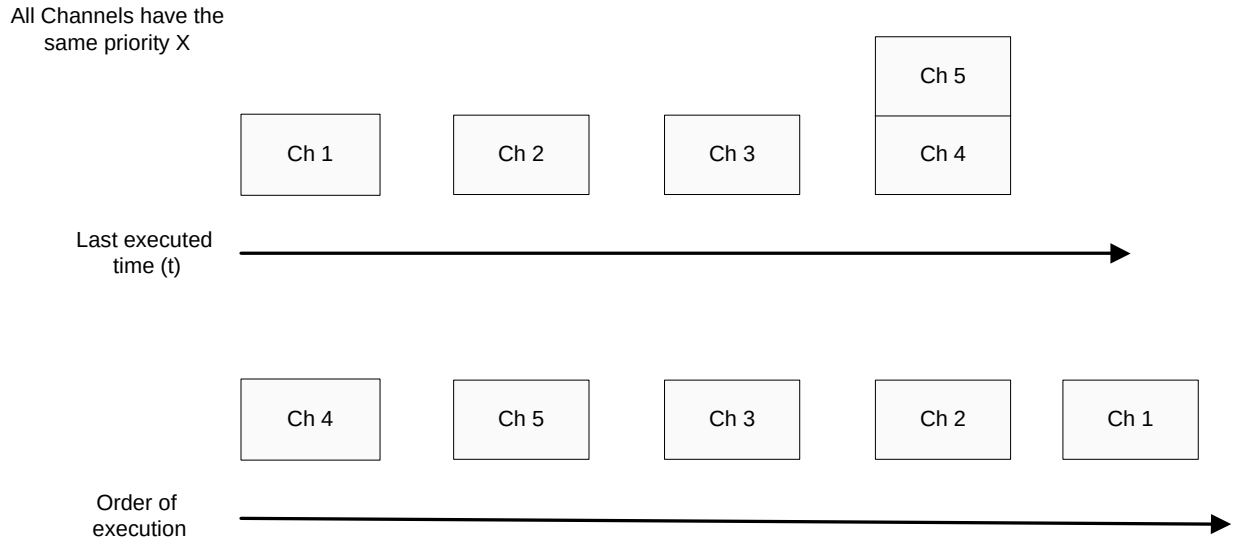


Because there are as many 24 DMA channels but only 8 priority levels, there can be multiple channels taking the same priority levels.

DMAC uses the Round Robin method to handle DMA Channels with same priority. In case of Round Robin algorithm, the DMA channel which 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 2 channels, then DMA Channel with lower number takes higher priority

Figure 6-10. Round Robin Scheduling



6.2.2.5 *DMA Latency in case of Nonideal Conditions*

The previous section explained the latency in case of ideal condition. But in real time, the ideal condition rarely exists. This section explains the latency calculation in case of nonideal conditions. The latency calculation in case of 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 by CPU also.

If there are multiple DMA channels sending request at the same time, the arbitration phase will take 2 cycles instead of the ideal 1 cycle

CPU Interrupts with Fetch Phase

The fetch phase ideally takes only 1 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 2 cycles for the access of configuration registers.

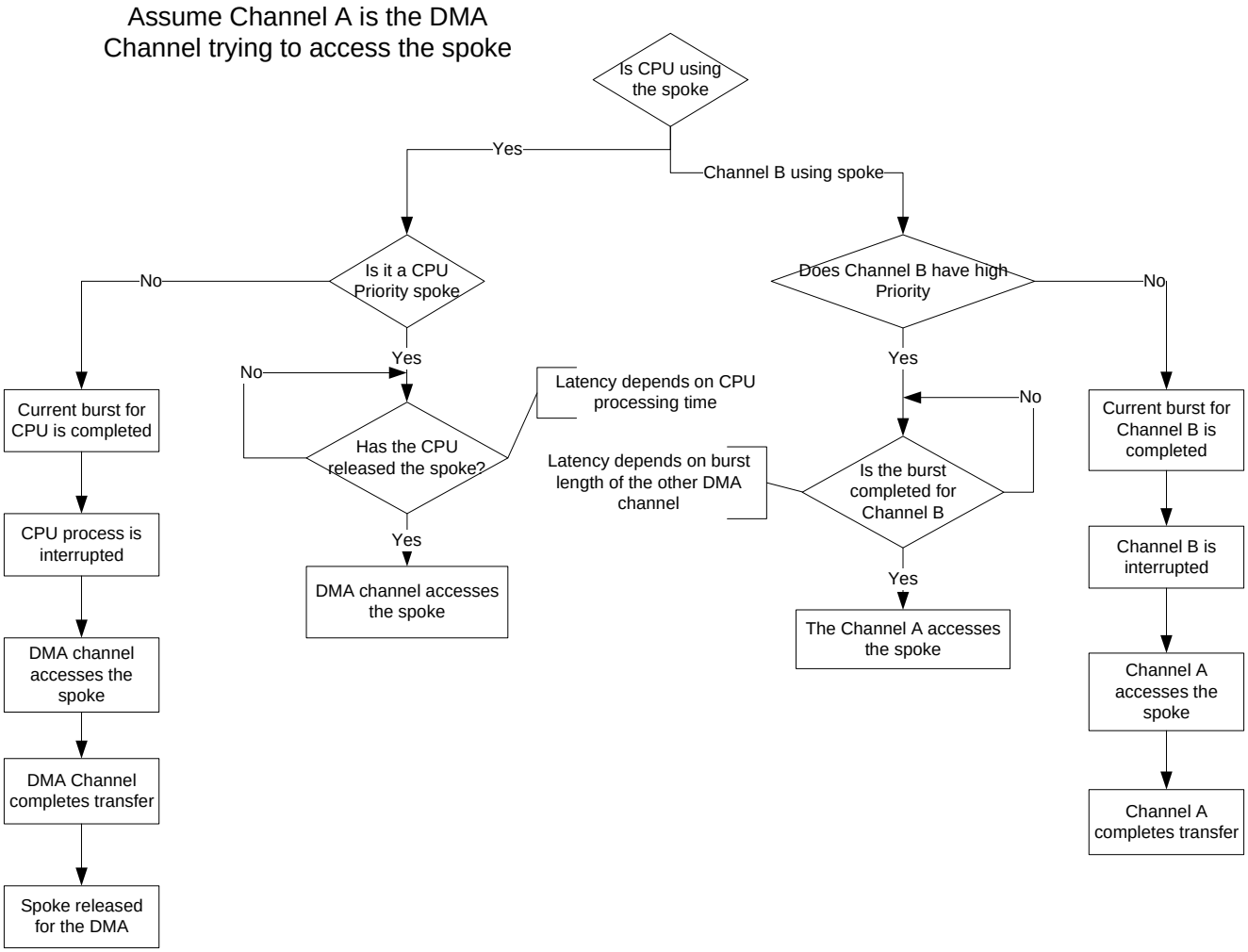
Also, there might be some high priority DMA channel in the Fetch phase. These scenarios will also add to the DMA Channel execution latency.

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 flow chart shows the arbitration mechanism.

Figure 6-11. DMA Channel Arbitration



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

Source and destination peripherals are not Ready

When the source or the destination peripheral is not ready to send or receive data, then the DMA channel has to wait till it is ready. In case of source peripheral not ready, the DMA channel will wait for the source peripheral to become ready

In case of destination peripheral not ready, the DMA channel will use the 16 byte FIFO of the PHUB. It reads the data from the source and fills it in the FIFO till the destination peripheral is ready. Thus the internal 16 byte FIFO is used during intra-spoke transfer and also during the conditions where the source and destination peripherals are no ready.

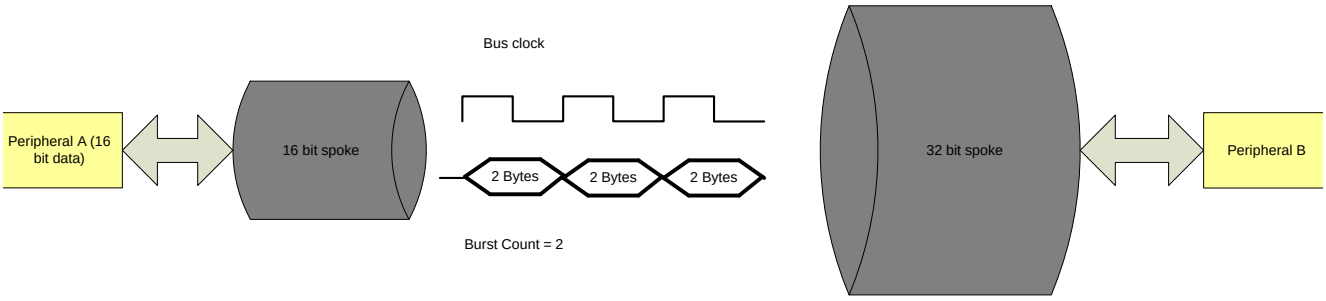
Source and destination spoke are of different width

The spoke widths play a very 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. Let's see 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)

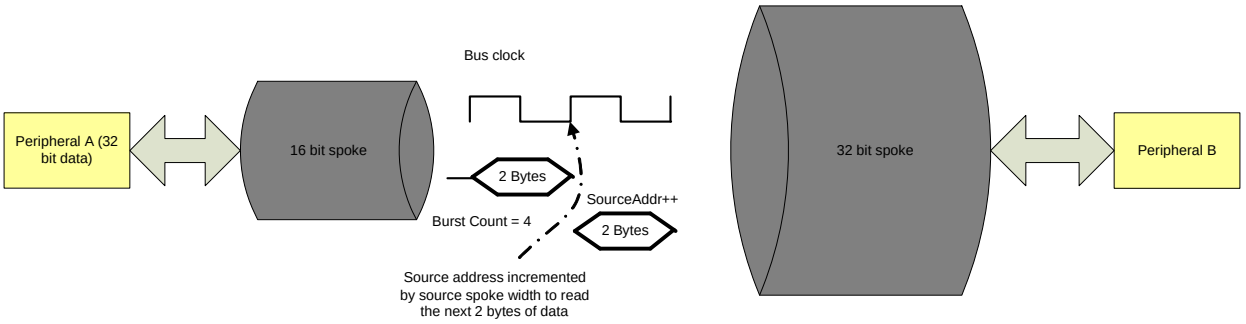
Figure 6-12. Data Transfer between 16-bit and 32-bit Spoke



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)

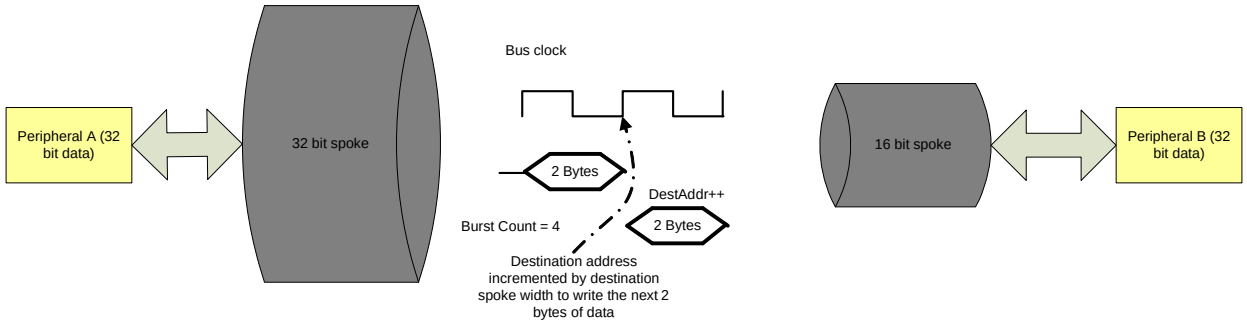
Figure 6-13. Data Transfer Between 16 bit and 32 bit Spoke



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

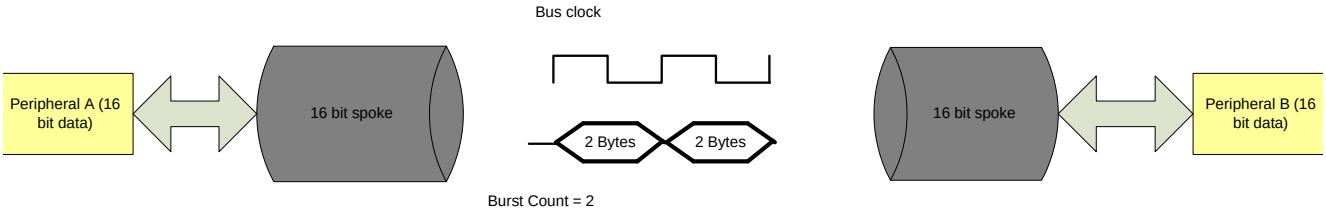
Figure 6-14. Data Transfer Between 16 bit and 32 bit Spoke



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

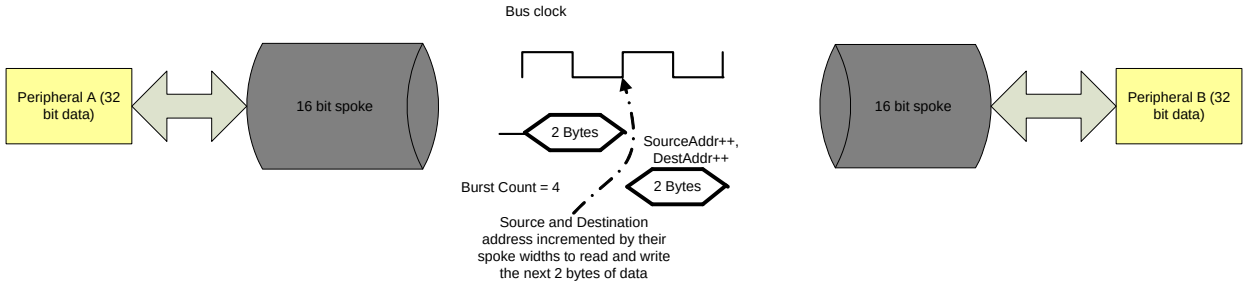
Figure 6-15. Data Transfer Between Two 16 bit Spoke



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

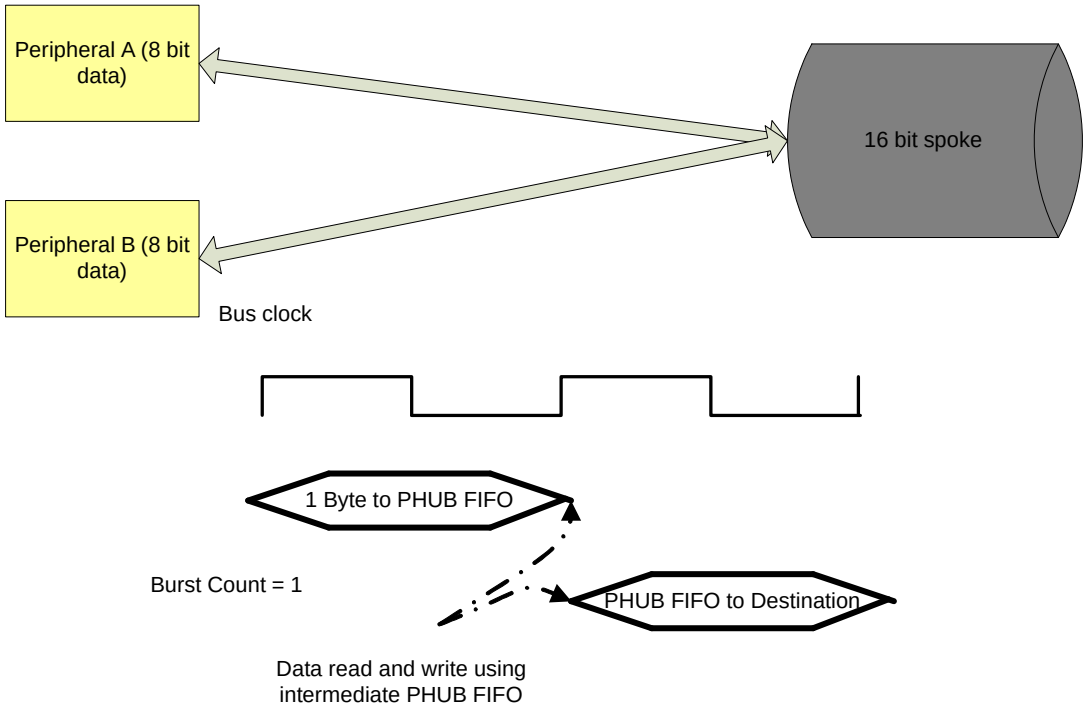
Figure 6-16. Data Transfer Between Two 16 bit Spoke



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

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

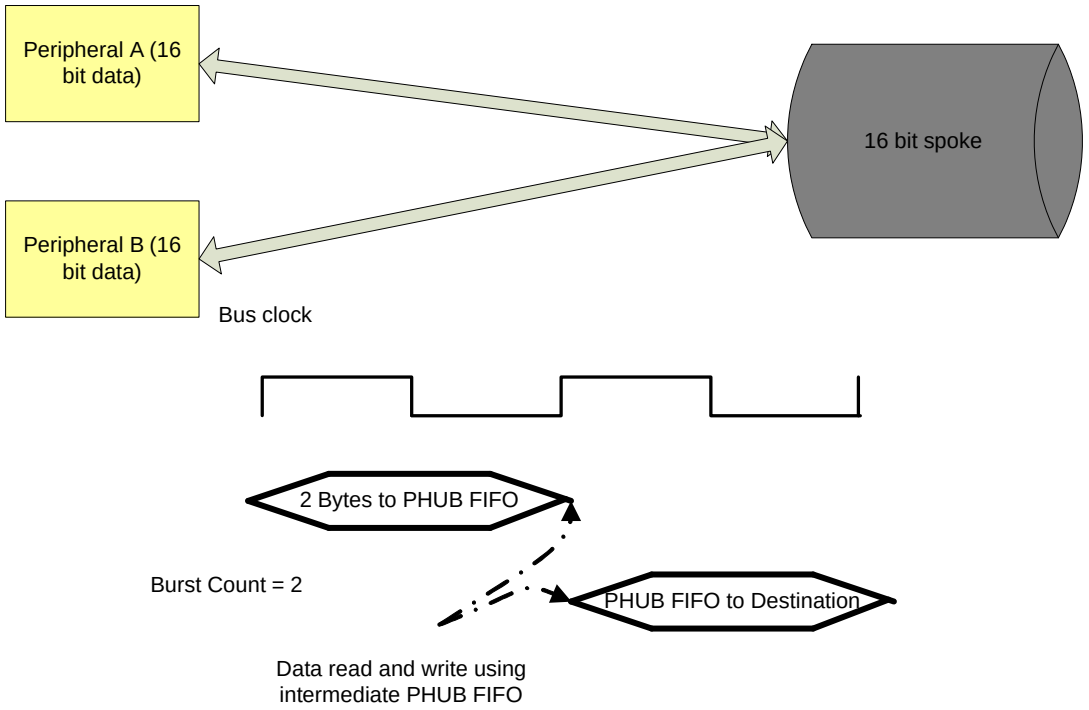
Figure 6-17. Intra Spoke Data Transfer



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

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

Figure 6-18. 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 6-19. Addressing in 32 bit Spoke

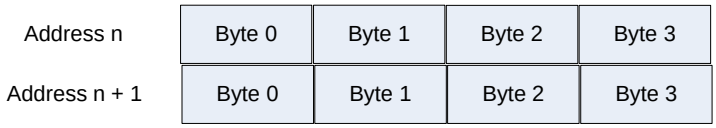
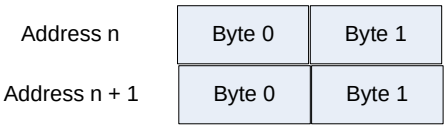
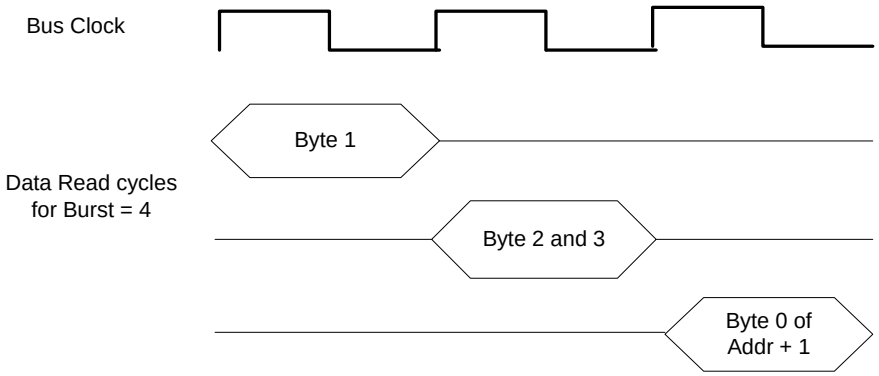


Figure 6-20. Addressing in 16 bit Spoke



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

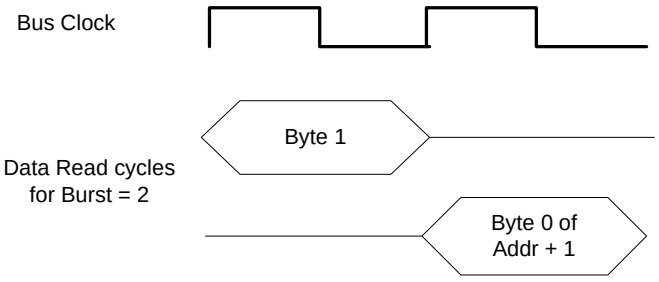
Figure 6-21. Odd Addressing in 32-Bit Spoke



As seen from the above figure, when the even addressing is not met, the bus cycle increases. In 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 6-22. 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 doesn't want to hog 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 till 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 very 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 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

6.3.1 Simple DMA

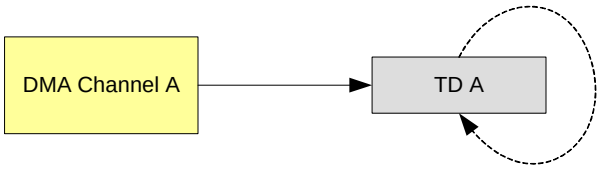
A single TD is used to transfer data between two peripherals or memory locations.

Figure 6-23. Simple DMA Transfer

6.3.2 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.

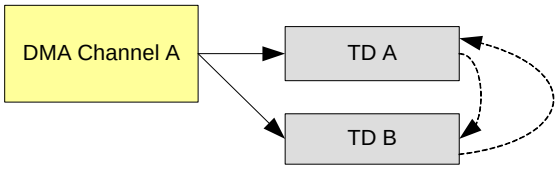
Figure 6-24. Auto Repeat DMA



6.3.3 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.

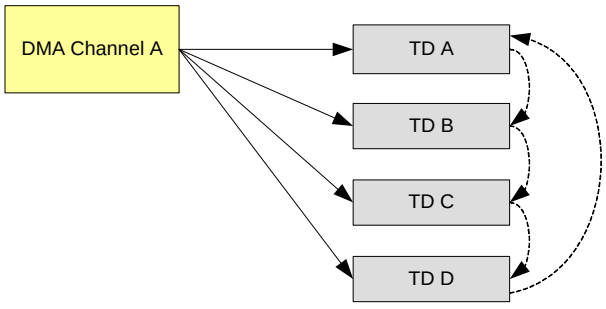
Figure 6-25. Ping Pong DMA



6.3.4 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 6-26. Circular DMA



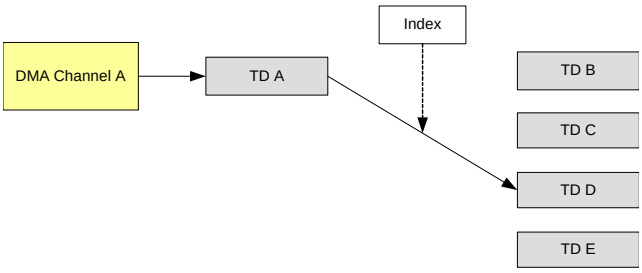
6.3.5 Indexed DMA

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

Example: If a peripheral was 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 6-27. Indexed DMA



6.3.6 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.

6.3.7 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 setup 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.

6.3.8 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.

6.4 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

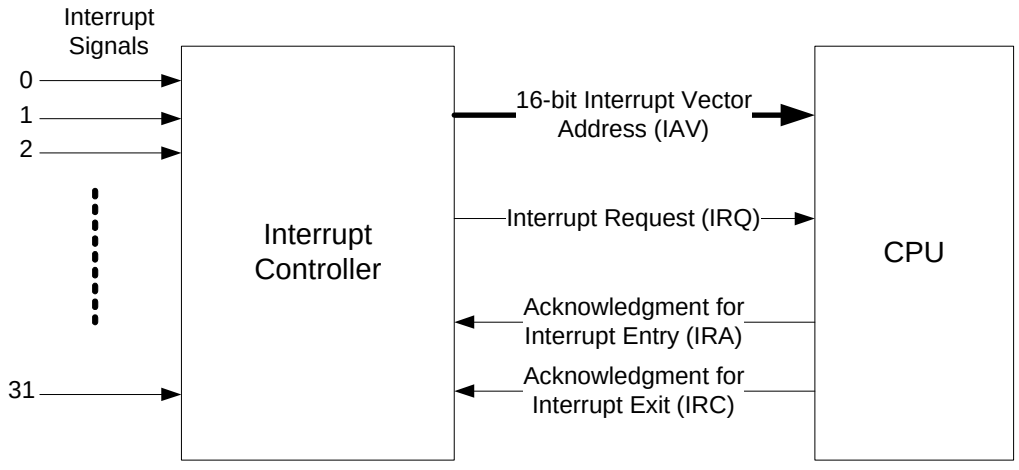
The following are features of the interrupt controller:

- 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

7.2 Block Diagram

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

Figure 7-1. Interrupt Controller Block Diagram



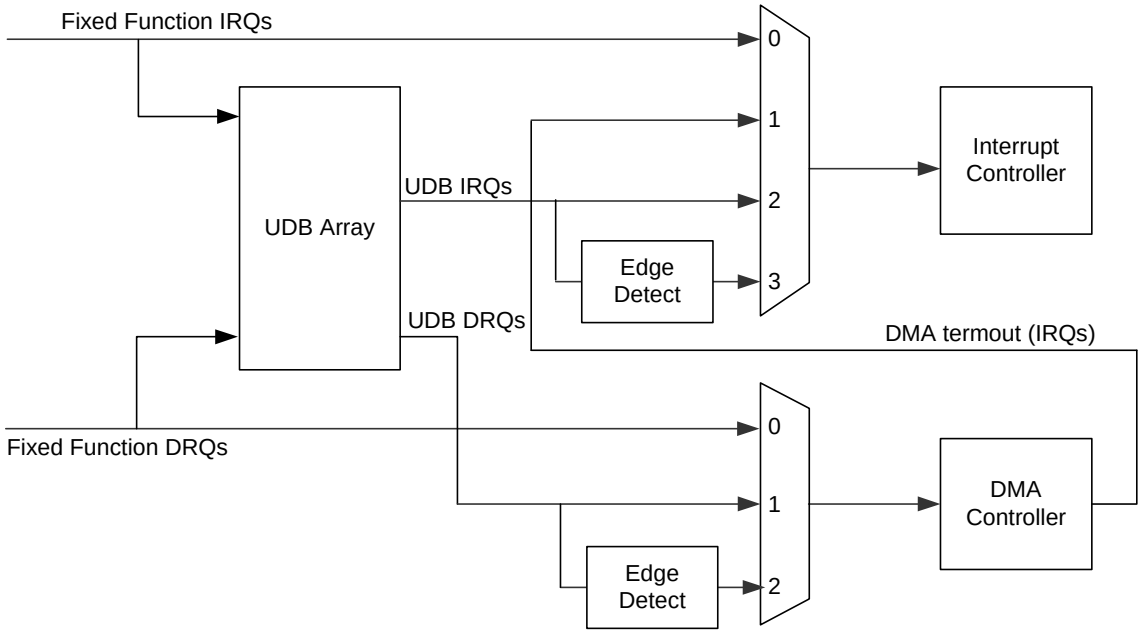
7.3 How It Works

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 signal routing is very flexible with PSoC 3architecture. 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.

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



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 7-1 shows the status of bits during read and write.

Table 7-1. 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 7-2. 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 or not 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 has to 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 has to 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 has to 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.

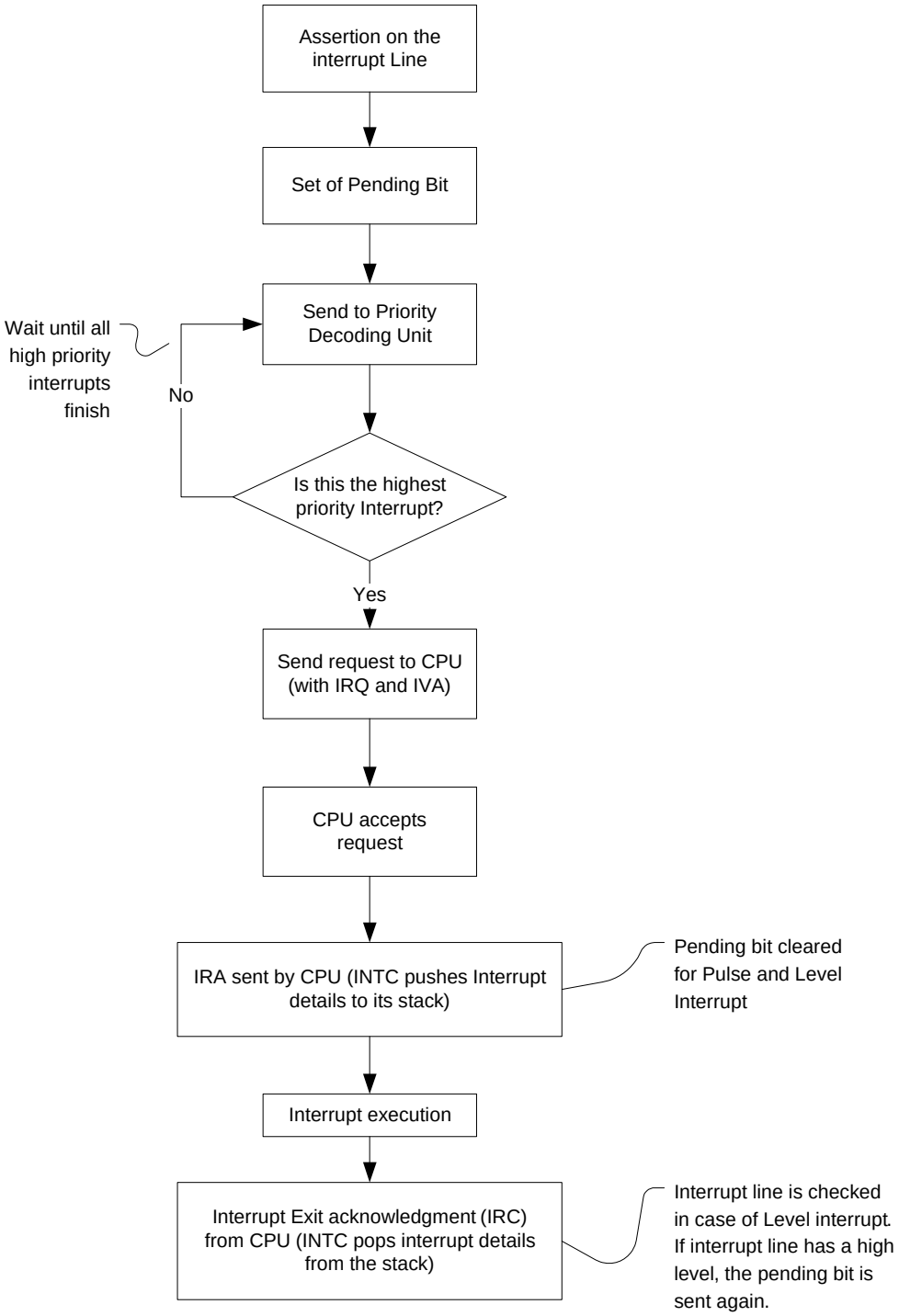
7.3.5 Interrupt Execution

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 3 Interrupt Controller Features

This section describes the following:

- Registers for the active interrupt
- Use of stacks during nesting of interrupts when a high priority interrupt is asserted during the execution of a low priority interrupt
- Registers that handle the vector addresses that correspond to every interrupt line

7.4.1 Active Interrupts

The active interrupt is the one being executed currently. The interrupt priority and interrupt number of the active interrupt are stored in eight level hardware stacks. The hardware stack is available with the Interrupt controller. There are two different stacks used to store the active interrupt details – one stores the interrupt priority, and the other stores the interrupt number. Following are the details of the stacks:

- STK – Stores the priority of the interrupt
- STK_INT_NUM – Stores the interrupt number

The top of the above stacks have the details of the current active interrupt and interrupt priority. The registers ACT_INT_NUM and ACT_VECT store the details of the latest interrupt number execution requested to the CPU and its corresponding vector address. The value in these registers is valid only between the “Interrupt request to the CPU (IRQ)” and “Interrupt Entry (IRA).” Any read outside this time frame may result in invalid values.

7.4.2 Interrupt Nesting

PSoC 3 devices support nesting of up to eight interrupts. Nesting of an interrupt occurs when a high priority interrupt is asserted during a low priority interrupt execution. In PSoC 3, the nesting of interrupts uses both the interrupt controller stack and the CPU stack.

- **Interrupt Controller Stack** – The interrupt controller stack is available with the interrupt controller and is used to store the interrupt number and interrupt priority. There are two stacks with a depth of eight levels. Following are the stack details:
  - STK – Stores the interrupt priority information.
  - STK_INT_NUM – Stores the interrupt number information.

Both of the stacks grow upwards. The push and pop in the interrupt controller stack is handled by the interrupt controller itself.

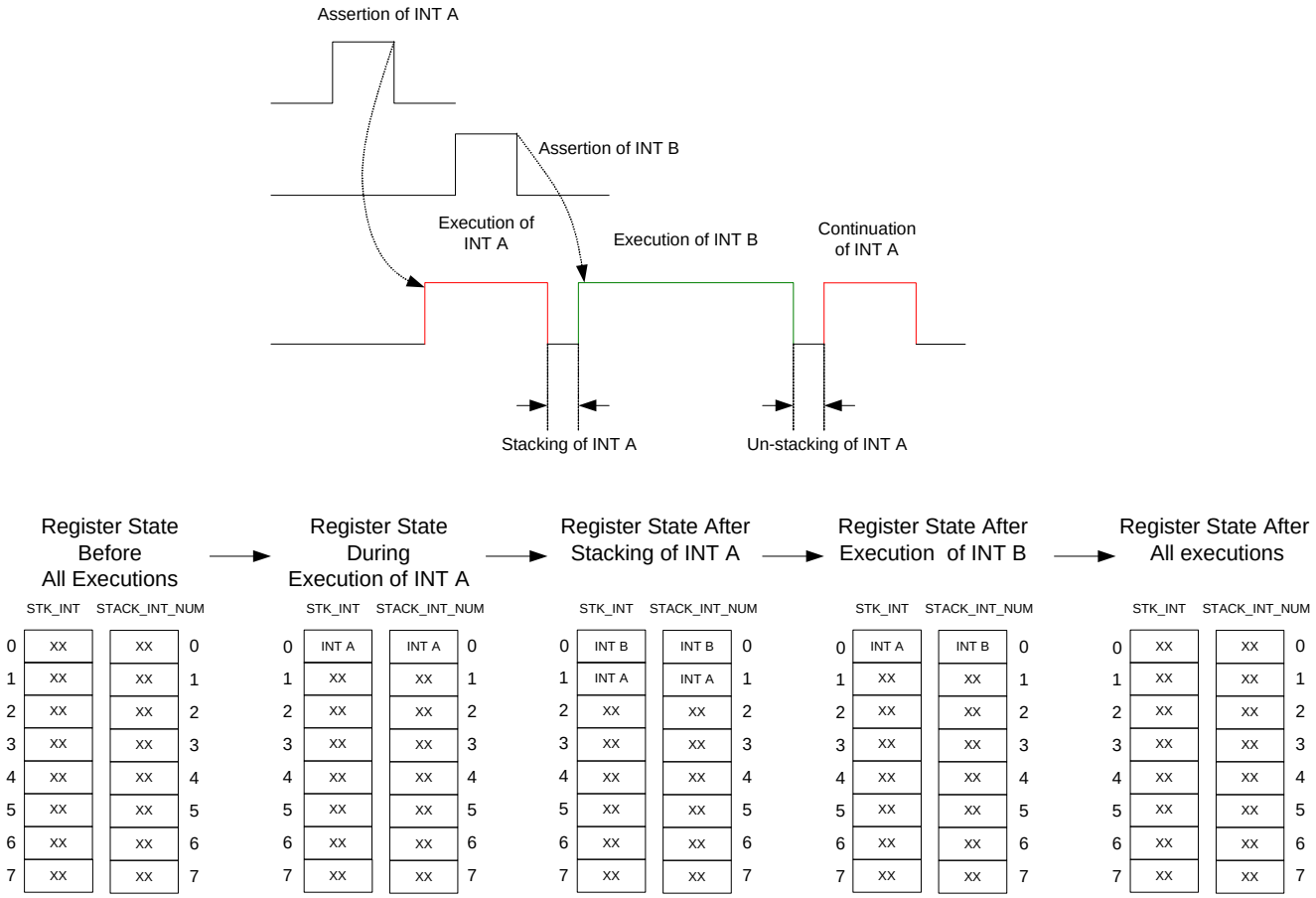
- **CPU Stack** – The CPU stack is used to store other general registers, such as the PC, GPR, SFR, PSW, ACC, and B, depending on the application. The CPU handles the automatic push/pop of the PC register (low byte pushed first). The rest of the required registers must be pushed/popped using the firmware inside the interrupt handler routine.

  The sequence of interrupt nesting is as follows:

1. When a high priority interrupt assertion occurs during the execution of the low priority interrupt, the low priority interrupt execution is stopped at that point.
2. The CPU accepts the request, stops the execution of the low priority interrupt, and pushes the PC.
3. The CPU sends the acknowledgment (for the high priority interrupt entry) to the interrupt controller.
4. The interrupt controller pushes the interrupt number and interrupt priority of the low priority interrupt to its stack. It pushes the higher priority interrupt context to the top of the stack.
5. The interrupt service routine can push the other registers, such as the PSW, GPR, SFR, ACC, and B, to the CPU stack. The high priority interrupt execution begins.
6. When the higher priority interrupt is executed, the CPU sends the IRC. The details of the high priority interrupt are popped from the interrupt controller stack leaving the low priority interrupt at the top of the stack. The details of the low priority interrupt are popped from the CPU stack. The low priority interrupt continues its execution from the point of suspension.
7. Because the push and pop of the stack are handled by the hardware, there is minimum latency, because no instruction is involved in the operation.

Figure 7-4 on page 95 shows the states of the stack during the nesting operation.

Figure 7-4. Register States During Nesting



In Figure 7-4, INT A is suspended, and the high priority interrupt INT B is executed. During nesting, INT A is pushed to registers. After INT B is executed, the registers are popped. When an interrupt begins to execute the interrupt, information is pushed to stack; when it finishes, the stack is popped.

7.4.3 Interrupt Vector Addresses

PSoC 3 devices have a feature that allows a user to specify the interrupt service routine starting address for every interrupt line. The address of the interrupt service routine is programmable. 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 address register, resulting in the direct call of the routine, preventing latency.

1. The interrupt service routine address is programmable and is stored in Vector Address registers called VECT[0…31].

   There are 32 vector address registers corresponding to the 32 interrupt lines.

   Each Vector Address register is 16 bits.

2. During the interrupt assertion, the address of the service routine is retrieved from these registers and given to the CPU for execution of the interrupt.

7.4.4 Sleep Mode Behavior

The Interrupt Controller works in all of the power modes (Active, Stand by, Sleep and Hibernate) unless the user switches the clocks off manually. All of the registers (status and configuration) except the pending register and interrupt controller stack, retain their values during Sleep mode. The Pending and Interrupt Controller Stack registers are set with the power on value at wakeup. Because the pending registers are nonretention registers, the requests that are pended will be missed when the device goes to sleep.

Do not change the power mode change inside the Interrupt Service routine. If a change in mode is requested, the device will finish the ISR, exit the ISR, and then switch the power mode.

The clock for the Interrupt Controller can be enabled and disabled by setting the register bit “CLOCK_EN” in the INTC_CLOCK_EN register. When the clock is switched off

for the Interrupt Controller, the CPU should not access the ISR (as the IRA and IRC cannot be processed by the Interrupt Controller).

7.5 Interrupt Controller and Power Modes

The CPU core (8051) 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).

Follow these steps before 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. An Error Correcting Code (ECC) can enable high reliability applications.

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. Additionally, selected configuration options, such as boot speed and pin drive mode, are stored in nonvolatile memory, allowing settings to become active immediately after power on reset (POR).

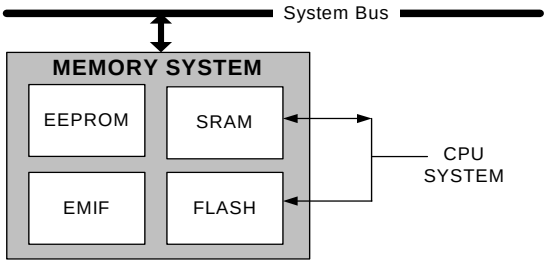
This section encompasses the following chapters:

- Nonvolatile Latch chapter on page 99
- SRAM chapter on page 103
- Flash Program Memory chapter on page 107
- EEPROM chapter on page 109
- EMIF chapter on page 111
- Memory Map chapter on page 119
- Cache chapter on page 147

Top Level Architecture

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

Memory Block Diagram



8. Nonvolatile Latch

A Nonvolatile Latch (NVL or NV latch) is an array of programmable, nonvolatile memory elements whose outputs are stable at low voltage. It is used to configure the device at Power on Reset. 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.

8.1 Features

NV latches include:

- A 4x8-bit NV latch for device configuration
- A 4x8-bit Write Once NV latch for device security

8.2 Device Configuration NV Latch

Device configuration NV latches allow configuration of PSoC® device parts before the CPU reset is released. For example, the user may configure each I/O port to be in one of four drive modes before CPU reset is released. Device configuration NV latch values have lower endurance and must be written in a narrower temperature window. Programming temperature range and endurance are traded off to meet the low voltage and wide temperature requirements. For endurance, retention, and temperature specs for NV latches see the specific device datasheet. The Device Configuration NV Latch register map is shown in Table 8-1.

Table 8-1. Device Configuration Register Map

Register Address	7	6	5	4	3	2	1	0
0x00	PRT3RDM[1:0]		PRT2RDM[1:0]		PRT1RDM[1:0]		PRT0RDM[1:0]	
0x01	PRT12RDM[1:0]		PRT6RDM[1:0]		PRT5RDM[1:0]		PRT4RDM[1:0]	
0x02	XRESMEN	DEBUG_EN	Reserved				PRT15RDM[1:0]	
0x03	DIG_PHS_DLY[3:0]				ECCEN	DPS[1:0]		CFGSPEED

8.2.1 PRTxRDM[1:0]

Port Reset Drive mode NVL bits enable selection of one of four drive modes to be in effect between the release of POR and the configuration of the device by user firmware. These four drive modes are a subset of the drive modes available by writing to the port drive mode registers. See the I/O System chapter on page 159 for more details. The following is a summary of the four NVL drive mode settings:

- 00b – High impedance analog
- 01b – High impedance digital
- 10b – Resistive pull up
- 11b – Resistive pull down

8.2.2 XRESMEN

GPIO pin (P1[2]) may be configured as an external reset (XRES_N) pin. The configuration of that pin is controlled with this NVL bit:

- 0 – GPIO
- 1 – XRES_N

8.2.3 DEBUG_EN

The Debug Enable bit allows access to the on-chip debugger and allows programming, either in JTAG or SWD mode, without having to acquire the device in test mode. JTAG or SWD can be selected by the Debug Port Select (DPS) bits. When DEBUG_EN is not set, it is required to enter test mode to gain debugger access and enable device programming.

- 0 – Debug Disabled (no debugger access except after test acquire)
- 1 – Debug Enabled (debugger access with or without test acquire)

8.2.4 CFGSPEED

The Configuration Speed NVL bit determines if the IMO defaults to a fast or slow speed. See the Clocking System chapter on page 123 for more details. This configuration is intended to balance the need for rapid boot and configuration against peak power consumption.

- 0 – Slow (12 MHz IMO frequency)
- 1 – Fast (48 MHz IMO frequency)

8.2.5 DPS[1:0]

Debug Port Select NVL bits allow the user to select a debugging port interface that is active after POR is released. If the debug port's disabled setting is used, the acquire functions of the test controller must be used to activate the debug port. See the Test Controller chapter on page 405 for more details. These NVL bits do not enable the debugger logic; they enable only the physical interface. The only way to enable the debug logic is for the user's firmware or configuration to write the debugger enable bit.

- 00b – 5-wire JTAG
- 01b – 4-wire JTAG
- 10b – SWD (single wire debug)
- 11b – Debug ports disabled

8.2.6 ECCEN

For devices that support an Error Correcting Code (ECC) in the flash, this NVL bit is used to set whether ECC is enabled. See the Flash Program Memory chapter on page 107 for more details.

- 0 – ECC disabled
- 1 – ECC enabled

8.2.7 DIG_PHS_DLY[3:0]

This bit selects the digital clock phase delay in 1 ns increments. See the Clocking System chapter on page 123 for more details,

- 0x00 – Clock disabled
- 0x01 – 2.5 ns delay
- 0x02 – 3.5 ns delay
- …
- 0X0A – 11.5 ns delay
- 0x0B – 12.5 ns delay
- 0x0C – Clock disabled
- 0X0D – Clock disabled
- 0X0E – Clock disabled
- 0X0F – Clock disabled

8.3 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 once 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 can write the key into the WOL, program the flash protection data, and then reset the part to lock it. See the Flash, Configuration Protection chapter on page 175 for details on flash protection.

8.4 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 427 for more details on 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.

8.5 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.

9. SRAM

PSoC® 3 devices include on-chip SRAM.

9.1 Features

PSoC 3 SRAM has these features:

- Organized as up to three blocks of 4 KB each, including the 4 KB trace buffer, for CY8C38 family.
- 8-, 16-, or 32-bit accesses. In PSoC 3 devices the CPU has 8-bit direct access to SRAM.
- 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.

9.2 Block Diagram

Figure 9-1 shows the PSoC 3 family SRAM accesses.

Figure 9-1. PSoC 3 Family SRAM Accesses

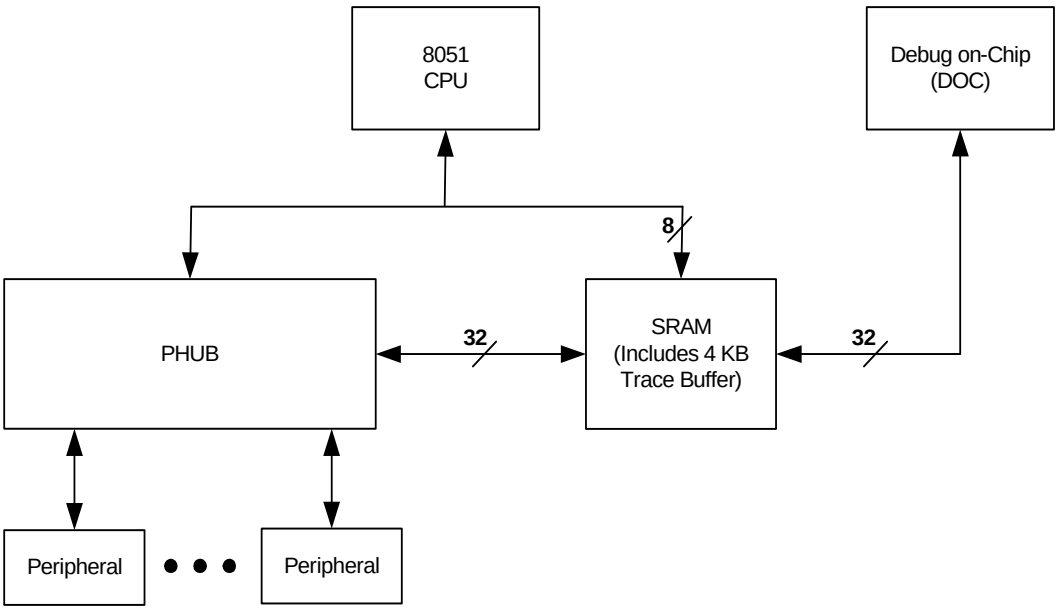
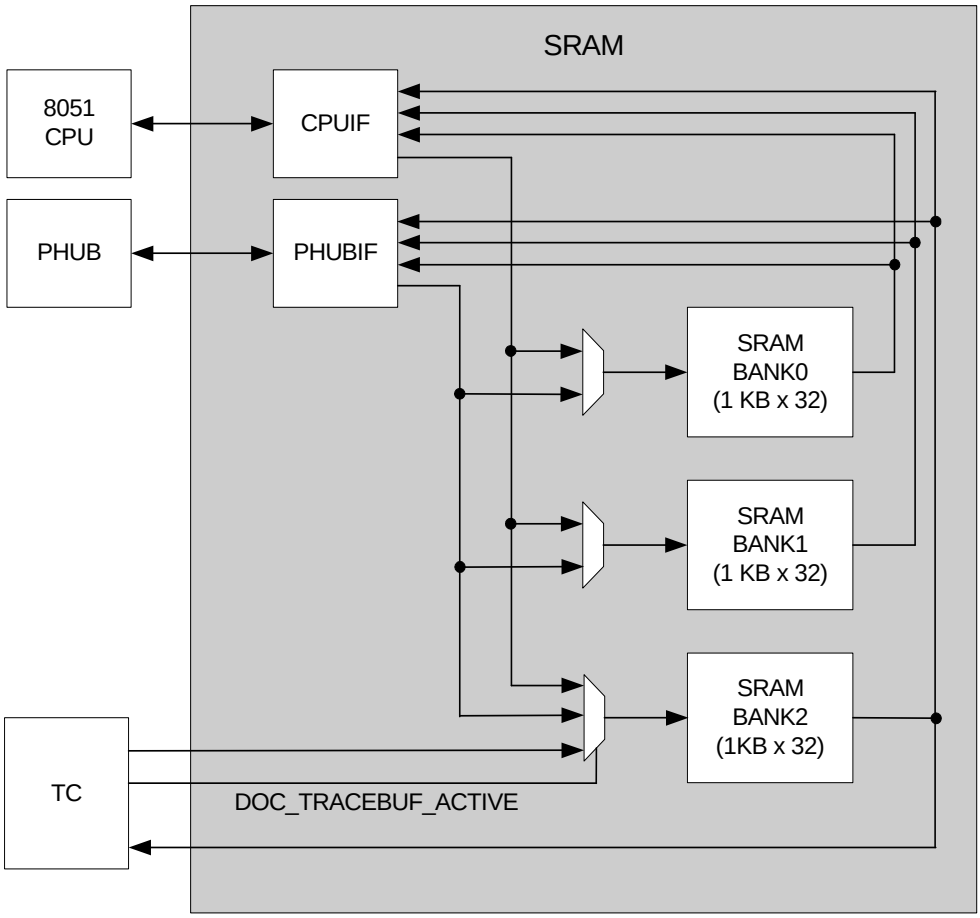


Figure 9-2 shows internal SRAM organization for the CY8C38 family.

Figure 9-2. CY8C38 Family SRAM Organization



9.3 How It Works

The CY8C38 family has up to 12 KB SRAM implemented as three 4 KB blocks. All 12 KB are accessible by the 8051 CPU and by the PHUB DMA controller in normal operation. During debug, the Debug on-Chip (DoC) accesses the upper 4 KB of SYSMEM as a trace buffer. If the trace buffer is not used for tracing, it may be used as additional SRAM for normal operation.

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

All data paths to SRAM are 32 bits wide except the data path from the 8051 CPU, which is 8 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:

- CY8C38 family – The 8051 has priority over the PHUB for the lower and upper 4 KB (SRAM BANK0 and BANK2), and the PHUB has priority over the CPU for the middle 4 KB (SRAM BANK1). When DoC tracing is active, the DoC has exclusive access to the upper 4 KB (SRAM BANK2) – neither the CPU nor the PHUB can access this bank while tracing is active.

The SRAM responds to CPU, PHUB, and 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.

10. Flash Program Memory

PSoC® 3 include on-chip flash memory. Additional flash is available for either error correction bytes or data storage.

10.1 Features

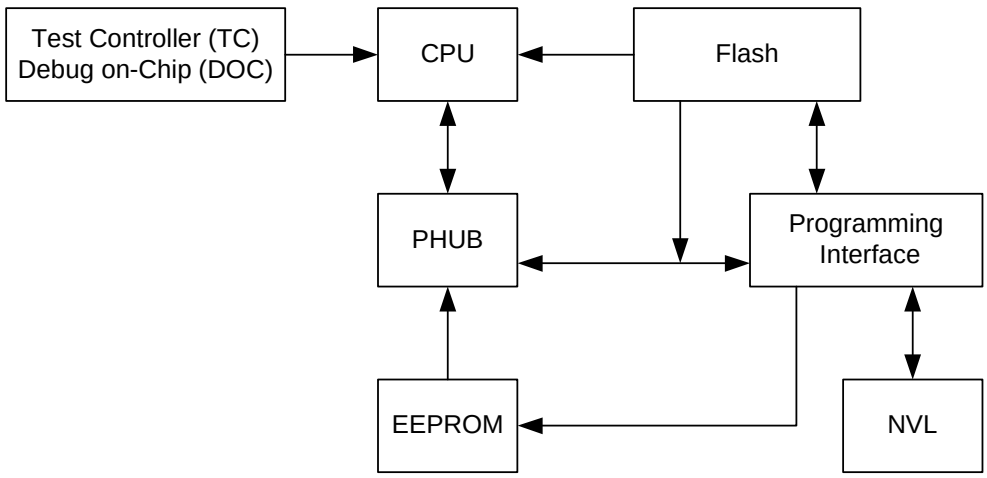
PSoC 3 flash memory has the following features:

- Organized in rows, where each row contains 256 data bytes plus 32 bytes for either error correcting codes (ECC) or data storage.
- For PSoC 3 architecture, organized as one block of 64, 128, or 256 rows.
- Stores CPU program and bulk or nonvolatile data
- PSoC 3 architecture has only 8-bit direct access.
- Programmable with a simple command / status register interface (see Nonvolatile Memory Programming chapter on page 427).
- Four levels of protection (see Nonvolatile Memory Programming chapter on page 427 and Flash, Configuration Protection chapter on page 175).

10.2 Block Diagram

Figure 10-1 is a block diagram of the flash programming system.

Figure 10-1. Flash Block Diagram



10.3 How It Works

Flash memory provides nonvolatile storage for firmware, device configuration data, bulk data storage, ECC data, 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 ECC data. Factory configuration and user-defined protection data are stored in the extended region, also known as the hidden rows of flash.

The main region of flash in the CY8C38 family is 72 KB, consisting of 64 KB of user space and 8 KB of ECC. The extended region is four rows of 256 bytes each.

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 427 and Flash, Configuration Protection chapter on page 175.

Flash can be read by both the CPU and the DMA controller.

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 427.

Note It can take as much as 20 milliseconds to write to EEPROM or flash. During this time the device should not be reset; otherwise, unexpected changes may be made to portions of EEPROM or flash. The reset sources (see Reset Sources on page 151) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. Also, configure the low-voltage detect circuits to generate an interrupt instead of a reset.

10.4 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 3 has only one flash array, 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.

10.5 ECC Error Detection and Interrupts

The ECC detects conditions that may interfere with software operation. The information is logged into individual interrupt registers that become latched until the software clears the corresponding valid bit. All interrupt sources within the ECC are passed through a mask condition; then, they are reduced into a single interrupt request to the Interrupt Controller unit.

When the software is notified about an existing interrupt in the ECC, the following sequence occurs:

1. The software reads the Interrupt Status register CACHE_INT_SR that provides the valid bits of all interrupts in a single read operation.
2. The software examines individual interrupt registers for more log information (CACHE_INT_LOG[0..5]).
3. Stored log information is cleared on read of registers.
4. After clearing of log information, the status register (CACHE_INT_SR) is automatically cleared, because it is a collection of valid bits of the log registers.

Logging is always enabled; reporting may be disabled through the Interrupt Mask Register (CACHE_INT_MSK).

The following conditions are detected by the hardware and logged as potential interrupt sources:

- **ECC – Single Bit** – A single bit error was encountered during a fill operation and was fixed.
- **ECC – Multi Bit** – A multi-bit error was encountered during a fill operation, but it cannot be corrected.
- **Attempted Flash Write** – If a write to flash through the PHUB is attempted.

11. EEPROM

PSoC® 3 devices have on-chip EEPROM memory. This family offers devices that range from 512 bytes to 2 kilobytes.

11.1 Features

PSoC 3 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
- Write and erase using SPC commands
- 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 427)

11.2 Block Diagram

There is no block diagram associated with EEPROM.

11.3 How It Works

EEPROM memory provides nonvolatile storage for user data. EEPROM write and erase operation is done using SPC commands. It may be read by both the CPU and the DMA controller, using the PHUB. All read accesses are 8-bit.

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. EEPROM can be taken in and out of sleep mode by setting the bit EE_SLEEP_REQ in SPC_FM_EE_CR[4], as shown in Table 11-1. 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.

It is also possible to check the current sleep status of the EEPROM by reading the bit EE_AWAKE in SPC_FM_EE_CR[5], as shown in Table 11-2.

Table 11-1. Bit Settings for EE_SLEEP_REQ in SPC_FM_EE_CR[4]

Setting	Description
0 (default)	Wake up EEPROM
1	Put EEPROM to sleep

Table 11-2. Bit Settings for EE_AWAKE in SPC_FM_EE_CR[5]

Setting	Description
0	EEPROM is asleep
1 (default)	EEPROM is awake

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. For more information see Nonvolatile Memory Programming chapter on page 427.

Note It can take as much as 20 milliseconds to write to EEPROM or flash. During this time the device should not be reset; otherwise, unexpected changes may be made to portions of EEPROM or flash. The reset sources (see Reset Sources on page 151) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. Also, configure the low-voltage detect circuits to generate an interrupt instead of a reset.

12. EMIF

PSoC® 3 architecture provide an external memory interface (EMIF) for connecting to external memory devices and peripheral devices. The connection allows read and write access to the devices. The EMIF operates in conjunction with UDBs, I/O ports, and other PSoC 3 components to generate the necessary address, data, and control signals.

The EMIF does not intercept address data between the PHUB and the I/O ports. It only generates the required control signals to latch the address and data at the ports. The EMIF generates a clock to run external synchronous and asynchronous memories. It can generate four different clock frequencies, which are the bus clock divided by 1, 2, 3, or 4.

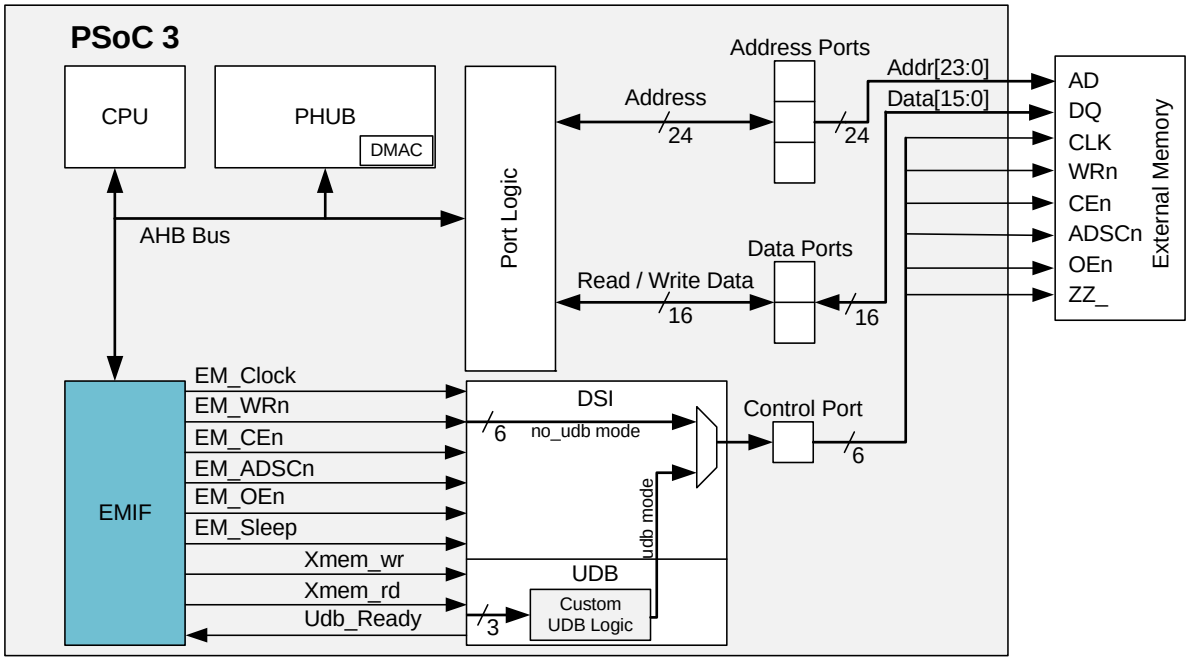
12.1 Features

The EMIF supports four types of external memory: synchronous SRAM, asynchronous SRAM, cellular RAM/PSRAM, and NOR flash. External memory can be accessed via the 8051 xdata space; up to 24 address bits can be used. The memory can be 8 or 16 bits wide.

12.2 Block Diagram

Figure 12-1 is the EMIF block diagram.

Figure 12-1. EMIF Block Diagram



12.3 How It Works

The address component of the EMIF uses up to three I/O ports. The I/O ports used for external memory address are selected by configuring the 3-bit portEmifCfg field in the PRT*_CTL register. The register can be configured so that the port is selected as either the most significant byte, the middle byte, or the least significant byte of the address. (See the I/O System chapter on page 159 for details of the PRT*_CTL register.)

The data component of the EMIF uses one or two I/O ports. The I/O port or ports used for external memory data are selected by configuring the 3-bit portEmifCfg field in the PRT*_CTL register. The register can be configured so that the port is selected as either the most significant byte or the least significant byte of the data. (See the I/O System chapter on page 159 for details of the PRT*_CTL register.)

The control component of the EMIF uses a single I/O port. The I/O port used for external memory control is selected by configuring the 3-bit portEmifCfg field in the PRT*_CTL register. The I/O port must be further configured by setting the byPass bit in the PRT*_BYP register. This allows the EMIF to drive the pins. The control signals are sent from the EMIF to the I/O port over the digital signal interface (DSI).

12.3.1 List of EMIF Registers

This table lists EMIF registers.

Table 12-1. EMIF Registers

Register	Usage
EMIF_NO_UDB	Controls whether a synchronous or asynchronous RAM is supported, versus a custom memory interface requiring additional UDB logic.
EMIF_RP_WAIT_STATES	Number of additional wait states used in a read operation.
EMIF_MEM_DWN	Puts the external memory into a power down state.
EMIF_MEMCLK_DIV	Sets the clock divider for the external memory clock frequency, which can equal the bus clock frequency divided by 1, 2, 3 or 4. Note that the external memory clock frequency cannot exceed 33 MHz.
EMIF_CLOCK_EN	Enables/disables the clock for the EMIF block, effectively turning the block on or off.
EMIF_EM_TYPE	Controls whether to generate control signals for a synchronous or asynchronous SRAM in NO_UDB mode.
EMIF_WP_WAIT_STATES	Number of additional wait states used in a write operation.

12.3.2 External Memory Support

Table 12-2 on page 115 shows how different external memory types can be connected to the PSoC 3 devices. Address lines use up to three I/O ports. Data lines use one or two ports, depending on whether the external memory is x8 or x16. Control lines use 3 to 6 pins on one I/O port. Spare pins on the address and data ports are not available for any other purpose. Spare pins on the control port are available for other purposes.

Figure 12-2. Synchronous SRAM

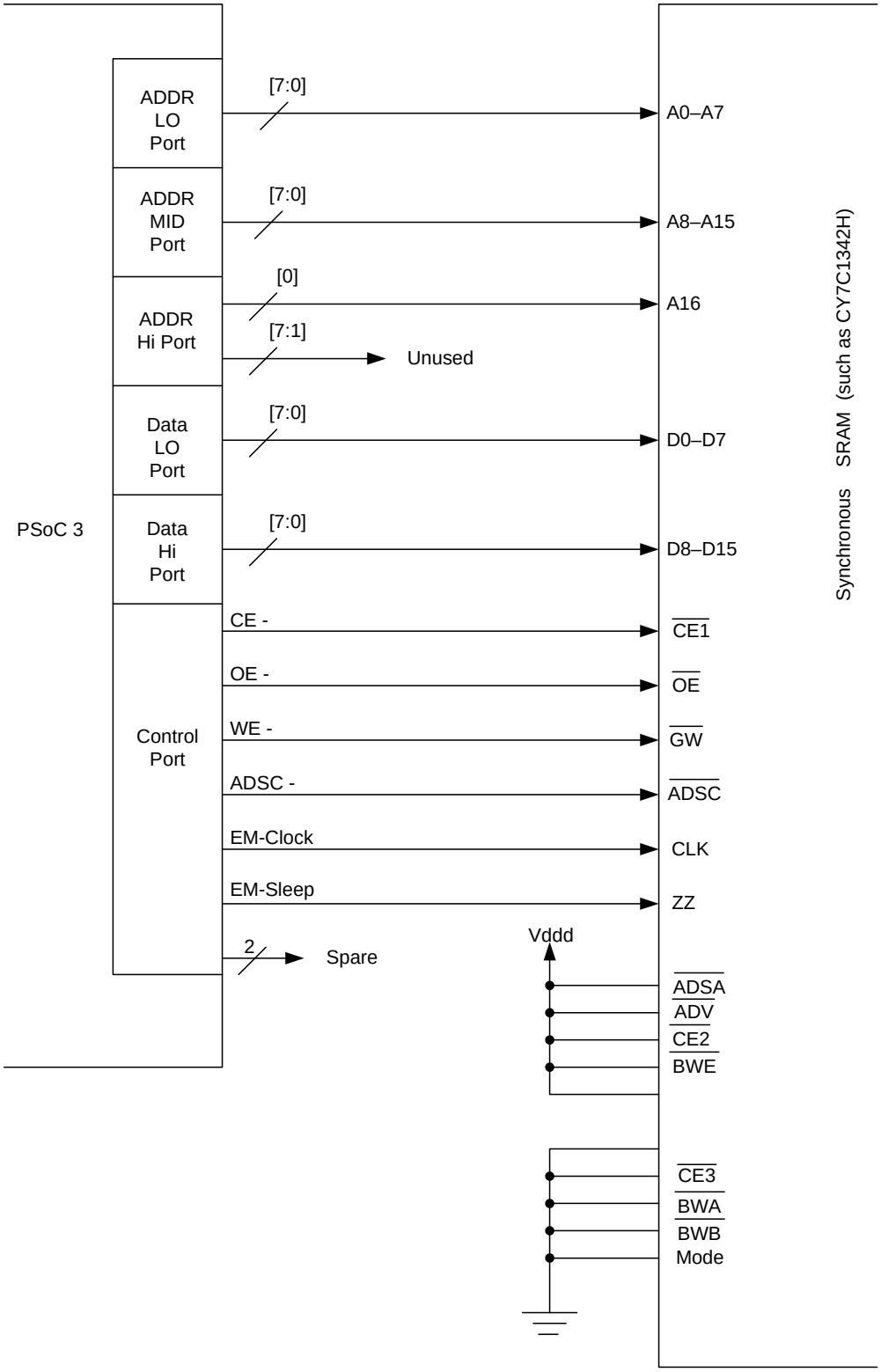


Figure 12-3. Asynchronous SRAM

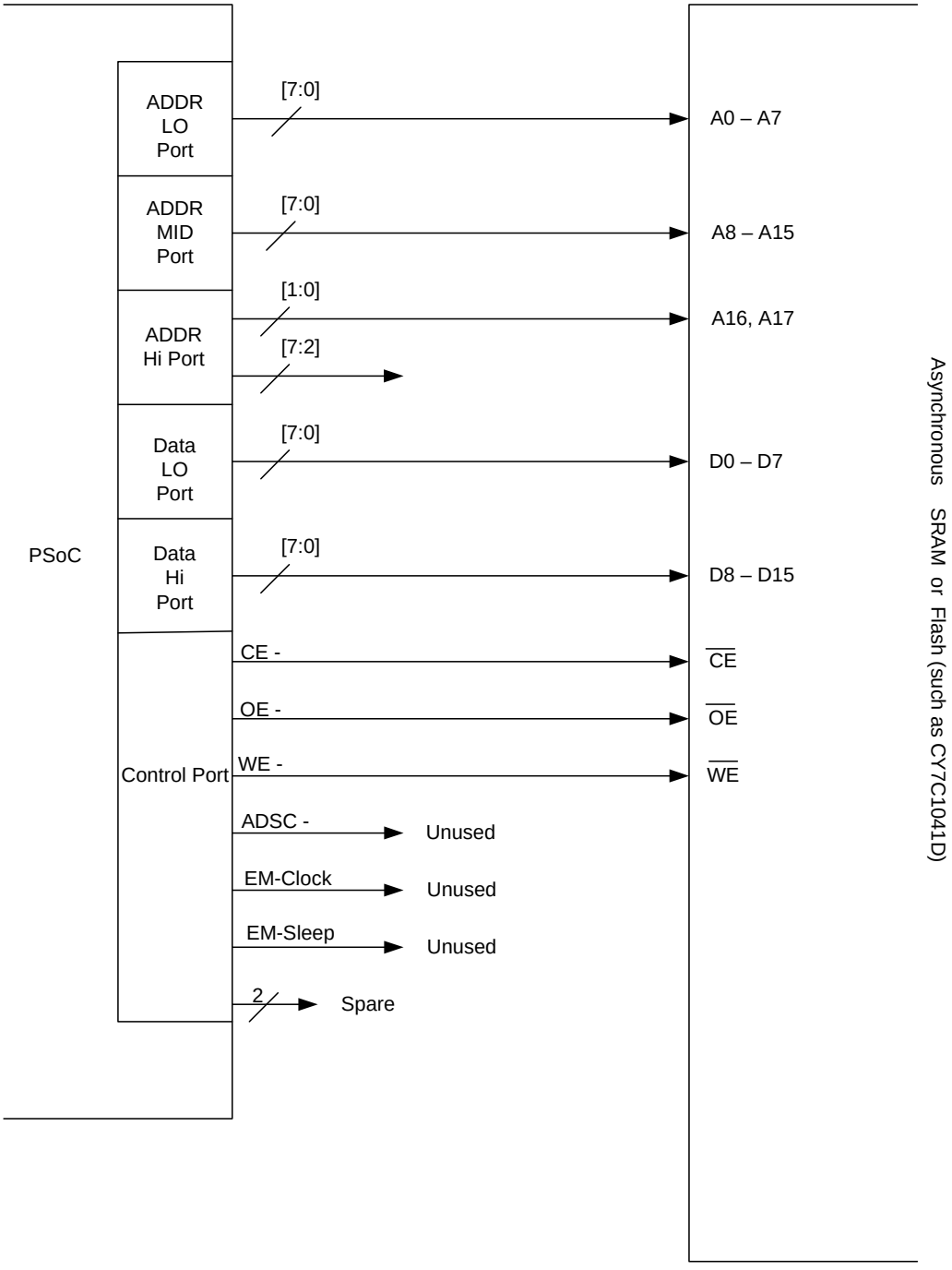


Table 12-2. External Memory Connections to PSoC 3 Devices

PSoC 3Connection	Synchronous SRAM Ex: CY7C1342H	Asynchronous SRAM Ex: CY7C1041D	Pseudo SRAM Ex: CYK256K16MCCB	NOR Flash Ex: Intel 28F800C3
3 I/O PORTs	A0 - A16	A0 - A17	A0 - A17	A0 - A18
2 I/O PORTs	D0 - D15	D0 - D15	D0 - D15	D0 - D15
1 I/O PORT pin: EM_$\overline{CE}$	$\overline{CE1}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$
1 I/O PORT pin: EM_$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$
1 I/O PORT pin: EM_$\overline{WE}$	$\overline{GW}$	$\overline{WE}$	$\overline{WE}$	$\overline{WE}$
1 I/O PORT pin: EM_$\overline{ADSC}$	$\overline{ADSC}$			
1 I/O PORT pin: EM_CLOCK	CLK			
1 I/O PORT pin: EM_SLEEP	ZZ			$\overline{RP}$[a]
tie high	$\overline{ADSP}$			$\overline{WP}$
tie high	$\overline{ADV}$			
tie high	CE2			
tie high	$\overline{BWE}$			
tie low	$\overline{CE3}$			
tie low	$\overline{BWA}$	$\overline{BHE}$	$\overline{BHE}$	
tie low	$\overline{BWB}$	$\overline{BLE}$	$\overline{BLE}$	
tie low	MODE			

a. $\overline{RP}$ is opposite polarity from the ZZ signal on the synchronous SRAM. Either add an inverter to the EM_SLEEP signal or program the EMIF_MEM_DOWN register with the opposite polarity.

12.3.3 Sleep Mode Behavior

All EMIF registers keep their value during sleep mode. The MEM_DWN register controls external memory sleep mode; the external control signal ZZ is asserted or deasserted. If an external memory access happens when MEM_DWN is set, ZZ is not asserted until after the current transfer is completed. ZZ is deasserted when the MEM_DWN register is cleared; it then takes two external memory clock cycles for the memory to wake up.

To completely turn off the EMIF block, clear the CLOCK_EN register.

12.4 EMIF Timing

The EMIF is clocked by bus clock – the same signal that clocks the CPU and the PHUB. Within the EMIF block, the bus clock can be divided by 1, 2, 3, or 4; the output is the EM_CLOCK signal to the external memory IC.

The following table shows the number of PHUB wait states generated by the EMIF depending on how much the input clock is divided.

Table 12-3. PHUB Wait States Generated by EMIF

EM_CLOCK = Bus Clock Divided By	Read Wait States	Write Wait States
1	1	2
2	3	4
3	5	6
4	7	8

The EMIF.WAIT_STATES register can also be used to add up to seven more wait states.

An important limitation is that the maximum I/O rate of PSoC 3 GPIO pins is 33 MHz. This makes the maximum frequency of EM_CLOCK 33 MHz. The following table shows limitations of EM_CLOCK frequency relative to the bus clock:

Table 12-4. Limitations of EM_CLOCK Relative to Bus Clock

Bus Clock Frequency	EM_CLOCK = Bus Clock Divided By
< 33 MHz	1, 2, 3, or 4
33 - 66 MHz	2, 3, or 4
> 66 MHz	3 or 4

The maximum frequency of the bus clock is 67 MHz for PSoC 3 devices. In most cases, EMIF_MEMCLK_DIV must be used to divide EM_CLOCK to a frequency less than or equal to 33 MHz.

Given the above restriction on EM_CLOCK frequency, and the relation of EM_CLOCK to EM_ADSC-, EM_CE-, and EM_WE-, it can be seen that the minimum pulse widths of these signals is 30.3 ns.

Figure 12-4. Synchronous Write Cycle Timing

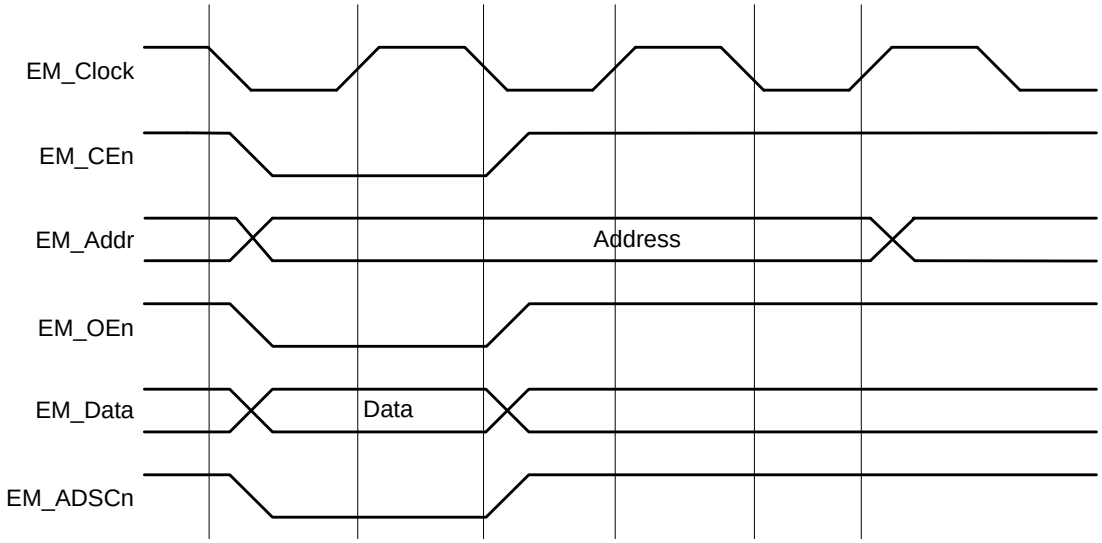


Figure 12-5. Synchronous Read Cycle Timing

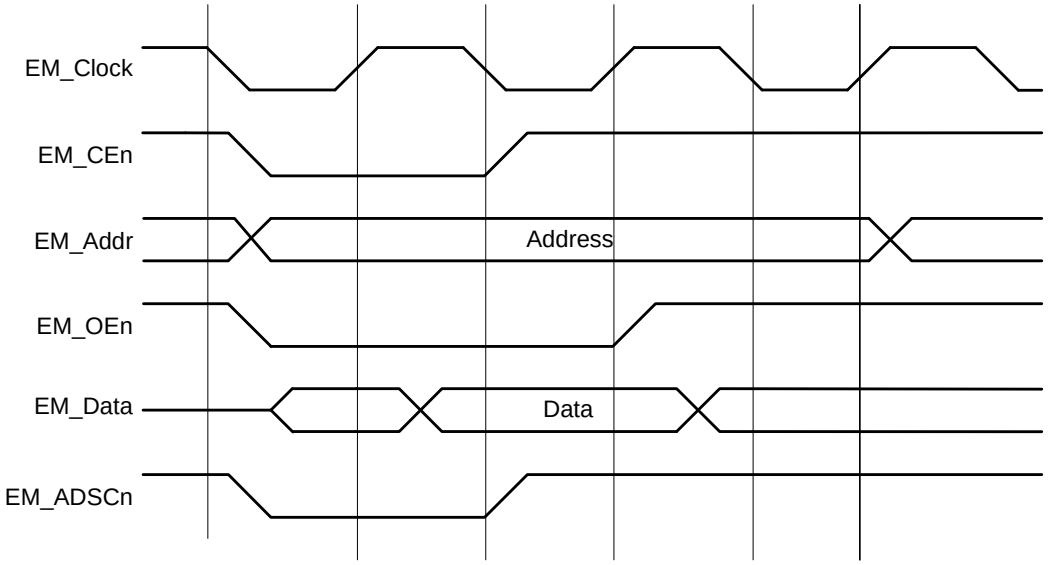


Figure 12-6. Asynchronous Write Cycle Timing

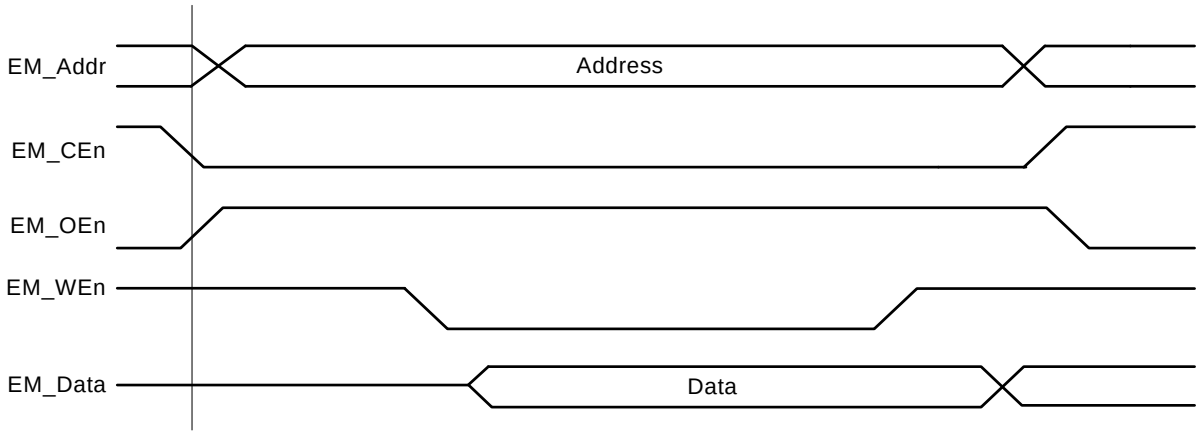
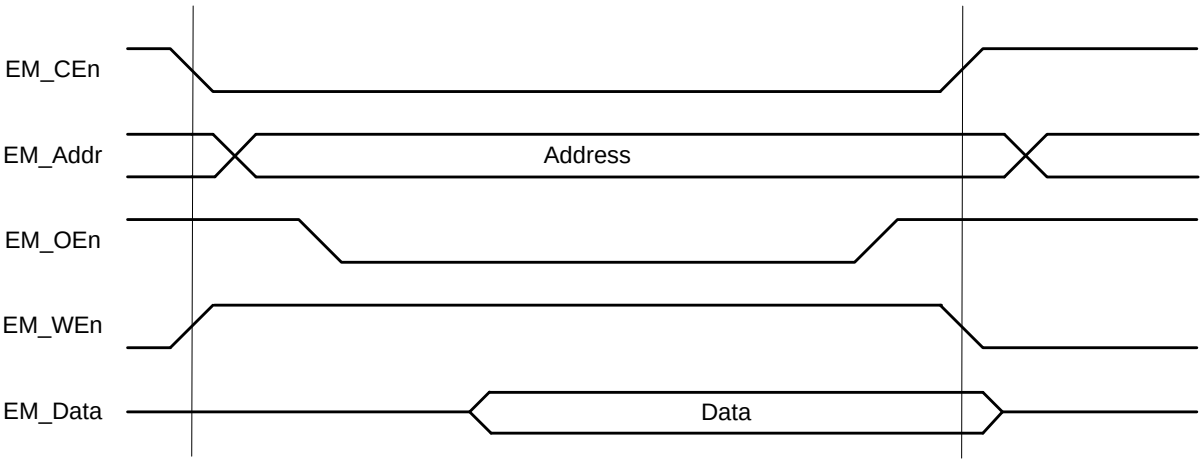


Figure 12-7. Asynchronous Read Cycle Timing



12.5 Using EMIF with Memory-Mapped Peripherals

The EMIF can also be used with external peripheral devices that have a bus interface similar to asynchronous memory devices, that is, they address, data, CE-, WE-, and OE-. The speed of the interface must be considered in the same manner as described above. The maximum data bus size is 16 bits, and the minimum address bus size is 8 bits. If multiple external memory and peripheral devices are used, address decoding to the multiple device selects may become complex and must be given careful consideration.

12.6 Additional Configuration Guidelines

The PHUB assumes all peripherals including external memory are byte addressable. Port logic is natively 16 bits wide, so care must be taken when setting up communication with either an 8 or 16 bit external memory. The following section describes some guidelines to configure the port pins and set up the memory access methods (either CPU or DMA) for optimal performance.

12.6.1 Address Bus Configuration

Configure three of the available ports as output EMIF address ports. Because PHUB peripherals are byte addressable regardless of the external memory data bus size, up to 2^24 bytes of external memory can be accessed. If an 8-bit memory is used, up to 24-bit address lines can be directly connected to the memory. If a 16-bit memory is used, the LSB address line (A0) of the memory chip should be connected to the second address line (A1) of the PSoC and the LSB address line (A0) of the PSoC should be ignored. This is because the PHUB increments the address by 2 while doing 16-bit transactions.

12.6.2 Data Bus Configuration

For 16 bit memories, two ports should be configured as bidirectional EMIF data ports. For 8bit memories, only one port should be configured as a bidirectional EMIF data port.

12.6.3 16-bit Memory Transfers

DMA Transfers: For DMA transfers to/from 16bit external memory, odd burst counts are not supported because 8 bit transfers are not supported on a 16bit interface.

CPU Transfers: With the 8bit 8051 processor in PSoC 3, 16-bit external memory cannot be directly accessed by the CPU.

12.6.4 8-bit Memory Transfers

DMA Transfers: For DMA transfers to/from an 8 bit external memory, the burst count should always be 1, irrespective of the transfer count. For example, if the burst count is set as 2 to transfer two bytes to external memory, the PHUB will try to do a 16-bit transfer in a single burst instead of breaking the transfer down into two individual transfers with the 8-bit memory.

13. Memory Map

All PSoC® 3 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.

13.1 Features

The PSoC 3 memory map has the following features:

- Flash is accessed in the 16-bit (64 KB) 8051 code space.
- All other memories, and all registers, are accessed in the 24-bit 8051 external memory space.
- 8051 has internal SFRs to provide fast access to some registers. Refer to the 8051 Core chapter on page 37 for details.

13.2 Block Diagram

There is no block diagram associated with the memory map.

13.3 How It Works

The PSoC 3 memory maps are detailed in the following sections. For additional information see the *PSoC® 3 Registers TRM (Technical Reference Manual)*.

13.3.1 PSoC 3 Memory Map

The map of the 8051 external memory (xdata) space is listed in Table 13-1. For additional information, refer to the 8051 Core chapter on page 37.

Table 13-1. PSoC 3 Memory Map

Address Range	Purpose
0x00 0000 - 0x00 1FFF	SRAM
0x00 4000 - 0x00 42FF	Clocking, PLLs, and oscillators
0x00 4300 - 0x00 43FF	Power management
0x00 4400 - 0x00 44FF	Interrupt controller
0x00 4500 - 0x00 45FF	Ports interrupt control
0x00 4700 - 0x00 47FF	Flash programming interface
0x00 4900 - 0x00 49FF	I^2C controller
0x00 4E00 - 0x00 4EFF	Decimator
0x00 4F00 - 0x00 4FFF	Fixed timer/counter/PWMs
0x00 5000 - 0x00 51FF	General purpose I/Os
0x00 5300 - 0x00 530F	Output port select register
0x00 5400 - 0x00 54FF	External memory interface control registers
0x00 5800 - 0x00 5FFF	Analog subsystem interface
0x00 6000 - 0x00 60FF	USB controller
0x00 6400 - 0x00 6FFF	UDB configuration
0x00 7000 - 0x00 7FFF	PHUB configuration
0x00 8000 - 0x00 8FFF	EEPROM
0x00 A000 - 0x00 A400	CAN
0x00 C000 - 0x00 C800	Digital filter block
0x01 0000 - 0x01 FFFF	Digital interconnect configuration
0x05 0220 - 0x05 02F0	Debug controller – accessible via JTAG/SWD only, not accessible via PHUB
0x08 0000 - 0x08 1FFF	Flash ECC bytes
0x80 0000 - 0xFF FFFF	External Memory Interface (EMIF)

Section D: System Wide Resources

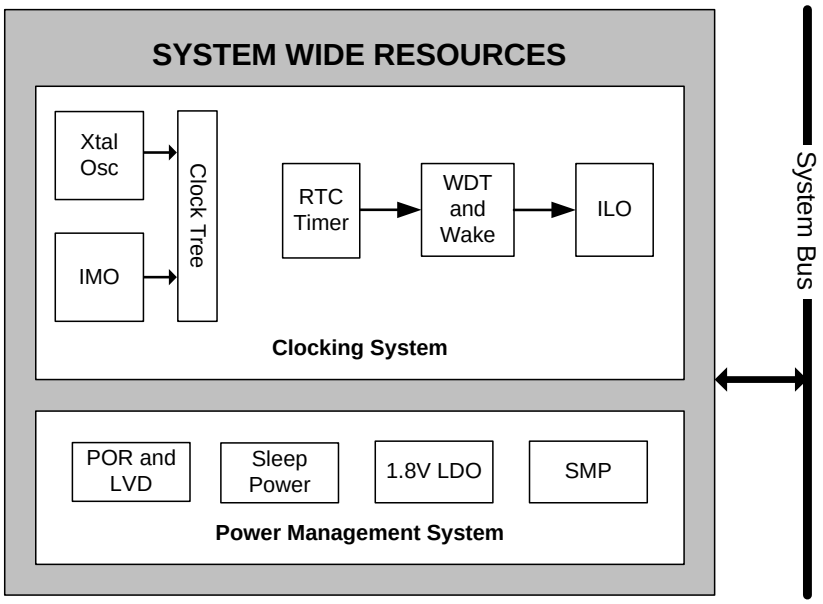
The System Wide Resources section details three types of I/O, internal clock generators, power supply, boost converter, and sleep modes.

This section contains these chapters:

- Clocking System chapter on page 123
- Power Supply and Monitoring chapter on page 137
- Low-Power Modes chapter on page 145
- Watchdog Timer chapter on page 149
- Reset chapter on page 151
- Auxiliary ADC chapter on page 179
- I/O System chapter on page 159
- Flash, Configuration Protection chapter on page 175

Top Level Architecture

System Wide Resources Block Diagram



14. 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.

14.1 Features

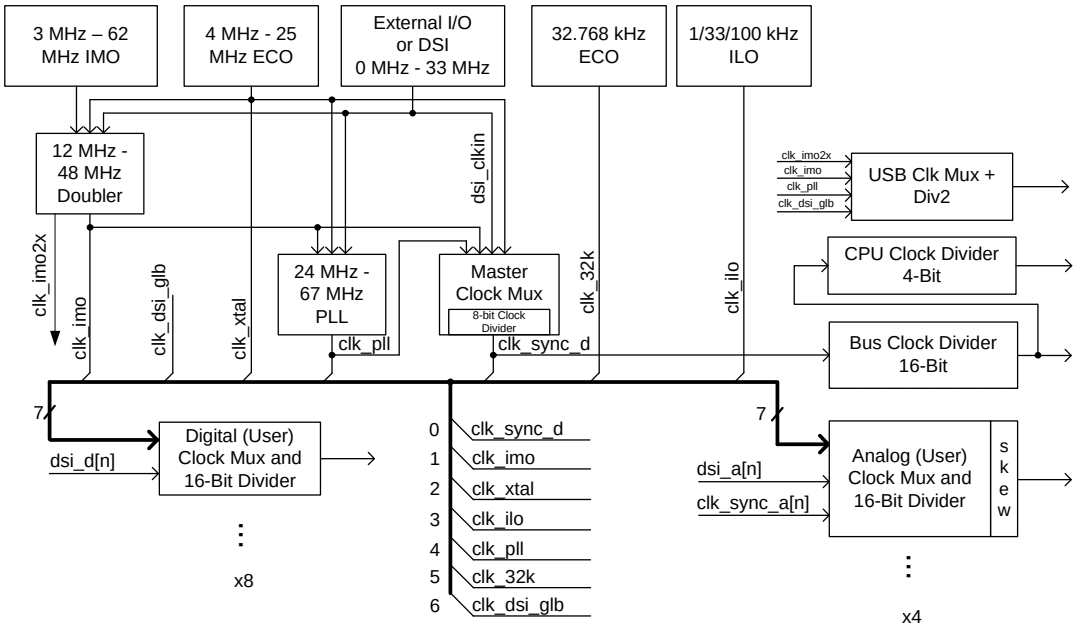
The clock system includes these clock resources:

- Four internal clock sources increase system integration:
  - 3 to 62 MHz internal main oscillator (IMO) ±1% at 3 MHz
  - 1 kHz, 33 kHz, 100 kHz internal low-speed oscillator (ILO) outputs
  - 12 to 67 MHz clock doubler output, 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)

14.2 Block Diagram

Figure 14-1 gives a generic view of the Clocking System in PSoC 3 devices.

Figure 14-1. 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

14.3 Clock Sources

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

14.3.1 Internal Oscillators

PSoC devices have two internal oscillators: the internal main oscillator (IMO) and the internal low-speed oscillator (ILO).

14.3.1.1 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, 48, and 62 MHz. Frequencies are selected using the register FASTCLK_IMO_CR[2:0]. The clock accuracy is 1% typical at 3 MHz and it varies with frequency. See the device datasheet for IMO accuracy specification.

Clock Doubler

The block has one additional clock output. A doubled clock, IMOCLKX2 outputs a clock at twice the frequency of the input clock. The doubler works for input frequencies in the range 6 – 33 MHz. The doubler is enabled by register bit FASTCLK_IMO_CR[4]. The doubler can also take clock inputs (XCLK) other than IMO and have a DSI or MHzECO as input. This feature is enabled by the bit FASTCLK_IMO_CR[5]. The DSI / MHzECO can be selected in the CLKDIST_CR[6] register bit.

The clock distribution register CLKDIST_CR[5:4] is responsible for selecting between IMO or IMO × 2 outputs.

Figure 14-2 is a summary block diagram of the IMO. **Note** The output of the clock doubler should only be used for clocking the USB block. It should not be used to clock any other peripherals in the device.

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 running user code.

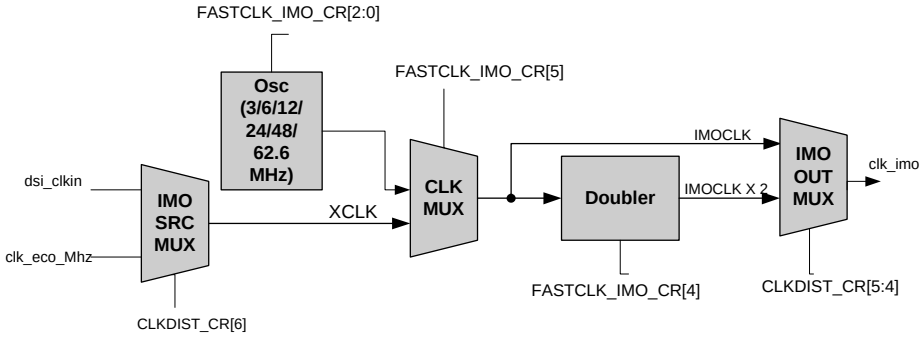
During the transition from FIMO to regular IMO, glitches can occur if the frequency selection for the two configurations are not the same. Stated explicitly, at the transition, FASTCLK_IMO_CR[2:0] should match PWRSYS_WAKE_TR1[2:0].

NVL Frequency Selection

Upon entering the boot phase of startup, the IMO frequency and a portion of its trim are set using values stored in user NVLs. This allows the user to select a faster clock frequency for a portion of device startup. The top two bits of IMO trim stored in the IMO_TR2 register are populated from the NVL register MNVL_FIMO_TRIM[1:0]. The frequency selection bits in register FASTCLK_IMO_CR[2:0] have their most significant bit populated using NVL register CNVL_CFGSPEED. The NVL register will set the frequency to 12 MHz when set to 0, and 48 MHz when set to 1. This NVL selection will be overwritten during firmware startup with a more complete frequency selection and trim.

Note 48-MHz startup should not be selected in devices with a maximum operating frequency rating below 48 MHz.

Figure 14-2. IMO Block Diagram



14.3.1.2 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, or it can be used to time intervals such as for sleep mode entry and exit. A third 33-kHz clock output is available — a divide-by-3 of the 100 kHz output.

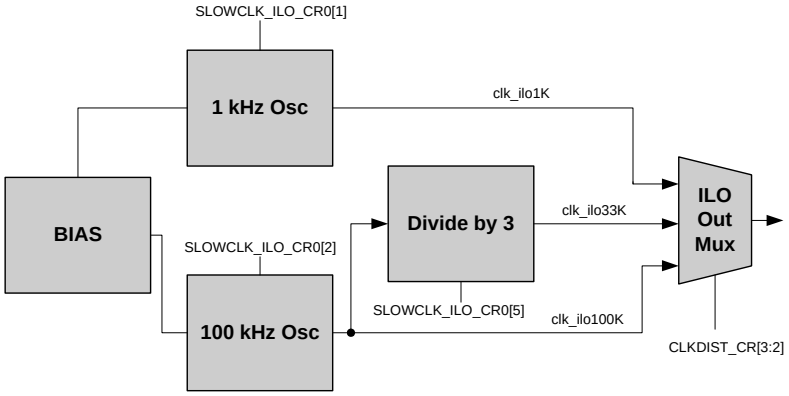
In addition to the multiplexed output that can enter the clock distribution, the output clocks route to the following functions:

- clk_ilo1K – to the central timewheel (also called the sleep timer) and watchdog timer. See the Low-Power Modes chapter on page 145 for more details.
- 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 14-3 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.

Figure 14-3. ILO Block Diagram



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.

14.3.2 External Oscillators

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

14.3.2.1 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, and it can also route through the IMO doubler to produce doubled frequencies, if the crystal frequency is in the valid range for the doubler.

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 14-4 is a block diagram of the MHzECO.

Figure 14-4. MHzECO Block Diagram

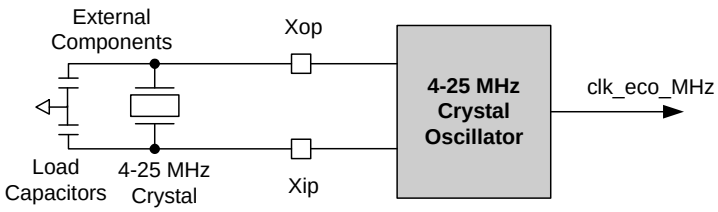
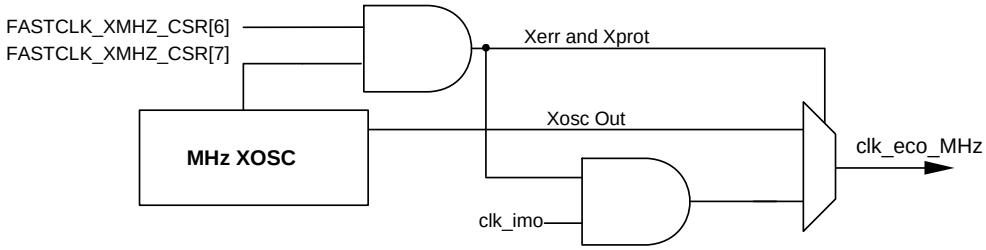


Figure 14-5. MHzECO Oscillator Fault Recovery



Fault Recovery

The block contains an option to detect crystal oscillator failure. Clock failure is detected by comparing the amplitude of the XIn signal to a user-selectable voltage. This voltage is selected using the "vref_sel_wd" bits in the register FASTCLK_XMHZ_CFG1. The clock failure can occur due to environmental conditions (such as moisture) that affect the crystal and cause oscillators to stop. Clock failure status is indicated by the clock error status bit (FASTCLK_XMHZ_CSR[7]).

If the FASTCLK_XMHZ_CSR[6] bit is set, the fault recovery option is enabled. In this case, when the crystal oscillator fails, the crystal oscillator output is driven low. The IMO is enabled (if it is not already running), and the IMO output routes through the crystal oscillator output mux. It takes six IMO cycles after error signal assertion for the IMO to appear outside of the block. In this way, the system can continue to operate through a crystal fault. This functionality is illustrated in Figure 14-5.

Low-power Operation

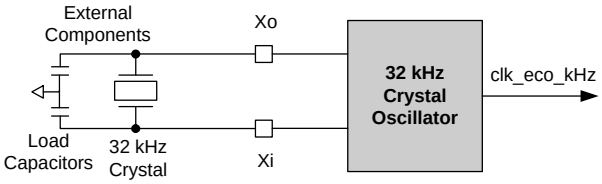
The MHz crystal oscillator does not operate in the SLEEP and HIBERNATE modes. This means that you need to disable the oscillator to enter SLEEP and HIBERNATE modes. The 32 kHz crystal oscillator can be kept active, for precise timing (RTC), in SLEEP mode. If the MHz crystal oscillator is not disabled 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 SLEEP mode, 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.

14.3.2.2 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 14-6 is a block diagram of the kHzECO.

Figure 14-6. 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, the oscillator is configured for high-power mode, which consumes 1-3 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 does not stabilize instantly, and requires some time to oscillate consistently. The ANA_STAT (SLOWCLK_X32_CR[5]) bit indicates whether oscillation is stable after measuring the waveform's amplitude. The oscillator must always be started in high power mode to avoid excessively long startup delays.

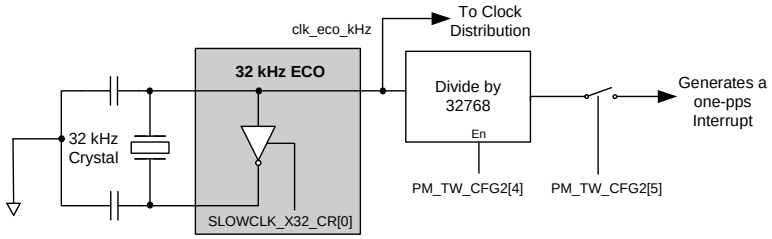
Real Time Clock

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 14-7.

The RTC timing is derived from the 32 kHz external crystal oscillator, as shown in Figure 14-7. 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. See the UDB Array and Digital System Interconnect chapter on page 225 for more details on usage. RTC functionality is available for use in all power modes except the Hibernate mode.

Figure 14-7. RTC Implementation



14.3.3 Oscillator Summary

A summary of the oscillator output frequency ratings is listed in Table 14-1.

Table 14-1. Oscillator Summary

Source	Fmin	Fmax
IMO	3 MHz	62 MHz
ILO	1 kHz	100 kHz
MHzECO	4 MHz	25 MHz
kHzECO	32.768 kHz	
PLL	24 MHz	67 MHz

14.3.4 DSI Clocks

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

14.3.5 Phase-Locked Loop

The on-chip Phase-Locked Loop (PLL) can be used to boost the clock frequency of the selected clock input (i.e., IMO, MHzECO, and DSI clock) to run the device at maximum operating frequency. The PLL can synthesize clock frequencies in the range of 24 – 67 MHz. Its input and feedback dividers allow fine enough resolution to create many desired system clock frequency. The PLL output routes to the clock distribution network as one of the possible input sources. The PLL is shown in Figure 14-8.

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 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.

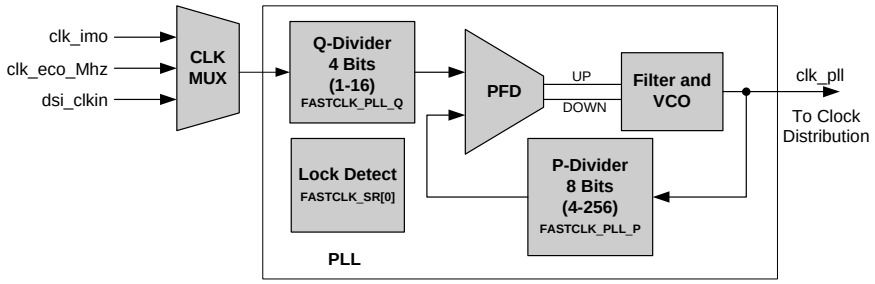
The PLL's charge pump current (Icp) can be configured using bits 6:4 of register FASTCLK_PLL_CFG1. This bitfield should be set to 0x01 for all configurations.

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.)

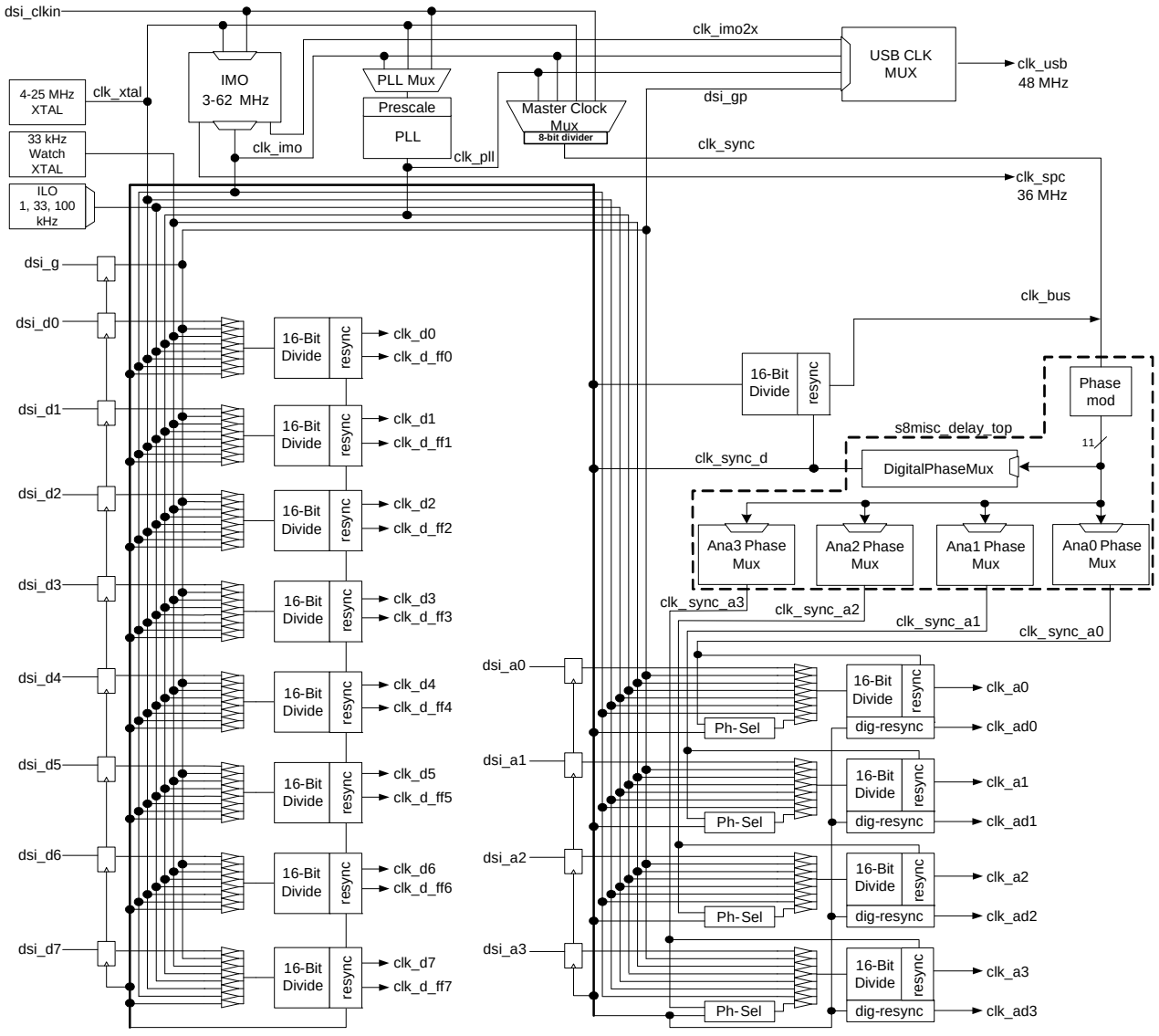
Figure 14-8. PLL Block Diagram



14.4 Clock Distribution

All of the clock sources discussed are distributed into the various domains of the device through clock distribution logic. Figure 14-9 shows a block diagram of the clock distribution system.

Figure 14-9. Clock Distribution System



All of 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 eight input mux. Also, the divider outputs are synchronized to their respective domain clocks.

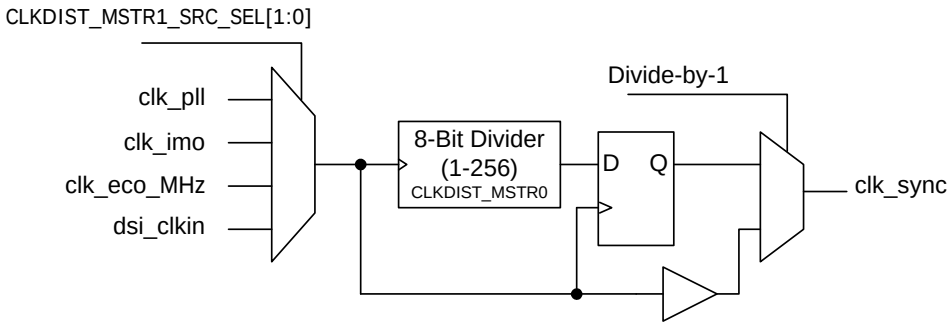
A Master Clock Mux is available for distributing the sync clocks. There are options to provide delay on the digital sync clock. All eight digital dividers are synchronized to the same digital clock, but each of the analog clock divider outputs can be synchronized to analog clocks of different delays. 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.

14.4.1 Master Clock Mux

The Master Clock Mux, shown in Figure 14-10, 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 14-10. Master Clock Mux

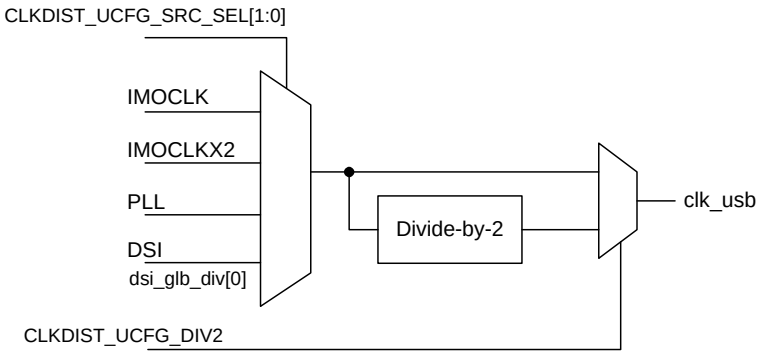


14.4.2 USB Clock

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 14-11, provides the clock to the USB logic.

Figure 14-11. 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 for 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].

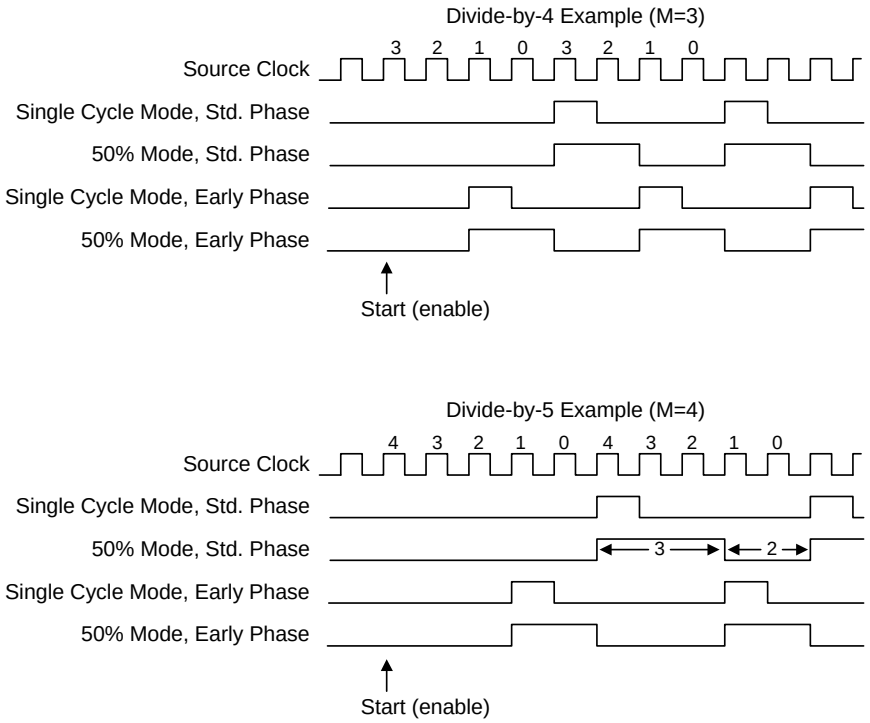
Alternately, a 24 MHz crystal controlled clock doubled to 48 MHz can be supplied for Full Speed USB operation. Other crystal frequencies, such as 4 MHz can be used with the PLL to synthesize the necessary 48 MHz.

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.

14.4.3 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 14-12.

Figure 14-12. 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 132).

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.

14.4.3.1 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 14-12.

14.4.3.2 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 14-12 on page 131 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.

14.4.3.3 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 called the Early Phase because the output can be shifted earlier in time to an approximate count, which 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.

14.4.4 Clock Synchronization

All digital and analog divider outputs can be synchronized to the clk_sync_dig signals (CLKDIST_DCFG[x]_CFG2[3] or CLKDIST_ACFG[x]_CFG2[3]), as shown in Figure 14-13.

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.

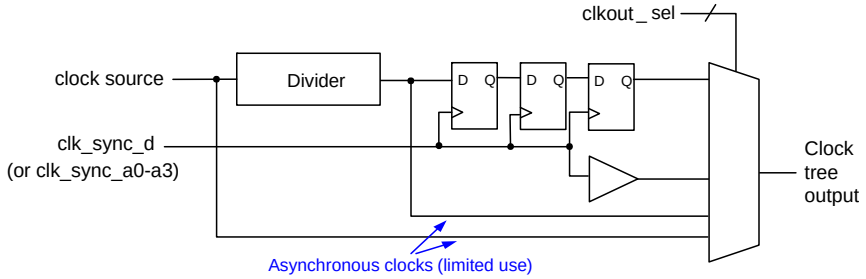
The output of each clock tree provides for selection of one of four output clocks:

- **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 134. 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.

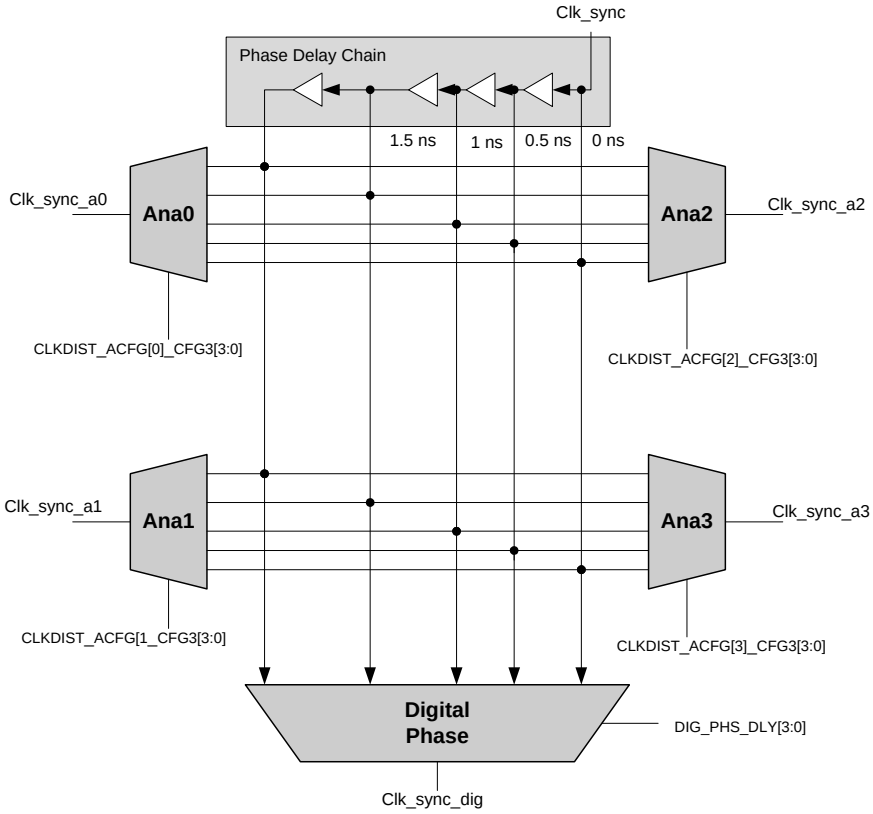
Figure 14-13. Resync Option Diagram



14.4.5 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 14-14.

Figure 14-14. 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.

14.4.6 Divider Update

To allow for 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 is 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.

14.4.7 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.

14.4.8 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.

14.4.9 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.

14.5 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 and buzz modes. See the Low-Power Modes chapter on page 145 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.

14.6 Clock Naming Summary

Table 14-2 lists clock signals and their descriptions.

Table 14-2. 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

15. Power Supply and Monitoring

PSoC® 3 devices have separate external analog and digital supply pins, labeled Vdda and Vddd, respectively. 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 datasheet.

15.1 Features

These regulators are available:

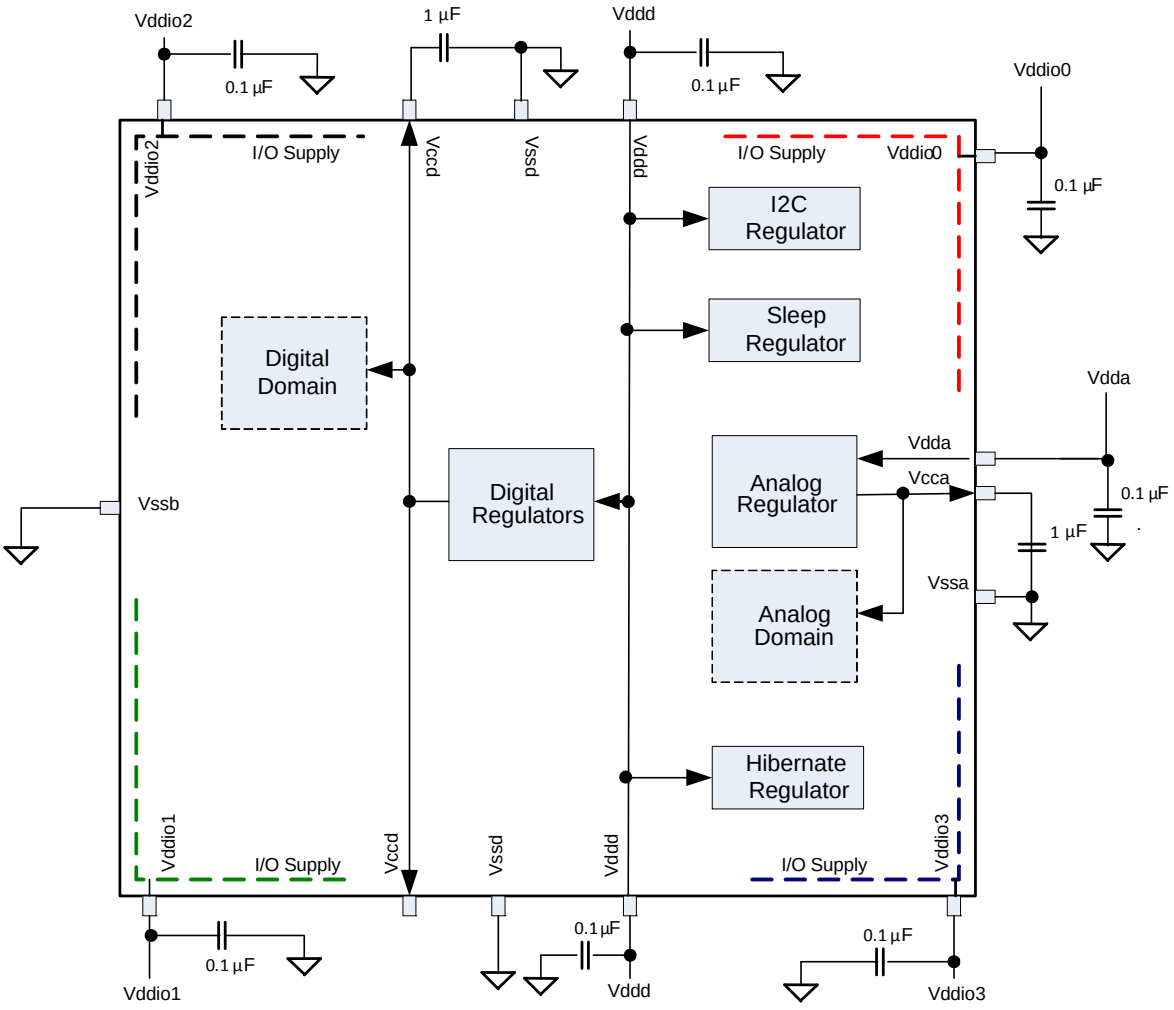
- Analog regulator for the analog domain supply
- Digital regulator for the digital domain supply
- Sleep regulator for the sleep domain
- I^2C regulator to power the I^2C logic
- Hibernate regulator to supply keep-alive power for state retention during hibernate mode

15.2 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 and the Vddio pins must have capacitors connected, as shown in Figure 15-1. The power system also contains a sleep regulator, an I^2C regulator, and a hibernate regulator.

Vdda must be greater than or equal to all other power supply pins (Vddd, Vddios) in PSoC 3. 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 15-1. Power Domain Block Diagram



15.3 How It Works

The regulators shown in Figure 15-1 power the various domains of the device. All regulators, except the analog regulator, draw their input power from the Vddd pin supply.

15.3.1 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 sleep and hibernate regulators are designed to fulfill power requirements in the low-power modes of the device.

15.3.1.1 Internal Regulators

For external supplies from 1.8 V to 5.5 V, regulators are powered and the supply is provided through the Vddd/ Vdda pins. An external cap of ~1 µF is connected to the Vccd and Vcca pins.

For the 1.71 V < Vcc < 1.89 V external supply, power up the device with Vccd/Vcca pins. In this mode, short the Vddd pin to Vccd and short the Vdda pin to Vcca. The internal regulator remains powered by default. After power up, disable the regulators, using the register PWRSYS.CR0 to reduce power consumption.

15.3.1.2 Sleep Regulator

The sleep regulator supplies power to these circuits during the device sleep mode.

- 32 kHz ECO
- ILO
- RTC Timer
- WDT
- Central Timewheel (CTW)
- Fast Timewheel (FTW)

15.3.1.3 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.

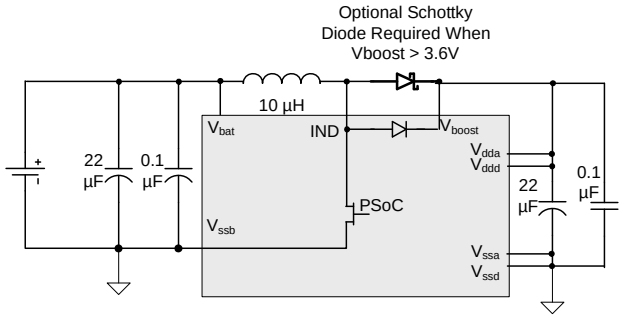
15.3.2 Boost Converter

PSoC devices also have a boost converter that accepts an input voltage supplied by a battery or other source and produces a selectable, higher output voltage than the input voltage; the voltage is boosted.

The input voltage can be from various sources, such as a battery or solar cell. The converter uses an external inductor to boost the voltage. An external Schottky Diode must be connected between the pins IND and Vboost when boost voltage is greater than 3.6 V.

Figure 15-2 is an application diagram of the boost converter.

Figure 15-2. Boost Converter Application Diagram



The boost converter is enabled or disabled by the register bit BOOST_CR1[3]. The device provides the option of changing the boost output voltage by writing into the register BOOST_CR0[4:0]. By default, at startup the boost converter is enabled and configured for a 1.8-V output. If the boost converter is not used in a given application, tie the VBAT, VSSB, and VBOOST pins to ground and leave the IND pin unconnected.

When using the boost converter to power the PSoC 3 device, power the Vdda pin also using the boost converter output, as shown in Figure 15-2. If the Vdda pin is powered by a different power supply with boost enabled, ensure that the pin is in the valid operating range before the Vboost pin. If this condition is not met, the boost registers may be written incorrectly and boost may malfunction.

When the bus clock is configured to a frequency greater than 24 MHz, the boost configuration registers – BOOST_CR0, BOOST_CR1, BOOST_CR2, and BOOST_CR3 – must be read with two consecutive read operations, discarding the result from the first read. It is not allowed to access any other boost configuration register between two reads of a boost configuration register, but it is acceptable to access non-boost register. Interrupts should also be disabled when reading these four registers at these higher bus clock rates to prevent this scenario. This requirement of reading the control registers twice is to avoid a timing issue in accessing these registers. Writing of the boost registers can however occur at any clock rate with a single write instruction.

15.3.2.1 Operating Modes

The boost converter has two main operating modes selected by the register BOOST_CR0[6:5]; these are:

- **Active** – This is the normal mode of operation where the boost regulator actively generates a regulated output voltage. This mode is used to provide regulated power during chip Active and Alternate Active modes.

  The switching frequency is selected by BOOST_CR1[1:0] and is not synchronized to any other clock. The switching frequency options are 400 kHz (2'b01) and 'external' (2'b11). The switching frequency is derived from the 32-kHzECO block when this option is set to 'external'.
- **Standby** – This is a low-power, low-current mode where most boost functions are disabled. This mode is used to provide minimum power during chip Sleep mode. Output voltage is continuously monitored and supervisory data provided in BOOST_SR[4:0]. This register provides supervisory data against the output voltage selected. Therefore, the processor can use the thump bit BOOST_CR0[7] to switch the transistor on for a 1-µs pulse.

  In boost standby mode, the external 32-kHz crystal can be used to trigger inductor boost pulses on the rising and falling edge of the clock when the output voltage is less than the configured value. This is called automatic thump mode (ATM).

The boost operating modes must be used in conjunction with chip power modes to minimize the total chip power consumption. Table 15-1 lists the boost power modes available in different chip power modes.

Table 15-1. Chip and Boost Power Modes Compatibility

Chip Power Modes	Boost Power Modes
Chip – Active or Alternate Active mode	Boost can be operated in either active or standby mode. It is recommended to operate boost in active mode for higher current supply capabilities. Boost can be used in its standby mode when the chip is in alternate active mode for low power consumption.
Chip – Sleep mode	Boost can be operated in either active or standby mode. However, the use of boost regulator is not recommended in device sleep mode due to excessive boost current draw.
Chip – Hibernate mode	Boost can only be operated in active mode. The use of boost regulator is not recommended in device hibernate mode due to excessive boost current draw.

15.3.2.2 Status Monitoring

Status monitoring for input and output voltages of the boost converter are available in the status register BOOST_SR.

- **Output Voltage Monitor** – The BOOST_SR[4:0] register gives a status of the output voltage against the set nominal voltage output.

  Bit 4: ov – Above overvoltage threshold (nominal + 50 mV).

  Bit 3: vhi – Above high regulation threshold (nominal +25 mV).

  Bit 2: vnom – Above nominal threshold (nominal).

  Bit 1: vlo – Below low regulation threshold (nominal to 25 mV).

  Bit 0: uv – Below undervoltage limit (nominal to 50 mV).

The boost converter generates a power manager interrupt when an undervoltage event occurs. This interrupt can be configured to wake the chip from Alternate Active or Sleep mode. The BOOST_SR status register allows status monitoring for input and output voltages of the boost converter. If an undervoltage event occurs, BOOST_SR2[0] will be set to ‘1’ until the register is read or a reset event occurs.

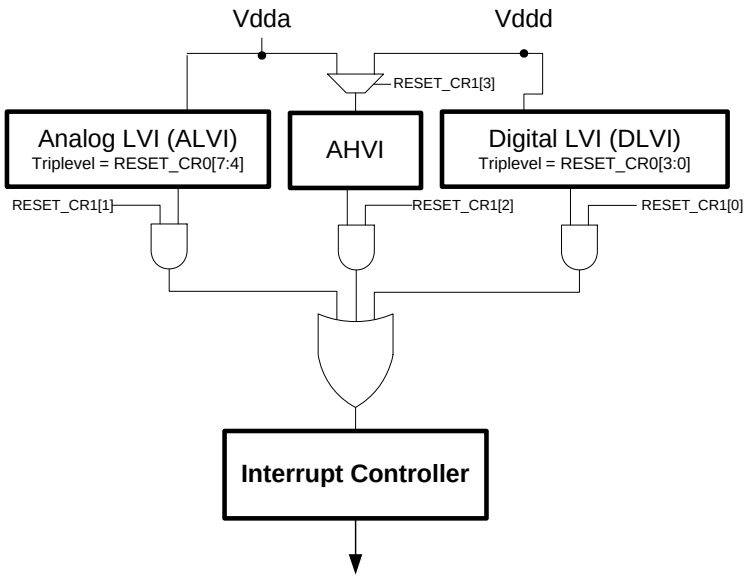
15.3.3 Voltage Monitoring

The device has two circuits for detecting 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 15-3.

Figure 15-3. Voltage Monitoring Block Diagram



15.3.3.1 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 1.7 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. Note that the LVI status bits in RESET_SR0 will be reset to ‘0’ when a device reset occurs due to a POR, LVI, or HVI condition. This bit is useful only when the LVI is configured as an interrupt source because an LVI reset also clears this bit. 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 (LVIA), enabled by RESET_CR1[1] and RESET_CR0[7:4], sets the LVIA threshold. The digital low-voltage interrupt (LVID), enabled by RESET_CR1[0] and RESET_CR0[3:0], sets the LVID threshold. Apart from this, when the voltage monitoring is enabled and the corresponding PRES bit is also enabled in RESET_CR3[7:6], the low-voltage condition triggers a corresponding reset. Both the LVIA and LVID resets are enabled by default. Note that the LVI reset will continuously occur as long as the LVI voltage condition persists. The user code configures the LVI for reset. When low-voltage condition occurs after this configuration is done, the device is reset once and the user code starts executing from flash address zero. Again, when the CPU reaches the code that configures LVI for reset and the voltage is still low, the device will be reset again. This continuous cycle of device reset occurs until the low-voltage condition is no longer present.

The interrupt is generated only when the corresponding bit in the RESET_CR1 register is set and the 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 you can determine if an under/over voltage condition is still in effect. Similar to the reset condition, the LVI interrupt is continuously triggered until the voltage goes above the low-voltage trip point.

The low-voltage detect (LVD) feature is available in active and standby modes of operation. The chip can return from standby to active mode when an LVI interrupt occurs. If LVD is required in sleep mode, then the chip should be configured to periodically wake up from sleep using CTW as the

wake up source; the LVD monitoring should be done in active mode. See the device datasheet for information on the voltage threshold settings.

15.3.3.2 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, the default selection is for the Vdda supply. 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. Note that the HVI status bits in RESET_SR0 will be reset to ‘0’ when a device reset occurs due to a POR, LVI, or HVI condition. This bit is useful only when the HVI is configured as an interrupt source because an HVI reset also clears this bit. This bit is set when the analog voltage value goes beyond the threshold value. Note that the HVI reset will continuously occur as long as the HVI voltage condition persists. The user code configures the HVI for reset. When high-voltage condition occurs after this configuration is done, the device is reset once and the user code starts executing from flash address zero. Again, when the CPU reaches the code that configures HVI for reset and the voltage is still high, the device will be reset again. This continuous cycle of device reset occurs until the high-voltage condition is no longer present.

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 you can determine if an overvoltage condition is still in effect. Similar to the LVIA/LVID events, HVIA event is also available in active and standby modes. The HVIA interrupt can return the chip to active mode from standby mode. Similar to the reset condition, the HVI interrupt is continuously triggered until the voltage goes below the high-voltage trip point.

15.3.3.3 Processing a Low/High Voltage Detect Interrupt

Both LVI and HVI circuits 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 3 generate a glitch when enabled. This may cause an unintended interrupt. Do the following workaround in firmware for this glitch condition.

1. Enable the LVI/HVI circuits by appropriately setting bits [2:0] of the RESET_CR1 register. When enabled, 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.

With the LVI configured as an interrupt, if the low-voltage condition and a soft reset (such as software reset, watchdog reset, segment reset) occur simultaneously, there is a chance that the low-voltage condition persists when the device resets due to the soft reset source. This will result in the low-voltage condition causing a hard reset as well. If a hard reset occurs, it results in the clearing of the soft reset status register bits in RESET_SR0 and RESET_SR1. The implication is that any soft reset occurring in conjunction with the LVI interrupt event will not be properly reflected in the RESET_SR0, RESET_SR1 status registers. However, there will be no impact on any other device operation; the device will undergo the normal sequence after the reset occurs. This behavior is applicable for the HVI interrupt as well.

15.3.3.4 Reset on a Voltage Monitoring Interrupt

The ALVI and DLVI can be configured to directly reset the device by setting the corresponding bits in RESET_CR3[7:6]. When this bit is set to ‘1’ along with the RESET_CR1[0/1] set to ‘1’, the corresponding LVI becomes an additional reset source through the PRES reset path. When this bit is cleared to ‘0’ along with the RESET_CR1[0/1] set to ‘1’, the corresponding LVI is only used as an interrupt source. If the RESET_CR1[0/1] is cleared to ‘0’, the bit state (either a zero or a one) has no impact on the reset or interrupt functionality.

The LVI glitch mentioned in 15.3.3.3 Processing a Low/High Voltage Detect Interrupt triggers a system reset if the LVI monitor is enabled after enabling the LVI reset (the LVI reset is enabled by default). To avoid this, disable the LVI reset by

clearing the corresponding bits in RESET_CR3[7:6] before enabling the LVI monitor.

Note that the LVI reset will not hold the device in reset until the voltage goes above the set value. When the LVI circuit detects a low voltage, reset is asserted. The reset is then released even if the voltage is still below the LVI set value.

15.4 Register Summary

Table 15-2. Power Supply Register Summary

Register	Function
PWRSYS_CR0	Regulator control
PWRSYS_CR1	Analog regulator control
BOOST_CR0	Boost Thump, voltage selection and mode select
BOOST_CR1	Boost enable and control
BOOST_CR2	Boost control
BOOST_CR3	Boost PWM duty cycle
BOOST_SR	Boost status
RESET_CR0	LVI trip value setting
RESET_CR1	Voltage monitoring control
RESET_SR0	voltage monitoring status
RESET_SR2	Real-time voltage monitoring status

16. Low-Power Modes

The PSoC® 3 devices feature a set of four power modes with a goal of reducing the average power consumption of the device.

16.1 Features

The PSoC 3 power mode features, in order of decreasing power consumption, are:

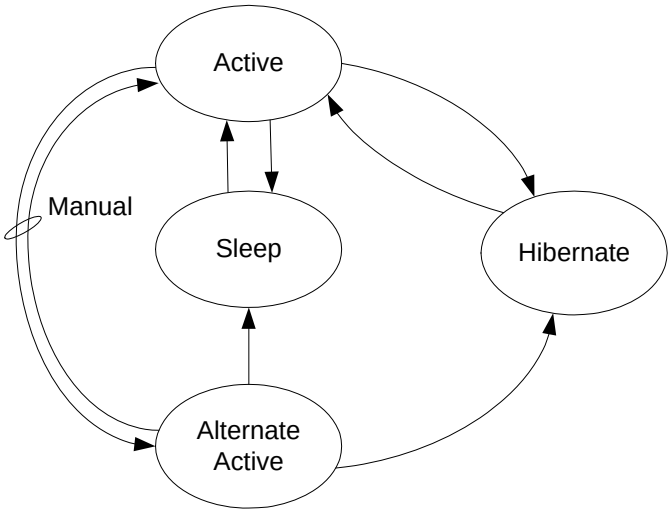
- Active
- Alternative Active
- Sleep
- Hibernate

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 16-1.

Figure 16-1. 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 (ECO32K, 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 16-1 and are listed in decreasing order of power consumption.

Table 16-1. Power Consumption-Reducing 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. Digital and analog regulators can be disabled if external regulation used.
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. Digital and analog regulators can be disabled if external regulation used.
Sleep	All subsystems automatically disabled	Manual register entry	Comparator, PICU, I^2C, RTC, CTW	ILO/kHzECO	Digital and analog regulators can be disabled if external regulation used.
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	PICU		Only hibernate regulator active.

16.2 Active Mode

Active mode is the primary power mode of the PSoC device. This mode provides the option to use every possible subsystem/peripheral in the device. 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.

16.2.1 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.

16.2.2 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.

16.3 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, you 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.

16.3.1 Entering Alternative Active Mode

To enter alternative active mode, write into [PM_MODE_CSR]. Firmware must ensure the SPC Idle bit in the SPC_SR[1] register is '1' before writing to the PM_MODE_CSR[2:0] register.

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.

16.3.2 Exiting Alternative Active Mode

Any interrupt or write to the [PM_MODE_CSR] register can return the system to active mode.

16.4 Sleep Mode

Sleep mode powers down the CPU and other internal circuitry to reduce power consumption. Supervisory services, such as the central timewheel, RTC, and WDT remain active.

When a wakeup event occurs, the system reactivates in a single phase and returns to active mode. The analog and digital LDO regulators are disabled during sleep mode. If the core supplies are configured for internal regulation, a weak keeper is used to hold the external capacitors at 1.8 V (nominal). Both regulators can be periodically activated (buzzed) to provide supervisory features for voltage monitoring and brownout detect (LVI, HVI, and PPOR). Buzzing is not required if these supervisory services are not used.

If a LVI, HVI, or Brown Out detect (power supply supervising capabilities) are required in a design during sleep, use the CTW to periodically wake the device, perform software buzz, and refresh the supervisory services. If LVI, HVI, or Brown Out is not required, then use of the CTW is not required.

16.4.1 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' before writing to the PM_MODE_CSR[2:0] register. 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.

16.4.2 Exiting Sleep Mode

Only PICU interrupts, comparator wakeup, supervisory interrupts, or resets wake up the system. At wakeup, the system activates all previously available domains from active mode template and begins executing the firmware in active mode.

16.5 Hibernate Mode

Hibernate mode consumes/dissipates the lowest power, and nearly all internal functions are disabled. There is no buzzing, and the external capacitors are permitted to discharge. The hibernate-regulator is always active to generate the keep-alive voltage (V_{pwrka}) used to retain the system state. See 15.3.3 Voltage Monitoring on page 141.

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 hibernate will put the system into sleep mode. This is important when there are chances of an accidental entry into hibernate mode and the watchdog is disabled.

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.

16.5.1 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' before writing to the PM_MODE_CSR[2:0] register. 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.

16.5.2 Exiting Hibernate Mode

Return from hibernate mode can occur only in response to a PICU or reset event. 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.

16.6 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.

16.6.1 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:

- Waking up the device from a low-power mode
- 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]}.

16.6.2 Fast Timewheel (FTW)

The fast timewheel (FTW) is a 100-kHz, 5-bit counter clocked by the ILO, which can also be used to wake the system. The FTW settings are programmable, using PM_TW_CFG0[4:0]; the counter automatically resets when the terminal count is reached. The FTW enables flexible, periodic wakeups of the CPU at a higher rate than the rate allowed using the CTW. To wake up on the FTW, write into register PM_TW_CFG2[0]. If the associated FTW interrupt is enabled using PM_TW_CFG2[1], an interrupt is generated each time the terminal count is reached.

16.7 Register List

Table 16-2. 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
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

17. 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. After 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 the startup code.

17.1 Features

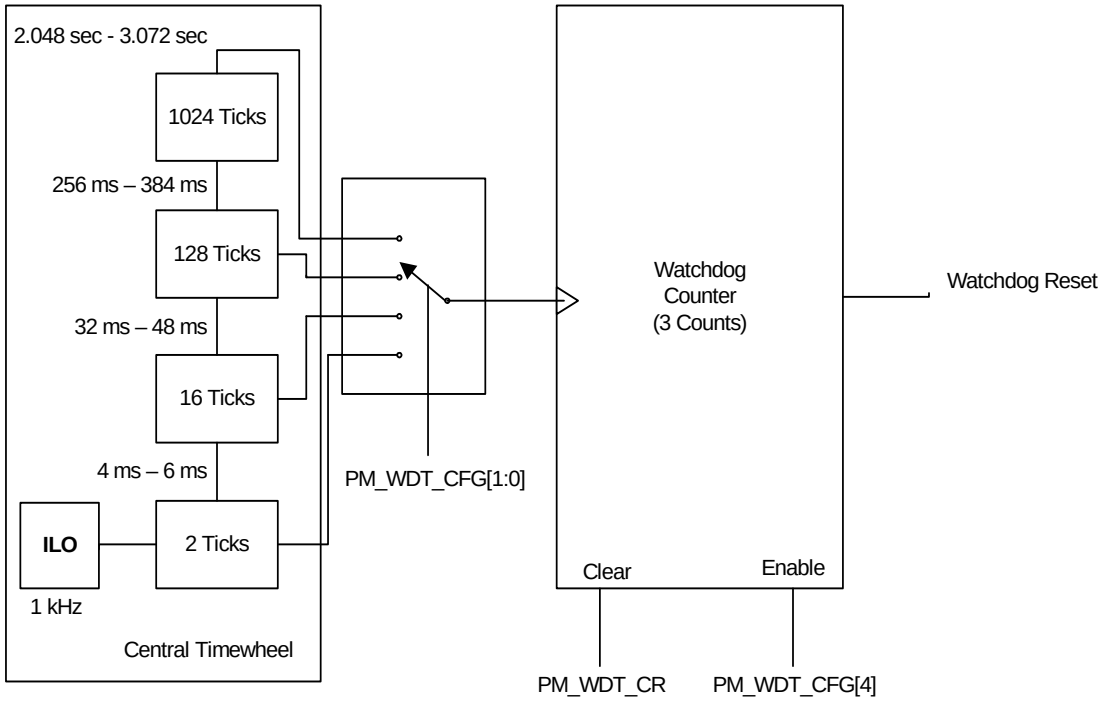
The WDT has the following features:

- Protection settings to prevent accidental corruption of the WDT
- Optionally-protected servicing (feeding) of the WDT
- A configurable low-power mode to reduce servicing requirements during sleep mode
- A status bit for the watchdog event that shows the status even after a watchdog reset

17.2 Block Diagram

Figure 17-1 is a block diagram of the WDT circuit.

Figure 17-1. Watchdog Timer Circuit



17.3 How It Works

The WDT circuit asserts a hardware reset to the device after a preprogrammed interval, unless it is periodically serviced in firmware. The system restarts if an unexpected execution path is taken through the code and the preprogrammed interval times out. It can also restart the system from the CPU halt state.

The WDT timeout is between two and three programmable tap periods, based on the free-running Central Timewheel. See the *PSoC® 3 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, program the software to service the WDT within two tap periods. Actual WDT timeouts may differ slightly from nominal, caused by inaccuracy of the ILO frequency.

17.3.1 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.

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

A status bit (RESET_SR0[3]) becomes 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.

17.3.2 Setting the WDT Time Period and Clearing the WDT

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 17-1 on page 149. The WDT counts until reaching three, 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.

17.3.3 Operation in Low-Power Modes

A configurable low-power mode of the WDT reduces servicing requirements during sleep mode. The register PM_WDT_CFG[6:5] governs the low-power mode for the WDT.

If the WDT is enabled, two bits define how the WDT behaves when the part enters Sleep/Idle/Hibtimers (low-power) mode. The default is 01; the system will automatically use the longest WDT interval when Sleep/Idle/Hibtimers mode is entered, so software is not burdened with waking just to feed the WDT. This is true regardless of the value programmed in the wdt_interval register. Upon wakeup, the interval will remain at the highest setting until the WDT is fed the first time. A feeding at this point will cause the interval to automatically return to the normal setting (value in wdt_interval). If this field is set to NOCHANGE ('00'), the system does not change the interval and does not feed the WDT when entering Sleep/Idle/Hibtimers mode. If DISABLED (wdt_lpmode=11), the WDT is turned off when Sleep/Idle/Hibtimers mode is entered and remains disabled until the first feeding by the user after active mode is reentered.

17.3.4 Watchdog Protection Settings

Using the MLOGIC_SEG_CR and MLOGIC_SEG_CFG0 registers, the WDT registers are protected from accidental corruption as follows:

- Clear, low-power enable, and Watchdog enable registers are protected as segment 0 as one-time system settings.
- The servicing of WDT clear is protected in segment 1 as a reconfigurable system setting.

See 20.3 Configuration Segment Protection on page 176.

17.4 Register List

Table 17-1. 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

18. Reset

PSoC® 3 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.

18.1 Reset Sources

The following is a description of reset sources. For power up supply monitoring, PSoC 3 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.

18.1.1 Power-on-Reset

Power-on-reset (POR) is provided primarily for a system reset at power-up. The IPOR holds the device in reset until all four voltages: Vdda, Vcca, Vddd, and Vccd, are to datasheet specification. The POR activates automatically at power-up and consists of:

- An imprecise POR (IPOR) – is used to keep the device in reset during initial device power-up until the POR can be activated
- A precision POR (PRES) – derived from a circuit calibrated for an accurate location of the POR trip point. The power on RESET clears all the reset status registers explained in 18.1.6 Identifying Reset Sources on page 152.

  A Vddx that goes up out of the PRES region must stay above the PRES region for at least 10 µs before it can drop back and reassert PRES. In other words, when deasserted, PRES cannot reassert for at least 10 µs.

18.1.2 Low-Voltage Reset and High-Voltage Reset

In addition to the POR reset for the voltage supplies described above, the device also supports device reset on a configurable low voltage, and a fixed high voltage condition. The reset due to the low-voltage condition is referred to as the Low-Voltage Interrupt (LVI) Reset. It is named so due to the fact that the low-voltage condition can be configured either as a reset source or an interrupt source. Two LVI resets are available, one for the Vddd supply (LVID reset) and the other for the Vdda supply (LVIA reset). The low-voltage value for the reset condition can be individually configured for each of the two voltage domains. When configured as a reset source, the device will be continuously reset as long as the Vdda/Vddd supply is below the set trip voltage point.

Similar to the low-voltage reset condition, the device also supports reset when the Vdda/Vddd supply goes above a fixed high voltage value of 5.75 V. This is referred to as the High-Voltage Interrupt (HVI) Reset. Unlike the LVI reset, the HVI reset can be configured only for either Vdda or Vddd supply at any time. The default selection is the Vdda supply.

See the Power Supply and Monitoring chapter on page 137 for details on the configuration of the LVI and HVI reset sources.

18.1.3 Watchdog Reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. Always set the WRES initialization code. This allows you to dynamically choose whether or not 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 WDT 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 WDT event occurred. See the Watchdog Timer chapter on page 149.

The RESET_SR0[3] status bit is set when a watchdog reset occurs. 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.

18.1.4 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.

18.1.5 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, as well as 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 do not 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 99 for 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. After XRES is deasserted, at least 10 µs must elapse before it can be reasserted.

18.1.6 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 device reset occurs due to one of the below mentioned sources.

18.1.6.1 Preservation of Reset Status

The device reset caused due to XRES, IPOR, PRES, LVI, and HVI sources clear the contents of the RESET_SR0 and RESET_SR1 registers. These sources are referred to as hard reset sources because they reset all the registers. The remaining reset sources, which include software reset, watchdog reset, and segment reset preserve the status of the RESET_SR0 and RESET_SR1 registers. For example, if an LVI reset and a software reset occur simultaneously, the LVI reset will clear the status bit corresponding to the software reset, making it impossible to detect a software reset condition. Also, the status bits corresponding to PRES, LVI, and HVI in RESET_SR0 and RESET_SR1 registers are meaningless because the respective reset conditions are hard resets, which clear all the register bits. The status bits corresponding to LVI and HVI will however be required when they are configured as interrupt sources instead of reset sources.

18.2 Reset Diagram

Figure 18-1 is a simplified logic diagram of the RESET module. Any active source of reset will make the system reset.

Figure 18-1. Logic Diagram of the RESET Module

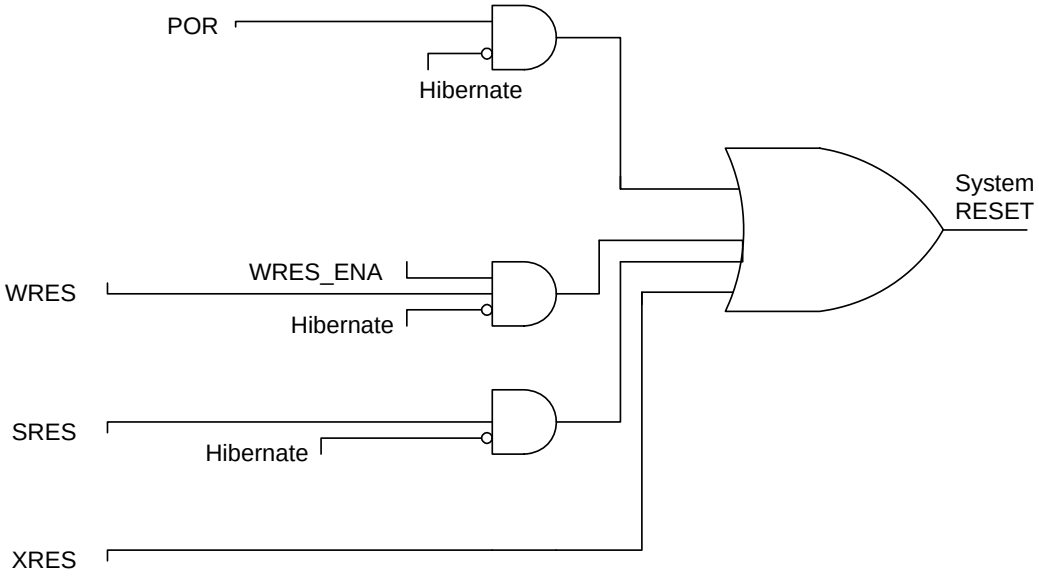
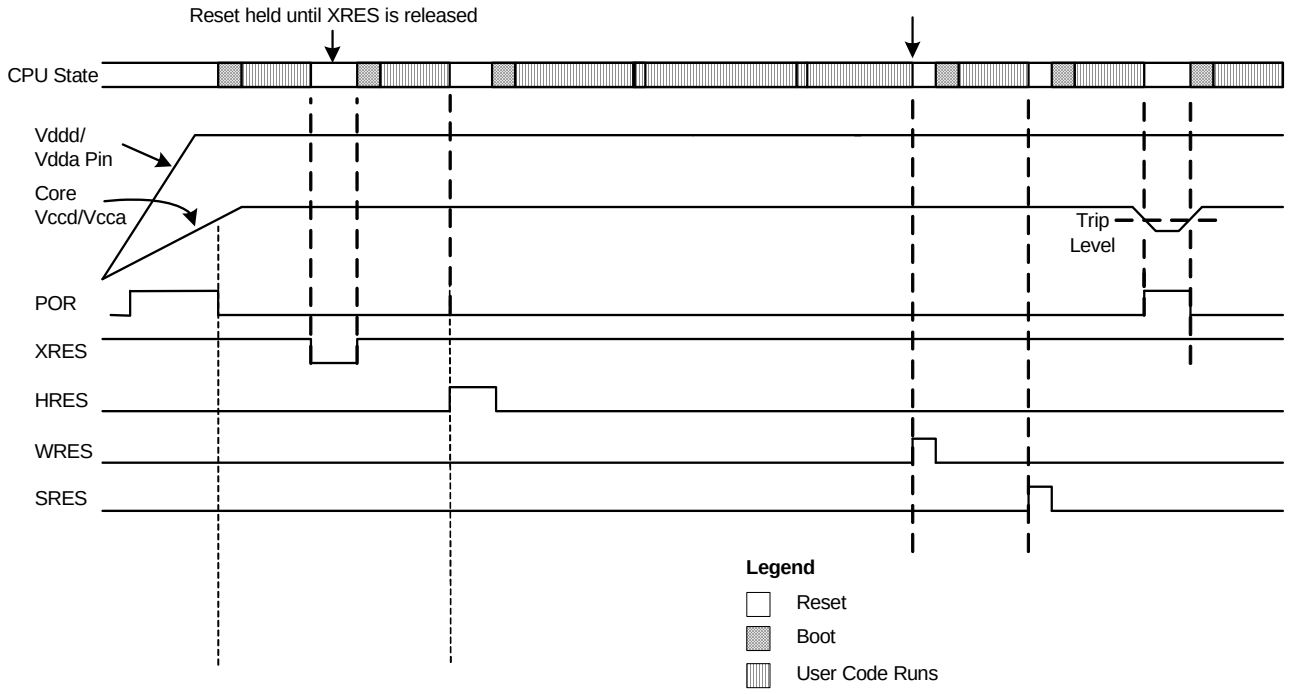


Figure 18-2 shows the operation of various RESETs with the change in Vdd/Vcc. The diagram also shows the functioning of RESETs in a normal power-up.

Figure 18-2. Resets Resulting from Various Reset Sources



18.3 Reset Summary

All reset sources and their triggers/effects are described in Table 18-1.

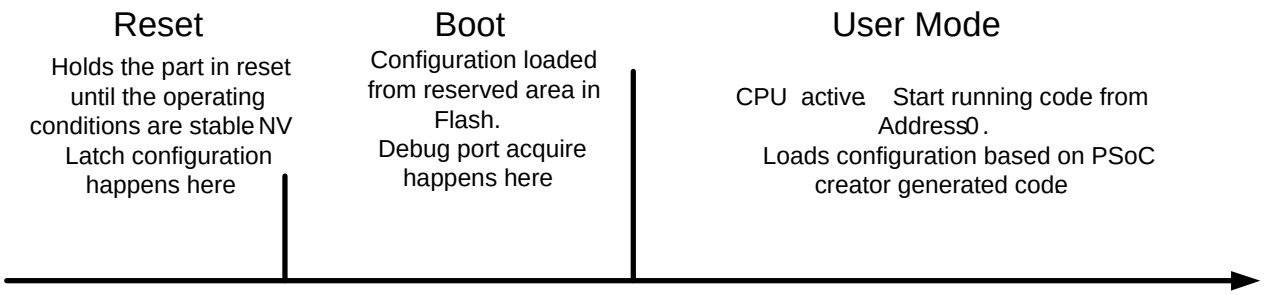
Table 18-1. Reset Sources, Triggers, and Effects

	POR	WRES	SRES	XRES_N
Trigger	V_{ccd} <1.6 V	WDT not written in time window	RESET_CR2[0] set	External XRES_N pin active
Enable by Default?	Yes	No	No	Yes (nonvolatile latch setting)
Block Power	50 µA	<1 µA	0	0
Sleep Mode Operation	Buzzed	Not in Hibernate	No	Yes

18.4 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® 3 life cycle consists of reset, boot, and user phases. Figure 18-3 gives a brief view of these phases.

Figure 18-3. Boot Process



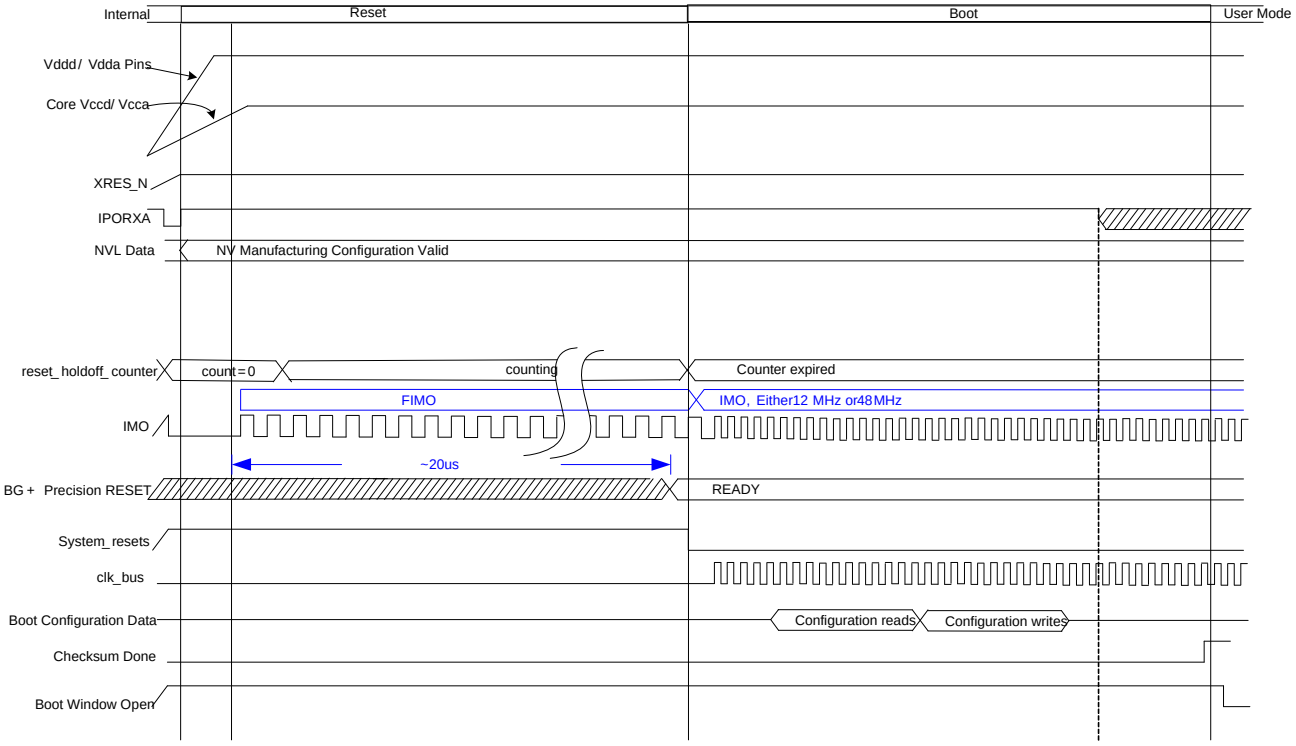
The process from supply voltage stabilization to user code entry is shown in Figure 18-4. After the voltage is high enough, the NVL data load is initiated. The NVL load takes care of loading configuration data stored in the NV latches. This configuration data controls the reset behavior of the device. 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 as well as setting the other Manufacturing Configuration data for the device. At this point, the device enters the reset state. The two types of NVL loads are explained in 18.4.1 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 bring the device out of the reset state.

The IMO clock is then started in a fast IMO (FIMO) mode, which is a faster startup version of the IMO. The reset hold-off counter continues to hold the device in reset until 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, the hold-off counter completes and the device is released from reset.

After releasing from reset, the IMO is switched to either 12 MHz or 48 MHz, the system bus clock is started and the boot cycle begins. Until now, the bus clock is fed from the FIMO, which has lesser accuracy compared to the IMO. After the reset is released, it moves into the IMO, which is more precise. The boot phase is explained in section 18.4.3 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 P1_0 and P1_1 implies a debug port acquire is being attempted which must trigger a debug port entry.

Figure 18-4. Power Up Reset Boot User Mode Cycle



In this phase, two types of NV latches are loaded to set reset states and trims in the device. The two types of the configuration, explained in sections 18.4.1 Manufacturing Configuration NV Latch and 18.4.1.1 Device Configuration NV Latch, occur simultaneously in the reset phase.

18.4.1 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 provide these trim values. An example of this 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. The NV latch's output is stable at approximately 1 V while the lowest operating voltage in the PSoC® 3 platform is 1.71 V.

18.4.1.1 Device Configuration NV Latch

Device configuration is similar to manufacturing configuration NV in that it occurs while the device is in reset; however, it differs in that customers select optional configuration settings not trim values for circuits. Manufacturing configuration and device configuration occur in parallel. One 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.

18.4.2 Boot Phase

Though many device settings are done using NV latch 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 159) and the boot process takes care of moving this data to the corresponding blocks. This loading of the configuration happens using the DMA and PHUB. A DMA channel fetches the configuration bytes from the flash and places them in the SRAM. The checksum block does a checksum to determine integrity. After the data is verified, it is then transferred using the DMA to the corresponding configuration register. If the checksum 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 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 to which manufacturing configuration data is written

When the boot process is complete, the device enters the user mode where the user code starts executing.

18.4.3 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().

18.5 Register List

Table 18-2. 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, PRES
RESET_SR2	Real-time Reset Status

19. 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 as well as 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.

19.1 Features

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

Supported by both GPIO and SIO pins:

- User programmable I/O state and drive mode on device reset
- Flexible drive modes
- Support level and edge interrupts on pin basis
- Slew rate control
- Supports CMOS and low voltage TTL input thresholds
- Separate port read and write registers
- Separate I/O supplies and voltages for up to four groups of I/O

Provided only on the GPIO pins:

- Supports LCD drive
- Supports CapSense
- Supports JTAG interface
- Analog input and output capability
- 8 mA sink and 4 mA source current
- Ports can be configured to support EMIF address and data

Provided only on SIO pins:

- Hot swap capability (5 V tolerance at any operating V_{dd})
- Single enable and differential input with programmable threshold
- Regulated output voltage level option
- Overvoltage tolerance up to 5.5 V
- Higher drive strength than GPIO
- 25 mA sink and 4 mA source current

USBIO features:

- USB 2.0 compliant I/O
- 25 mA source/24 mA sink current

19.2 Block Diagrams

Figure 19-1, Figure 19-2 on page 161, and Figure 19-3 on page 161 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, as well as the signals and register settings that control the main blocks.

Figure 19-1. GPIO Block Diagram

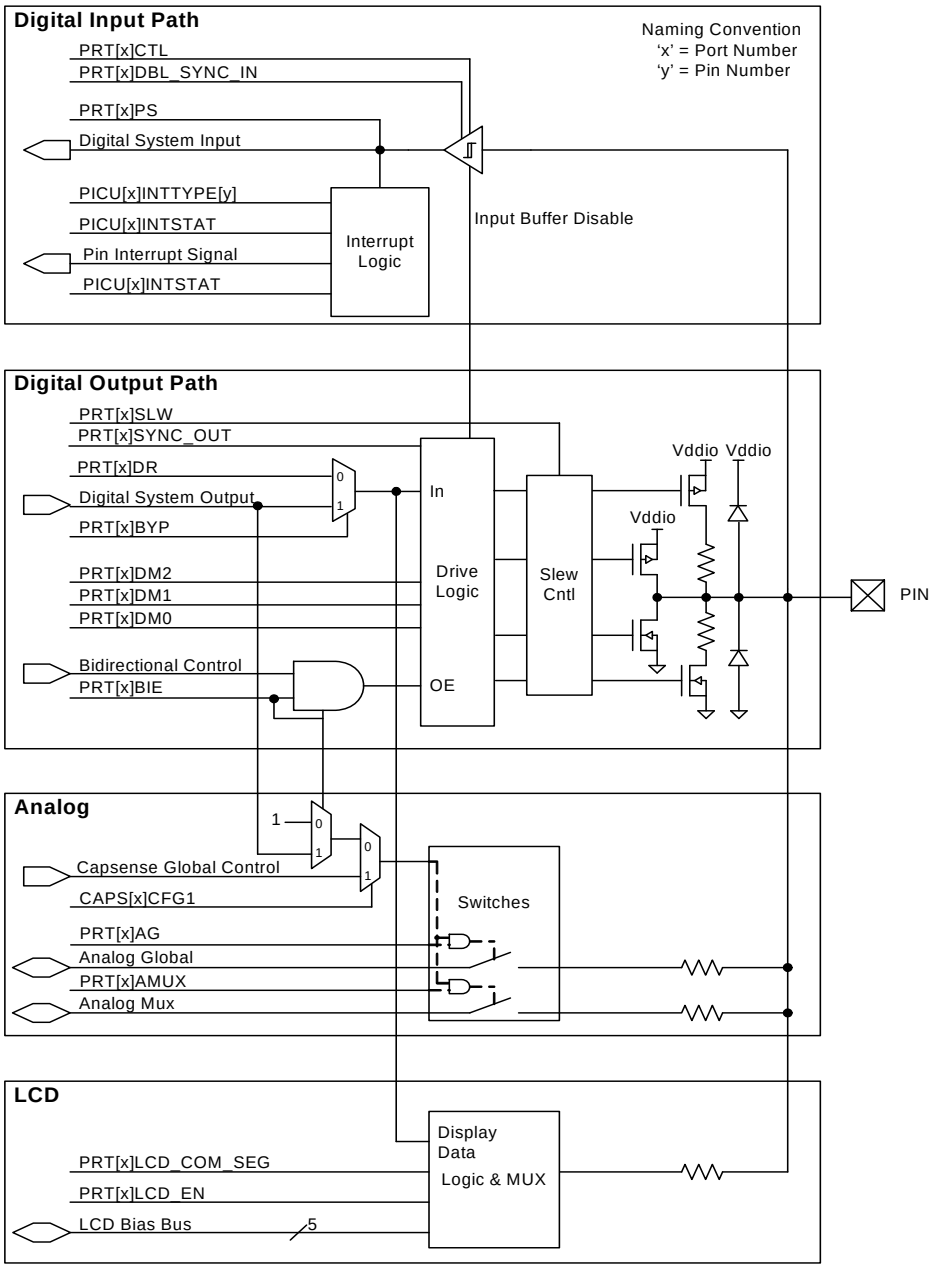


Figure 19-2. SIO Block Diagram

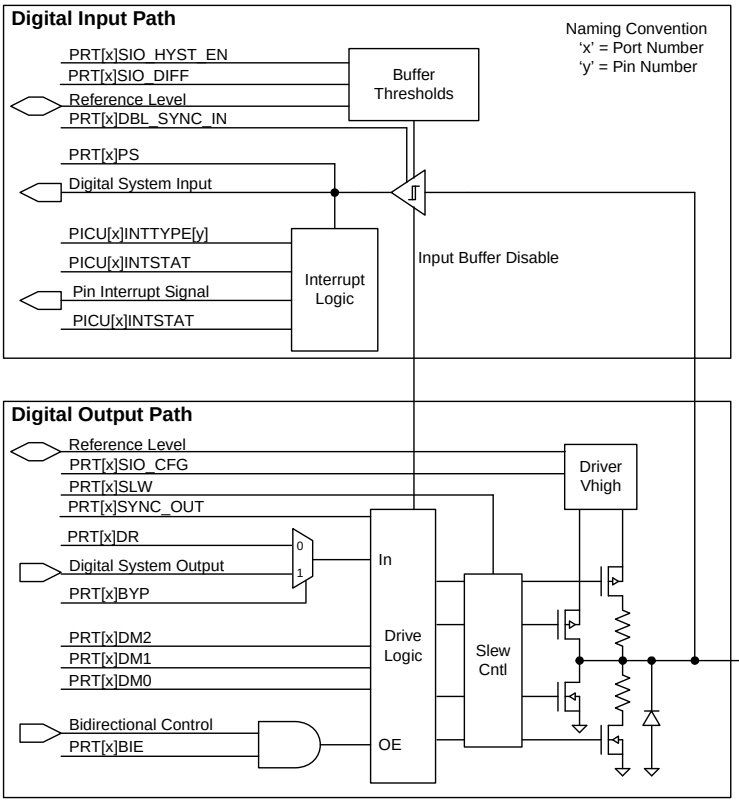
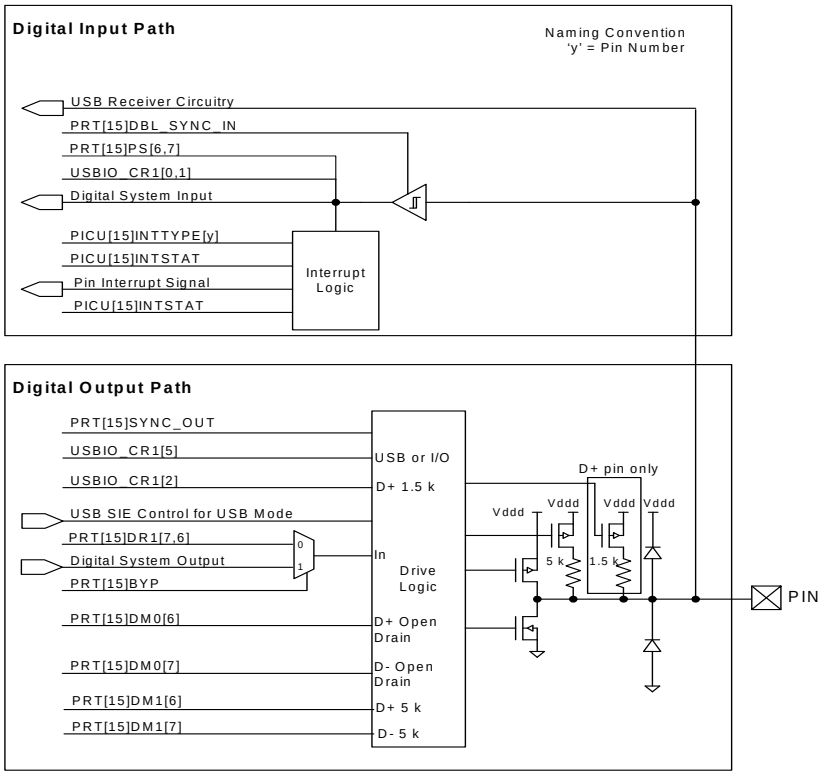


Figure 19-3. USBIO Block Diagram



19.3 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 and EMIF
- 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, crystal oscillator). 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.

19.3.1 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.

19.3.2 I/O Drive Modes

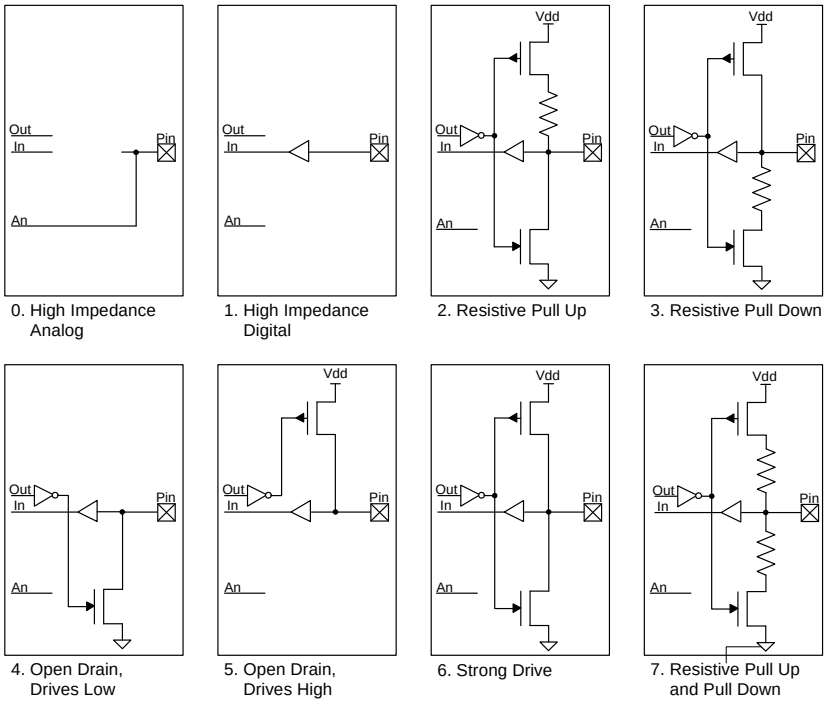
Each GPIO and SIO pin is individually configurable into one of the eight drive modes listed in Table 19-1 and shown in Figure 19-4, which depicts a simplified pin view based on each of the eight drive modes.

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. When the drive mode of a pin is changed, it is possible that the input buffer may be turned-off for a short period during the drive mode transition. Therefore, pin interrupts should be disabled while changing pin configuration.

Table 19-1. 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 19-4. I/O Drive Mode Diagram



The 'Out' connection is driven from either the Digital System (when the Digital Output terminal is connected) or the Data Register (when HW connection is disabled).
The 'In' connection drives the Pin State register, and the Digital System if the Digital Input terminal is enabled and connected.
The 'An' connection connects to the Analog System.

19.3.2.1 Drive Mode on Reset

The factory drive mode default is high impedance analog mode, which is appropriate for most designs. The Drive Mode on Reset feature allows the user to change the factory default to any of the four listed drive modes if the application requires faster configuration to low or high logic levels. The Reset drive mode is set at POR release. The Drive Mode on Reset setting is a port wide setting and is not set per pin. Each pin is individually configured during the device configuration step after POR release; this setting overwrites the reset drive mode. The Resistive Pull Up Drive Mode on Reset also sets the Port Data Register to 0xFF to ensure the port is pulled up; all other modes leave the Data Register 0x00.

- High impedance analog
- High impedance digital
- Resistive pull up
- Resistive pull down

See the Nonvolatile Latch chapter on page 99 for details.

19.3.2.2 High Impedance Analog

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

19.3.2.3 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.

19.3.2.4 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.

19.3.2.5 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.

19.3.2.6 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.

19.3.2.7 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.

19.3.3 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.

19.3.4 Digital I/O Controlled by Port Register

The Port Control registers (see Table 19-2 on page 164) 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}).

19.3.4.1 Port Configuration Registers

Table 19-2 lists port control registers.

Table 19-2. Functional Registers Accessed through Pin and Port Configuration Registers

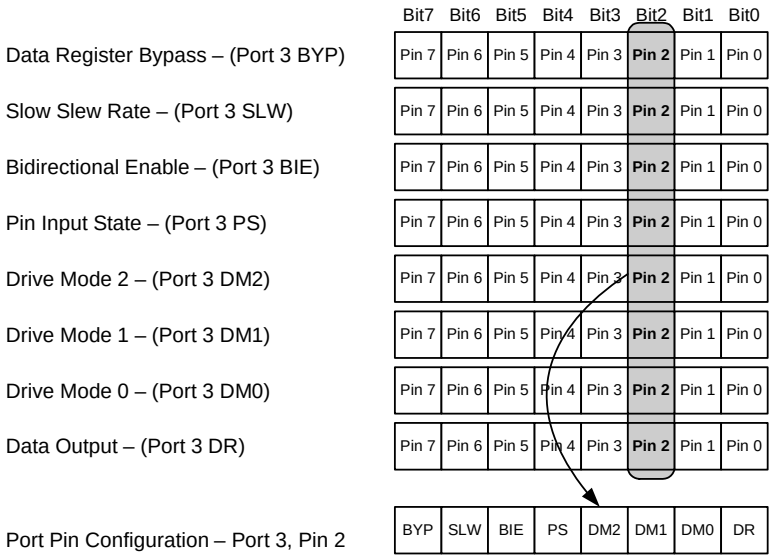
Address	Description
PRT[0..11]_BYP	A bit set in this register connects the corresponding port pin to the Digital System Interconnect (DSI), and disconnects it from the DR register.
PRT[0..11]_SLW	Each bit controls the output edge rate of the corresponding port pin – fast edge rate mode (Slew=0) or slow 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.

19.3.4.2 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 19-5 on page 165.

Figure 19-5 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 19-5. Effect of a Read of the Pin Configuration Register {PRT*_PC[4]}

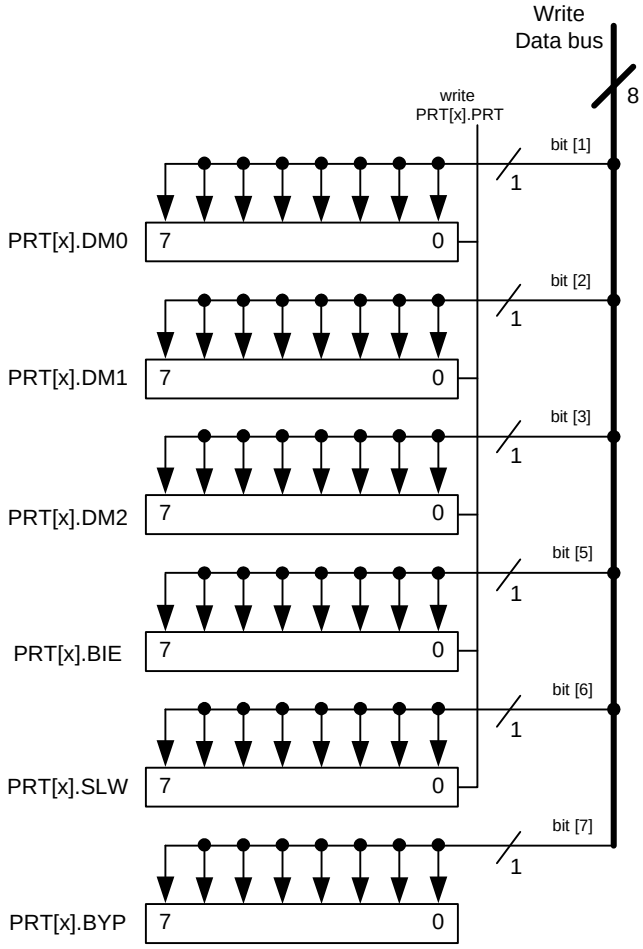


19.3.4.3 Port Wide Configuration Register Alias

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 19-6. {PRT*.PRT} Write Example



19.3.5 SFR to GPIO

All I/Os allow 8051 Special Function Register (SFR) direct read/write access to the data register and read access to the pin state register. This feature gives the user another method to read from and write to the ports.

I/O ports are linked to the CPU through the PHUB. In addition, all of the I/O ports are linked to the SFR bus. Each of the I/O ports supports two interfaces: the SFR bus and the PHUB bus. The SFR bus allows for the most efficient I/O access during normal operation, while the PHUB bus provides full control to all I/O registers.

As shown in Table 19-3, SFR registers contain three registers for each I/O port.

Table 19-3. SFR Registers

Address[a]	Description
SFR_USER_GPIOx	Sets the output data state for port x with respect to setting in SFR_USER_GPIOx_SEL register
SFR_USER_GPIOx_SEL	Sets output for each bit in port x register to corresponding pin in port x
SFR_USER_GPIRDx	Read only register; contains pin state value of port x

a. x is port number and includes ports 0-6, 12, and 15

Table 19-4 shows three examples illustrating results from setting selected bits in the SFR register.

Table 19-4. SFR Register Bit Examples

Example	Settings	Result
For Port 0, Bit 0	SFR_USER_GPIO0_SEL, bit0=1; SFR_USER_GPIO0, bit 0=1	Port0 [0] =1
For Port 0, Bit 0	SFR_USER_GPIO0_SEL, bit0=1; SFR_USER_GPIO0, bit 0=0	Port0 [0] =0
For Port 0, Bit 0	SFR_USER_GPIO0_SEL, bit0=0; SFR_USER_GPIO0, bit 0=1	SFR does not set corresponding GPIO

For more information, see the 8051 Core chapter on page 37.

19.3.6 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.

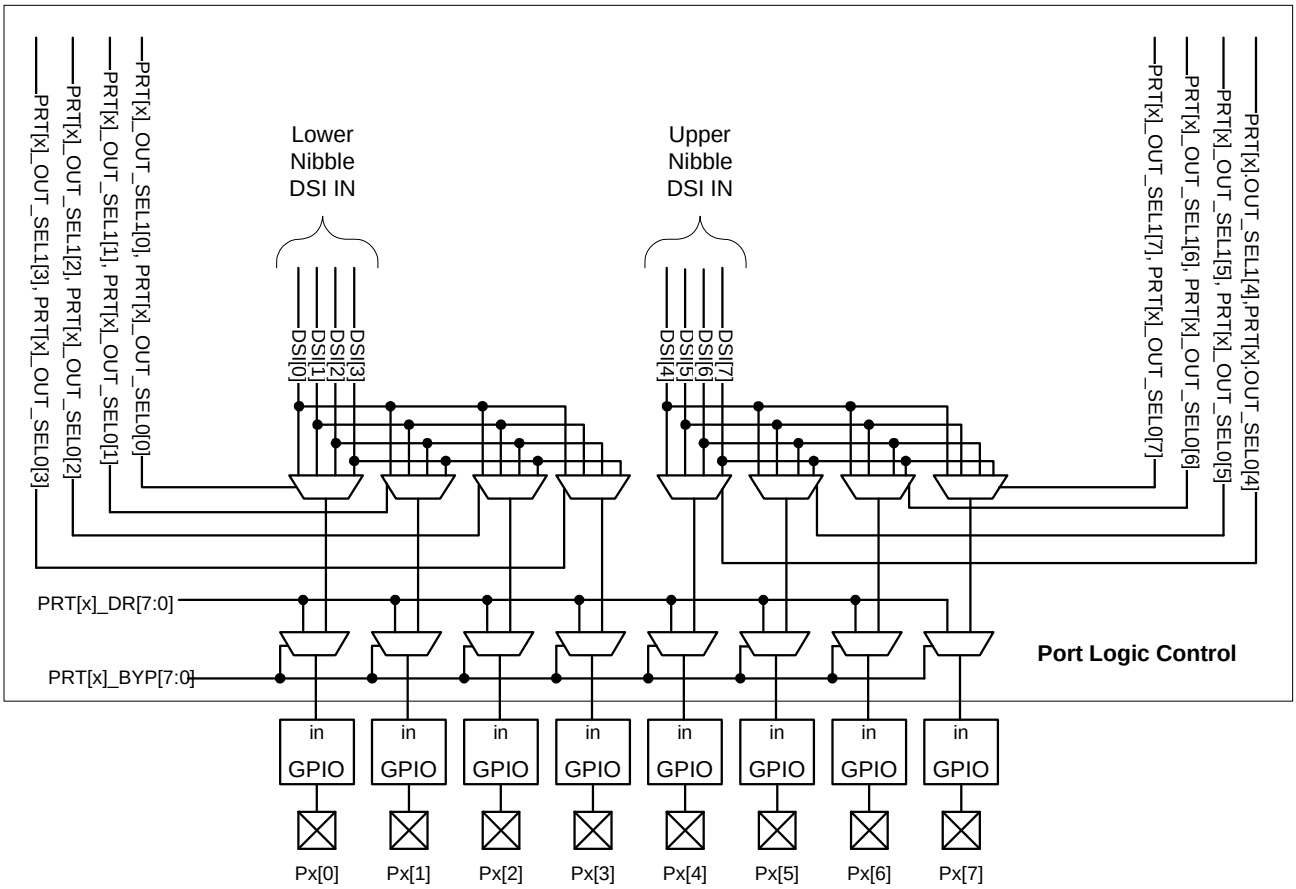
19.3.6.1 DSI Output

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 19-7 on page 167.

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

Figure 19-7. Digital System Input to Pad Selection



19.3.6.2 DSI Input

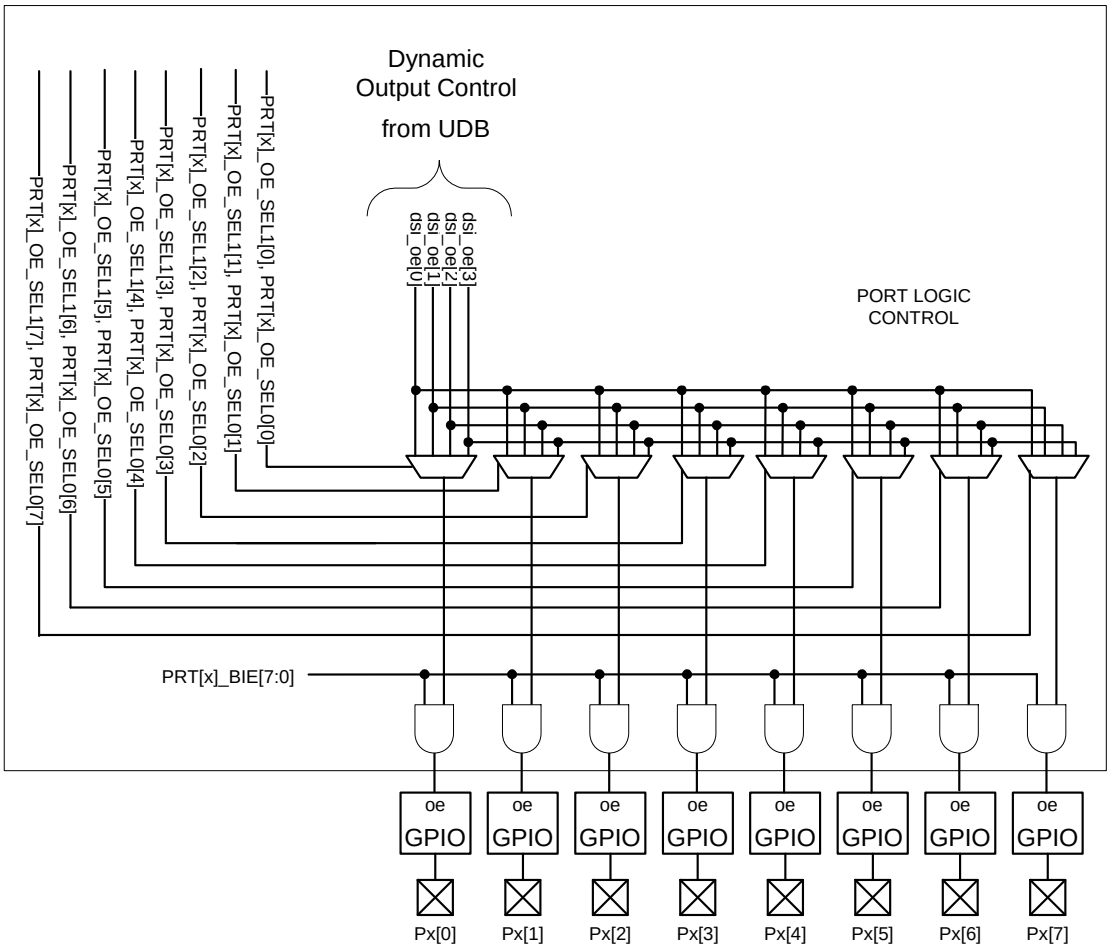
The port pin input is directly connected to the UDB array through DSI for routing the input to various internal peripheral blocks. The control for these port inputs are at the DSI inputs. See the Universal Digital Blocks (UDBs) chapter on page 183 for port-to-DSI connections.

19.3.6.3 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 19-8 on page 168.

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

Figure 19-8. Mapping of DSI Control Signal to Port Pin Output Enable



19.3.7 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.

See the Analog Routing chapter on page 327 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.

19.3.8 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 into a High Z output mode, allowing the LCD drivers to control the pin state.

19.3.9 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 365 for more information.

19.3.10 External Memory Interface (EMIF)

The EMIF uses the port interface and the UDB to connect to external memory. When in EMIF mode, the ports directly pass to the pads the address and data out from the PHUB. Data reads from the EMIF pass through the port to the PHUB. See the EMIF chapter on page 111 for more information.

19.3.11 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.

19.3.11.1 Regulated Output Level

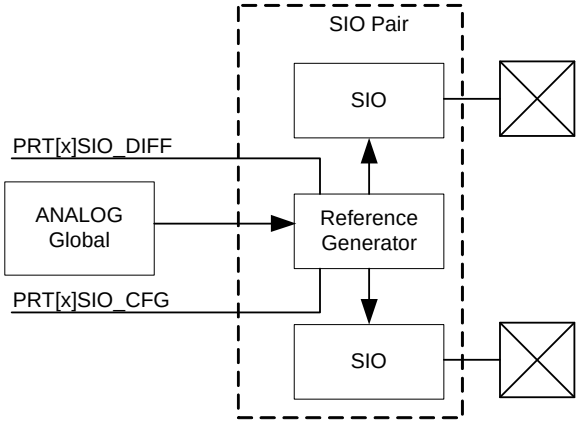
SIO port pins support the ability to provide a regulated high output level. This can be useful for interfacing to external signals with voltages lower than the SIO V_{ddio}. This regulated output sets the V_{oh} for the SIO pair. The SIO are grouped into pairs. Each pair shares the same reference generator, 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 19-5. 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 19-9. SIO Configuration Diagram



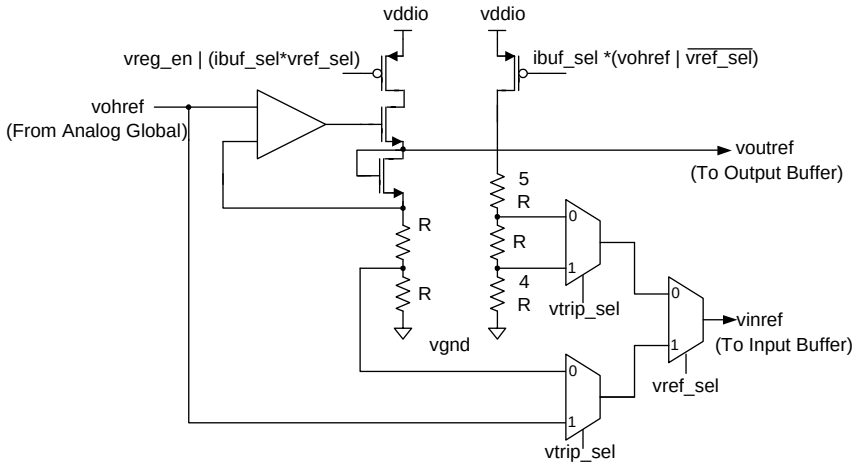
19.3.11.2 Adjustable Input Level

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 19-6. 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 19-10. SIO Reference Voltage



19.3.11.3 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.

19.3.12 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
- CAN
- USB

Special functions and peripherals such as I^2C, crystal oscillators, USB, XRES_N, JTAG TAP, 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 either the dedicated XRES_N pin or the P1[2] GPIO (because the XRES_N pin is not bonded on the 48-pin package). The IEEE 1149.1 JTAG TAP five pin interface may be enabled on the P1[0:1,3:5] pins.

Serial wire debug is supported over the USBIO pins (P15[6:7]) or the same pins as TMS / TCK (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 except the 48-pin package.

Table 19-7. Fixed Pin Assignments

Function	Signal Name	Pad #	Pad Name	Pad Type	TQPF 100	QFN 68	SSOP 48	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	42	SIO pair on Vio3
	SDA	62	P12[1]	SIO	54	39	43	
MHz ECO	Xo	49	P15[0]	GPIO / Xtal	42	27	39	
	Xi	50	P15[1]	GPIO / Xtal	43	28	40	
32 kHz ECO	Xo	62	P15[2]	GPIO / Xtal	55	40	44	
	Xi	63	P15[3]	GPIO / Xtal	56	41	45	
FS USB	D+	39	P15[6]	USBIO	35	22	34	
	D-	40	P15[7]	USBIO	36	23	35	
XRES_N	XRES_N	19	XRES_N	XRES_N	15	10	-	Fixed function XRES_N / TSTRST_N pin
		26	P1[2]	GPIO	22	13	27	XRES_N / TSTRST_N option for 48-pin package
IEEE 1149.1 JTAG TAP	TMS	24	P1[0]	GPIO	20	11	25	
	TCK	25	P1[1]	GPIO	21	12	26	
	TDO	26	P1[3]	GPIO	23	14	28	
	TDI	28	P1[4]	GPIO	24	15	29	
	nTRST	29	P1[5]	GPIO	25	16	30	
Serial Wire Debug	SWDIO	24	P1[0]	GPIO	20	11	25	SWD on GPIO pins option
		39	P15[6]	USBIO	35	22	34	SWD on USB pins option
	SWDCK	25	P1[1]	GPIO	21	12	26	SWD on GPIO pins option
		40	P15[7]	USBIO	36	23	35	SWD on USB pins option
	SWO	27	P1[3]	GPIO	23	14	28	

Table 19-7. Fixed Pin Assignments *(continued)*

Function	Signal Name	Pad #	Pad Name	Pad Type	TQPF 100	QFN 68	SSOP 48	Comment
VIDAC High Current Output	Abuffer0L	82	P0[6]	GPIO	78	55	10	
	Abuffer1L	83	P0[7]	GPIO	79	56	11	
	Abuffer0R	51	P3[0]	GPIO	44	29	-	
	Abuffer1R	52	P3[1]	GPIO	45	30	-	
External Vref	Extref0	78	P0[3]	GPIO	74	51	6	
	Extref1	53	P3[2]	GPIO	46	31	-	
Analog Linear Output Buffer	Abuf0+	77	P0[2]	GPIO	73	50	5	
	Abuf0-	78	P0[3]	GPIO	74	51	6	
	Abuf0out	76	P0[1]	GPIO	72	49	4	
	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	8	
	Abuf2-	81	P0[5]	GPIO	77	54	9	
	Abuf2out	75	P0[0]	GPIO	71	48	3	
	Abuf3-	53	P3[2]	GPIO	46	31	-	
	Abuf3+	54	P3[3]	GPIO	47	32	-	
	Abuf3out	59	P3[7]	GPIO	52	37	-	

19.3.13 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.

19.3.14 Power Up I/O Configuration

By default, all I/Os power up in a known state, either driving a 0, driving a 1, or set to High Z. Input buffers are disabled during power up. The value set in the nonvolatile (NV) latches determines the value driving each port.

A pair of NV latches is associated with each I/O port; these latches serve two functions:

- Latch values configure the pins on a port-wide basis during power up.
- Latch values load reset values for the drive mode and data registers to correctly configure the port, when IPOR_disabled is deasserted.

See the Nonvolatile Latch chapter on page 99 for more information.

If the NVLs are set to 0x00 for the port, by default all I/Os reset to the High Impedance Analog state but are reprogrammable on a port-by-port basis. They can be reset as High Impedance Analog, Pull Down, or Pull Up, based on the requirements of the application.

19.3.15 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 1.8 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.

19.3.16 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 provide different I/O interface levels for different pins on the device. See the datasheet to determine V_{ddio} capability for a given device and pin.

SIO port pins support an additional regulated high output capability, as discussed in 19.3.11.2 Adjustable Input Level.

19.3.17 Sleep Mode Behavior

The GPIO/SIO pad will maintain the current pin state during sleep modes. Port pin interrupts remain active in all sleep modes, allowing the PSoC device to wake from an externally generated interrupt.

19.3.18 Low-power Behavior

In all low-power modes, I/O pins retain their states until the part is awakened and changed or reset. To awaken the part, use a pin interrupt, because the port interrupt logic continues to function in all low-power modes.

19.4 Port Interrupt Controller Unit

This section describes the functions of the port interrupt controller unit (PICU) for PSoC I/O.

19.4.1 Features

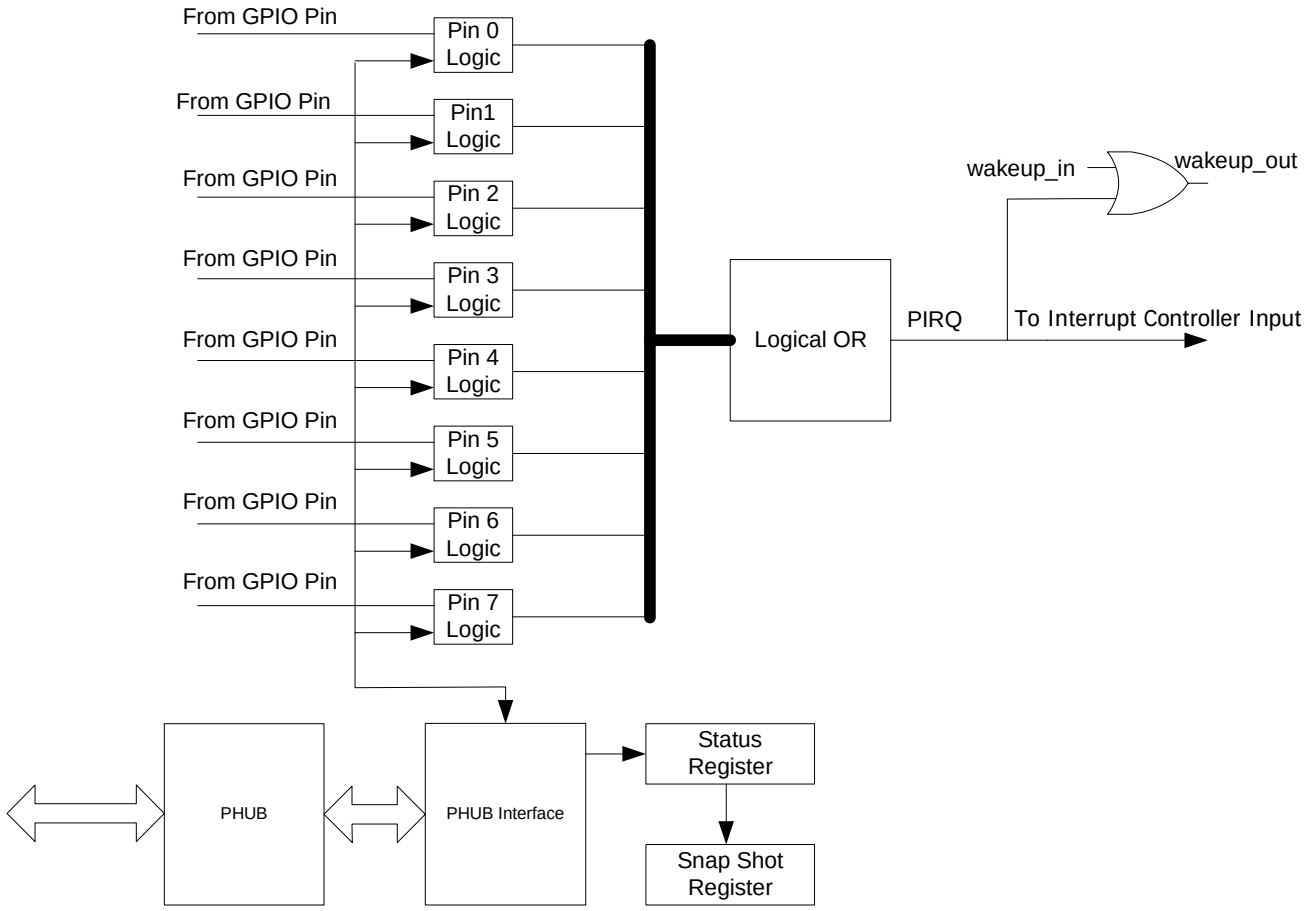
The features of the PICU are as follows:

- 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

19.4.2 Interrupt Controller Block Diagram

Figure 19-11 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 19-11. PICU Block Diagram



19.4.3 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.
4. Each PICU has a wakeup_in input and a wakeup_out output signal. The wakeup_in signal in a PICU is ORed together with other pin interrupts to generate a wakeup_out signal, as shown in Figure 19-11.
5. All of the PICUs are daisy chained together to generate a final wakeup signal that goes to the power manager.

19.5 Register Summary

Registers shown in Table 19-8 are associated with a single I/O port and are specific to both the GPIO and SIO ports.

Table 19-8. 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 19-9 shows registers specific to a GPIO port.

Table 19-9. GPIO Registers

Address	Description
{PRT*_CTL}	Port-wide configuration register. This contains the portEmifCfg[2:0] and 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 19-10 shows registers specific to an SIO port.

Table 19-10. 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 19-11 involve DSI bit selection. These registers are associated with all I/O ports and are located within the port logic.

Table 19-11. 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 19-12 shows the register associated with the PICU.

Table 19-12. 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 on currently posted interrupts – 1 register for each PICU
{PICU*_SNAP}	The Port Snapshot register provides information on the state of the input pins at the most recent read to the status (INTSTAT) register – 1 register for each PICU

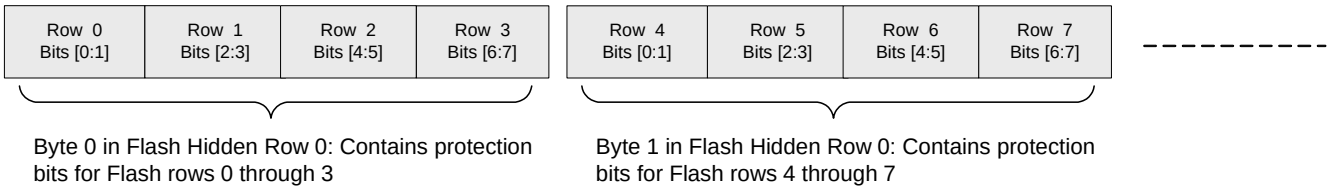
20. Flash, Configuration Protection

PSoC® 3 devices offer a host of flash and configuration 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 usage cases.

20.1 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 3 device that has protection options is the flash; there are no EEPROM and NV latch protection options. Flash memory in PSoC 3 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 3 architecture offer 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 20-1). See the Flash Program Memory chapter on page 107 to learn more about flash memory organization in PSoC 3 devices.

Figure 20-1. 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 are 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 427 for erasing and programming flash. Table 20-1 shows the protection modes.

Table 20-1. 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 20-1, four flash protection levels are available for every row of flash in a device. A customer 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 on 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.

20.2 Device Security

The objective of device security is to prevent the PSoC 3 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, you 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 427 for information about writing to the Write Once (WO) nonvolatile latch.

20.3 Configuration Segment Protection

Part of the PSoC platform’s value to customers is its ability to change the functionality of the device in real time. Changing the functionality can be as simple as enabling an external crystal or as dramatic as changing the functionality of UDBs from timers to CRC generators. Based on the application needs, the customer may also want to protect certain Configuration registers.

Not all configuration registers need the same level of security and protection. Hence, the configuration registers are grouped into four segments, with registers assigned to a segment based on the presumed application use cases. The registers under each of the four segments are listed in Table 20-3 to Table 20-6. The device registers that are not listed in these tables do not have any segment protection. This is to ensure that the protection logic is supported only on important registers, thereby saving chip area where the protection logic is not required.

Segment 0. One time system settings. This segment has system registers that are configured only once during program execution. The registers in this segment come under the following broad categories:

- Power System
- Reset
- Watchdog
- Internal low speed oscillator (ILO)

Segment 1. Reconfigurable system settings. This segment has registers that can be reconfigured during program execution. The registers in this segment come under the following broad categories:

- LVI Detect
- Voltage regulators
- Power Manager
- Wakeup Sources

- Boost Converter

Segment 2. UDB array configuration registers.

- All UDB array configuration registers, such as the clock selection and datapath input/output multiplexer selection, come under this segment.

Segment 3. Analog interface (Registers related to analog interface configuration).

It must be noted that Segment 0 registers can be configured either as the one time configurable or reconfigurable type. The same applies to Segment 1 and Segment 2 registers as well. But as a best practice, it is advisable to set Segment 0 registers as one time configurable. The settings for the rest of the segments depend on application requirements. To find out the segment to which a register is allocated, see the segment field for the register in the *PSoC® 3 Registers TRM (Technical Reference Manual)*.

Write access to the Configuration registers in various segments is enabled using the Segment Configuration register (MLOGIC_SEG_CFG0). Write access to the Segment Configuration register (MLOGIC_SEG_CFG0) is enabled using the Segment Control register (MLOGIC_SEG_CR).

20.3.1 Locking/Unlocking Segment Configuration Register

The 8-bit Segment Control register (MLOGIC_SEG_CR) is used to control write access to the Segment Configuration register (MLOGIC_SEG_CFG0) bits. By default, write access to the Segment Configuration register is disabled. Attempted writes will appear to execute normally, but the contents of the register will remain unchanged.

Segment configuration write access is enabled by writing 0xB5 to the Segment Control register and is disabled by writing 0xB4 to the Segment Control register. Upon device reset, the Segment Control register resets to the locked state and disables write to the Segment Configuration register.

When illegal values (values other then 0xB4 and 0xB5) are written to the Segment Control register, it causes a device reset and is indicated by the Segment reset (SEGRS) bit in Reset Status (RESET_SR1) register. The segment reset bit remains set until cleared by the user or POR.

20.3.2 Locking and Protecting Segments

The 8-bit Segment Configuration register (MLOGIC_SEG_CFG0) holds a pair of bits for each segment (Segment 0 to Segment 3) that are used to regulate access to the Configuration registers in that segment. The pair consists of one protect bit and one lock bit; these bits operate independently of each other.

Protect Bit. The segment protect (LOCK_PROTECT_x) bit controls the ability to write the segment's lock bit.

If the segment protect bit is '0', the segment's lock bit can be written as a '0' or '1' at anytime. If the protect bit is '1', the segment's lock bit cannot be modified.

The segment protect (LOCK_PROTECT_x) bit is a write-to1 once bit. It cannot change from a '1' to a '0' except as a result of a hardware reset, such as a POR or XRES_N. For one time configuration of a segment, it must be locked and protected after configuration.

Lock Bit. The segment lock (LOCK_x) bit controls the write access to the Configuration registers in the segment. Setting the LOCK_x bit prevents write access to the Control registers; clearing the lock bit allows a write.

For dynamic configuration of a segment, it must not be protected and can be locked after every configuration.

Table 20-2 describes the behavior for different protect and lock bit settings.

Table 20-2. Protect and Lock Bit Settings

Protect/Lock Bits	Description
00b	The Configuration registers are not protected and not locked. They can be written at anytime.
01b	The Configuration registers are not protected but locked. This is used to temporarily lock the configuration and is used in the case of dynamic reconfiguration.
10b	The Configuration register are protected and not locked. They can be written at anytime.
11b	The Configuration registers are protected and locked. This is used for one time configuration.

Table 20-3. Segment 0: One Time System Settings

Category	Register Names	PHUB Address
Reset	RESET_CR3	0x46F7
	RESET_CR4	0x46F8
	RESET_CR5	0x47F9
	RESET_TR	0x46FB
	RESET_IPOR_CR0	0x46F0
	RESET_IPOR_CR1	0x46F1
	RESET_IPOR_CR2	0x46F2
	RESET_IPOR_CR3	0x46F3
Power System	PWRSYS_HIB_TR0	0x4680
	PWRSYS_HIB_TR1	0x4681
	PWRSYS_I2C_TR	0x4682
	PWRSYS_SLP_TR	0x4683
	PWRSYS_BUZZ_TR	0x4684
	PWRSYS_WAKE_TR0	0x4685
	PWRSYS_WAKE_TR1	0x4686
	PWRSYS_BREF_TR	0x4687
	PWRSYS_BG_TR	0x4688
	PWRSYS_WAKE_TR2	0x4689
	PWRSYS_WAKE_TR3	0x468a
	PWRSYS_CR0	0x4330
ILO	ILO_TR0	0x4690
	ILO_TR1	0x4691
	SLOWCLK_ILO_CR0	0x4300
Watchdog	PM_WDT_CFG	0x4383

Table 20-4. Segment 1: Reconfigurable System Settings

Category	Register Names	PHUB Address
LVI Detect	RESET_CR0	0x46F4
	RESET_CR1	0x46F5
	RESET_CR2	0x46F6
Volt Regulators	PWRSYS_CR1	0x4331
Power Manager	PM_TW_CFG0	0x4380
	PM_TW_CFG1	0x4381
	PM_TW_CFG2	0x4382
	PM_WDT_CR	0x4384
	PM_MODE_CFG0	0x4391
	PM_MODE_CFG1	0x4392
	PM_MODE_CSR	0x4393
Wakeup Sources	PM_WAKEUP_CFG0	0x4398
	PM_WAKEUP_CFG1	0x4399
Boost	BOOST_CR0	0x4320
	BOOST_CR1	0x4321
	BOOST_CR2	0x4322
	BOOST_CR3	0x4323
Fast Clock	FASTCLK_*	0x4200-0x42FF
	IMO_*	0x46A0-0x46A7
	XMHZ_TR	0x46A8
FLASH LPM	CACHE_CR1	0x4801

Table 20-5. Segment 2: UDB Array

Category	Register Names	PHUB Address
UDB Config	UCFG_*	0x10000-0x150FF

Table 20-6. Segment 3: Analog Interface

Category	Register Names	PHUB Address
Analog Interface Routing and Configuration Registers		0x5800-0x5FFF
Analog Interface Trim Registers		0x4600-0x467F

20.3.3 Example

The device peripherals are enabled/disabled by the PM_ACT_CFG* registers in Active mode. These registers are mapped in Segment1. The following steps explain the procedure to configure these registers and then lock the configuration information so that runaway code does not overwrite the values.

1. Write 0xB5 to the Segment Control register (MLOGIC_SEG_CR) to enable the write access to the Segment Configuration register.
2. Clear the lock bit for Segment 1 to get write access to the Configuration registers in Segment 1. This is done by clearing the lock bit corresponding to Segment 1, which is MLOGIC_SEG_CFG0[2]. Here, it is assumed that the protect bit for this segment, MLOGIC_SEG_CFG0[3], is not set. If the protect bit has been set by the user, the lock bit cannot be modified, other than by a device reset.
3. Write to the Active Power Mode Template registers (PM_ACT_CFG*) to enable/disable the required peripherals.
4. Set the lock bit (MLOGIC_SEG_CFG0[2]) and clear the protect bit (MLOGIC_SEG_CFG0[3]) for Segment 1 in the Segment Configuration register (MLOGIC_SEG_CFG0).
5. Write 0xB4 to the Segment Control register to disable the write access to the Segment Configuration register.

20.4 Frequently Asked Questions About 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. See the Nonvolatile Memory Programming chapter on page 427 to learn more about flash erase/program commands.

Question 3. Is it possible to reprogram a flash memory that is 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 instance, 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 3 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. Entering debug mode while using JTAG boundary scan is also not possible. This prevents external access to registers and nonvolatile memory. See Device Security on page 176 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 3 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® 3 architecture provides 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.

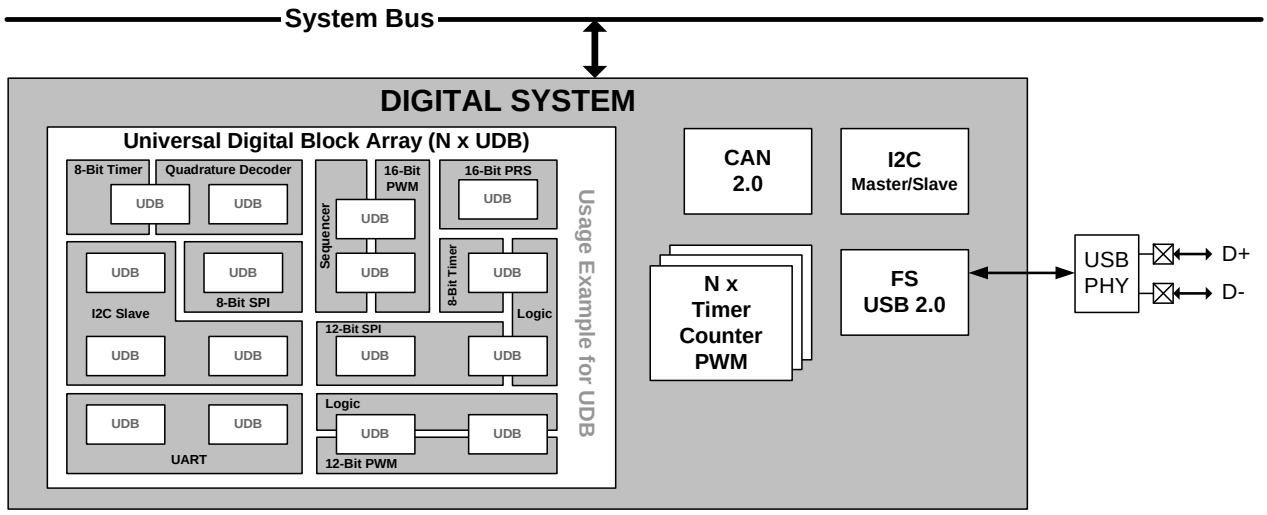
In addition to the flexibility of the UDB array, PSoC devices provide configurable digital blocks targeted at specific functions. These blocks can include 16-bit timer/counter/PWM blocks, I^2C slave/master/multi-master, Full Speed USB, and CAN 2.0b. See the device datasheet for a list of available specific function digital blocks.

This section encompasses the following chapters:

- Universal Digital Blocks (UDBs) chapter on page 183
- UDB Array and Digital System Interconnect chapter on page 225
- Controller Area Network (CAN) chapter on page 233
- USB chapter on page 249
- Timer, Counter, and PWM chapter on page 265
- I^2C chapter on page 281
- Digital Filter Block (DFB) chapter on page 295

Top Level Architecture

Digital System Block Diagram



21. Universal Digital Blocks (UDBs)

This chapter shows how the PSoC® 3universal 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.

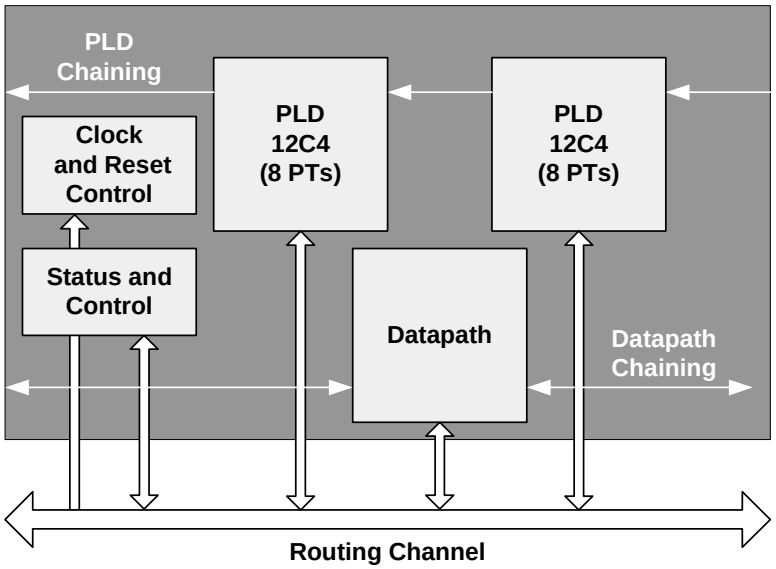
21.1 Features

- 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
- PSoC 3 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

21.2 Block Diagram

Figure 21-1 on page 184 illustrates the UDB as a construct containing a pair of basic PLD logic blocks, a datapath, and control, status, clock and reset functions.

Figure 21-1. UDB Block Diagram



21.3 How It Works

The major components of a UDB are:

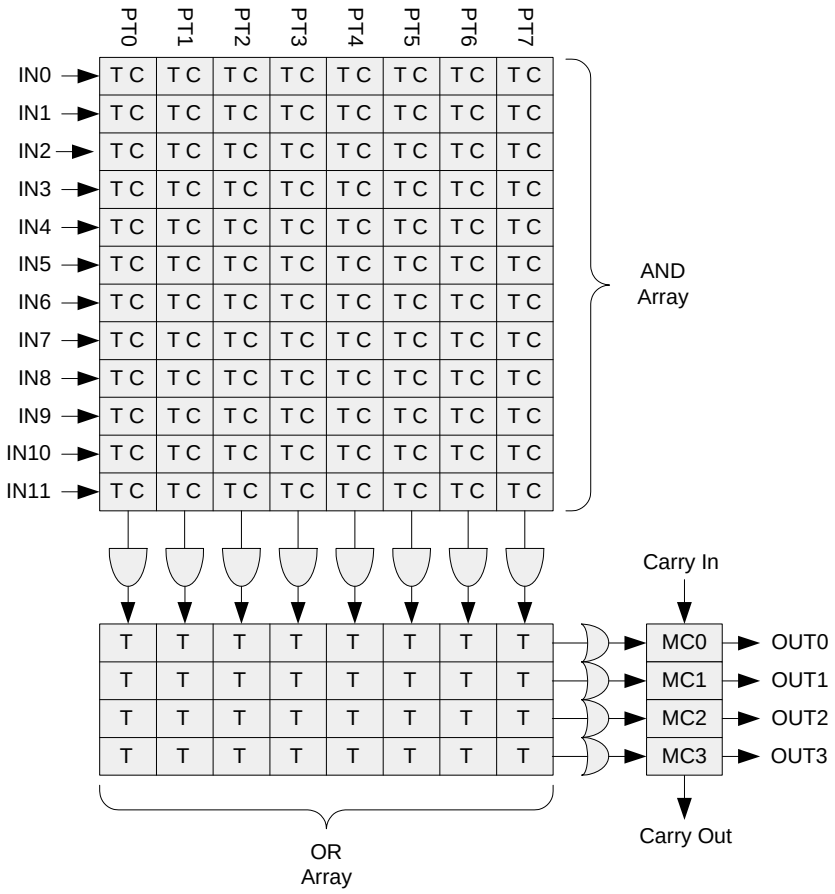
- **PLDs (2)** – These blocks take inputs from the routing channel and form registered or combinational sum-of-products logic to implement state machines, control for datapath operations, conditioning inputs, and driving outputs.
- **Datapath** – This block contains a dynamically programmable ALU, four 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 225.
- **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.

21.3.1 PLDs

There are two “12C4” PLDs in each UDB. PLD blocks, shown in Figure 21-2 on page 185, can be used to implement state machines, perform input or output data conditioning, and to create lookup tables (LUTs). PLDs may also be configured to perform arithmetic functions, sequence the datapath, and generate status. General purpose RTL can be synthesized and mapped to the PLD blocks. This section presents an overview of the PLD design.

A PLD has 12 inputs which feed across eight product terms (PT) in the AND array. In a given product term, the true (T) or complement (C) of the input can be selected. The 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.

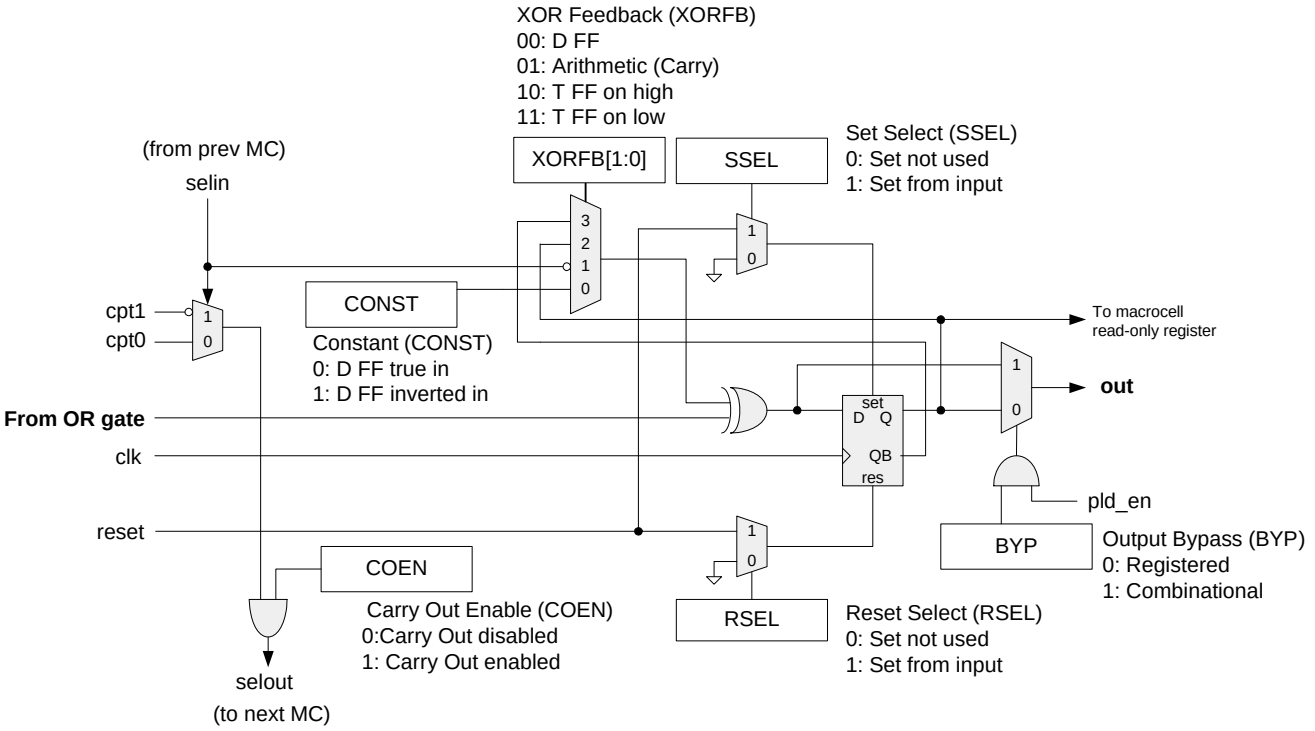
Figure 21-2. PLD 12C4 Structure



21.3.1.1 PLD Macrocells

The macrocell architecture is shown in Figure 21-3 on page 186. 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.

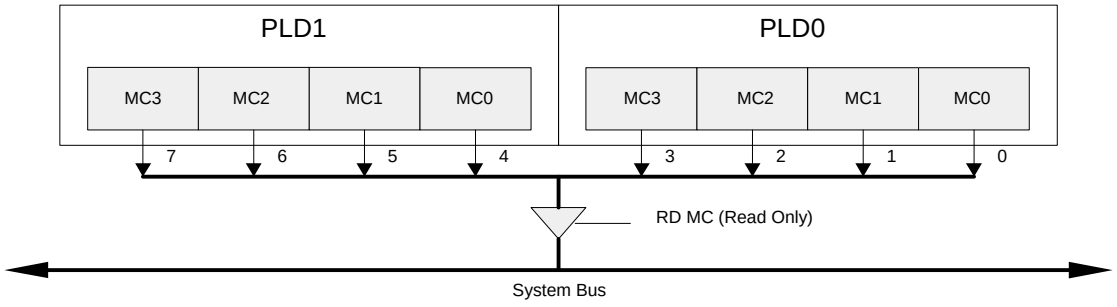
Figure 21-3. Macrocell Architecture



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.

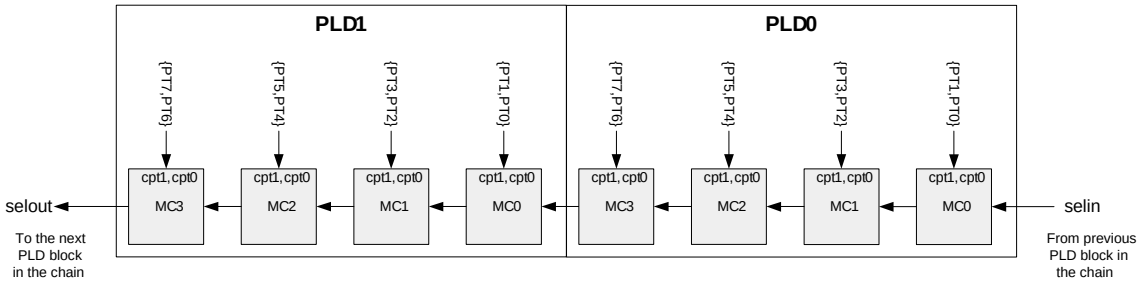
Figure 21-4. PLD Macrocell Read Only Register



21.3.1.2 PLD Carry Chain

PLDs are chained together in UDB address order. As shown in Figure 21-5 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 21-5. PLD Carry Chain and Special Product Term Inputs



21.3.1.3 PLD Configuration

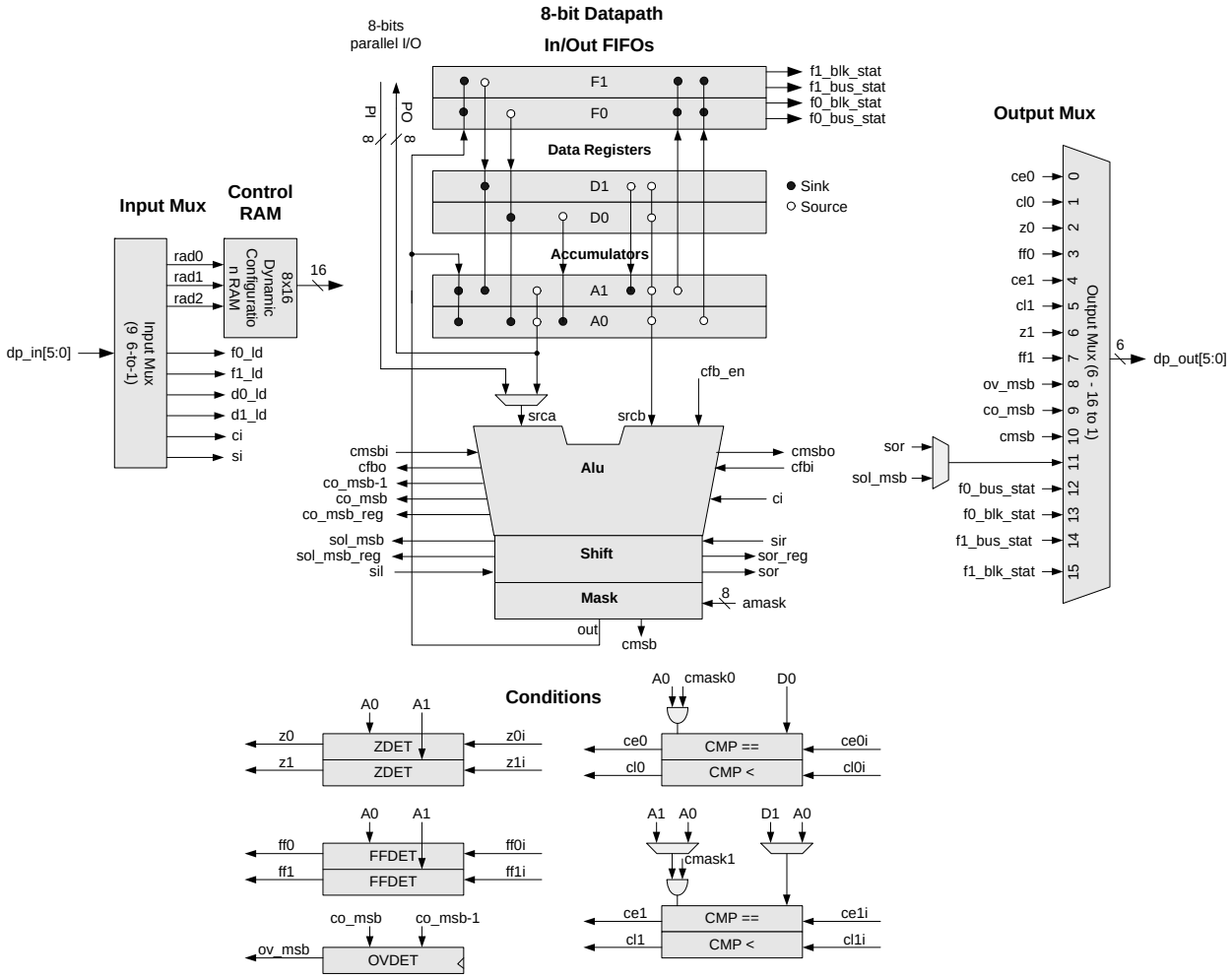
Each PLD appears to the CPU or DMA as a 16-bit wide RAM. The AND array has 12 X 8 X 2 bits, or 24 bytes, for programming, and the OR array has 4 x 8 bits, or 4 bytes, for programming. In addition, each macrocell has one configuration byte, resulting in 32 total configuration bytes per PLD. Because each UDB contains two PLDs, there are 64 total PLD configuration bytes per UDB. See UDB Configuration Address Space on page 222 for more information.

21.3.2 Datapath

The datapath, shown in Figure 21-6 below, 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 21-6. Datapath Top Level



21.3.2.1 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 21-1. 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. These registers retain their values across sleep intervals.
FIFOs	F0, F1	The two 4-byte FIFOs provide both a source and a destination for buffered data. The FIFOs can be configured as both input buffers, both output buffers, or as one input buffer and one output buffer. Status signals indicate the 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.

21.3.2.2 Datapath FIFOs

FIFO Modes and Configurations

Each FIFO has a variety of operation modes and configurations available:

Table 21-2. 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 21-7 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 21-7. FIFO Configurations

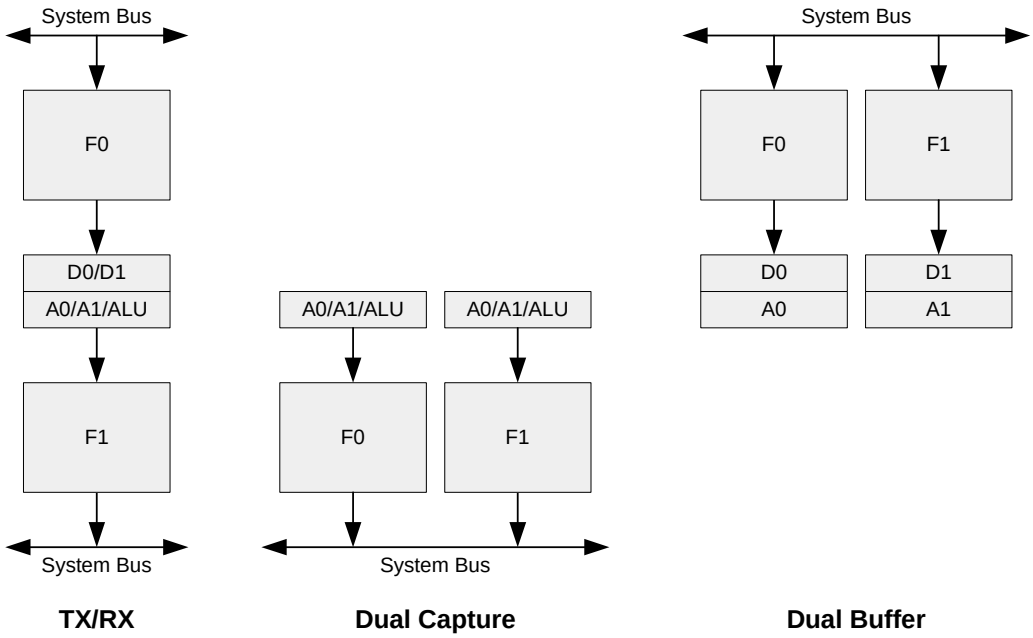
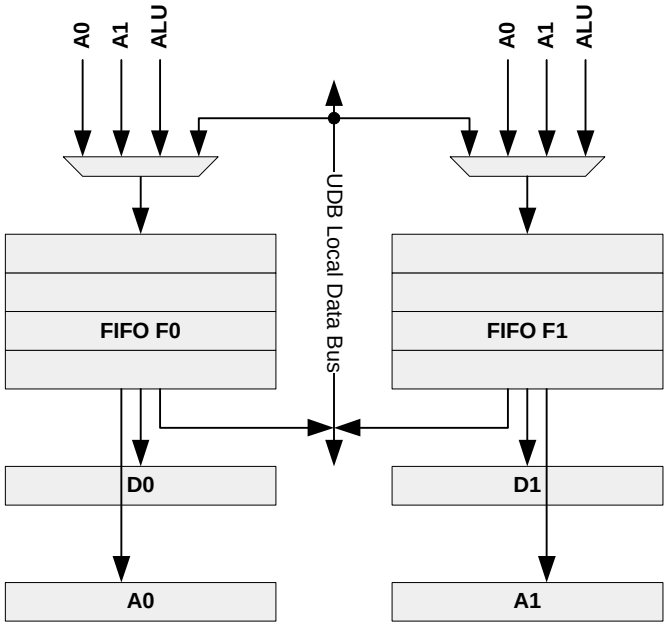


Figure 21-8 shows a detailed view of the FIFO sources and sinks.

Figure 21-8. 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 21-3. 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:

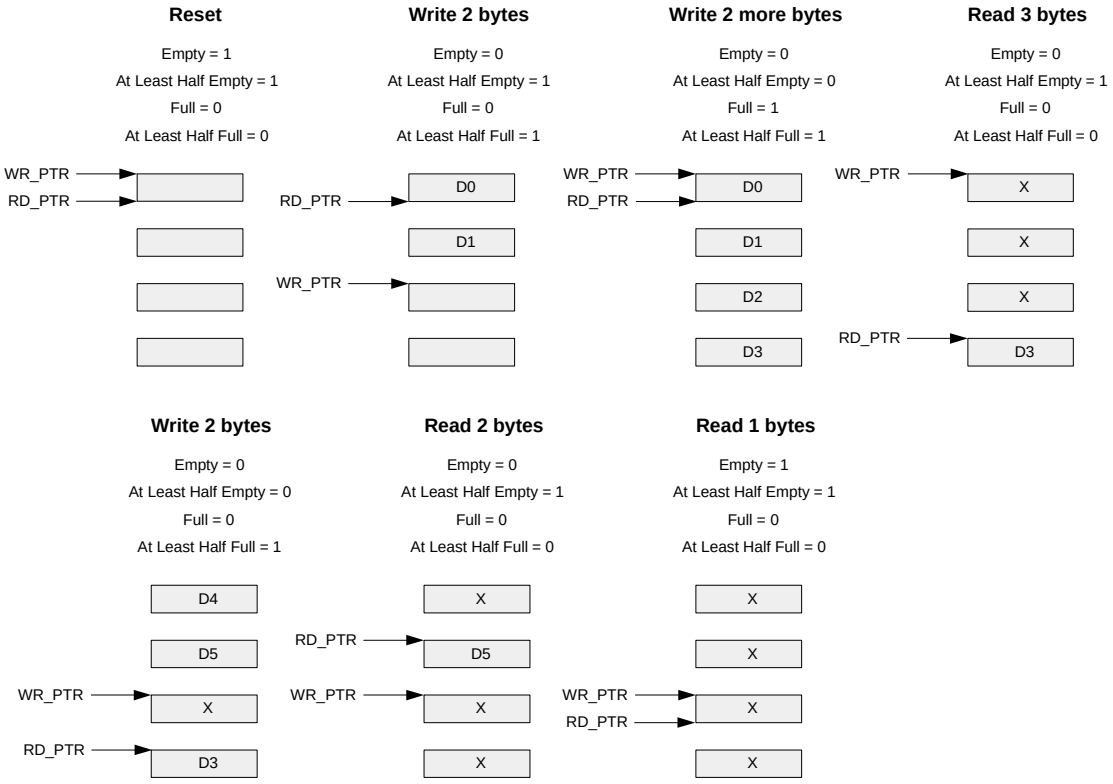
Table 21-4. FIFO Status Options

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 21-9 on page 193 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 21-9. Detailed FIFO Operation Sinks

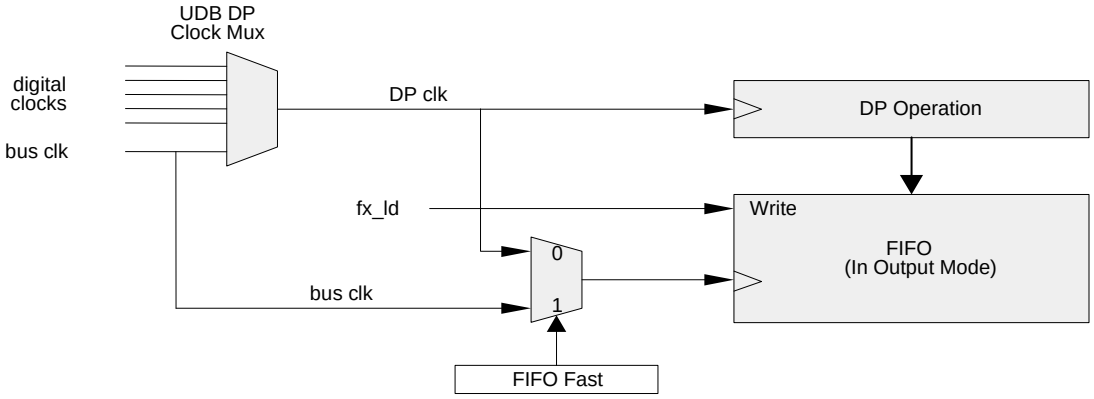


FIFO Fast Mode (FIFO FAST)

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 21-10, 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 21-10, the fast load operation is independent of the currently selected datapath clock, however, using the bus clock may cause higher power consumption.

Figure 21-10. FIFO Fast Configuration Sinks

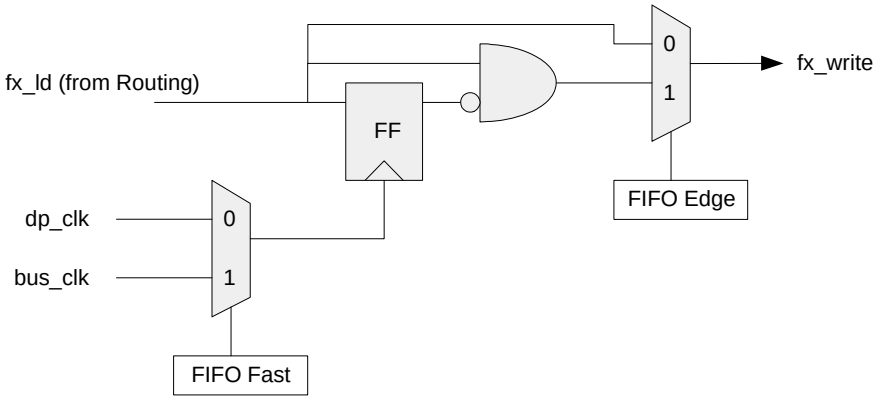


FIFO Edge/Level Write Mode

There are two modes for writing the FIFO from the datapath. In the first mode, data is synchronously transferred from the accumulators to the FIFOs. The control for that write (FX_LD) is typically generated from a state machine or condition that is synchronous to the datapath clock. The FIFO will be 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 21-11 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 21-11. 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 21-12, 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.

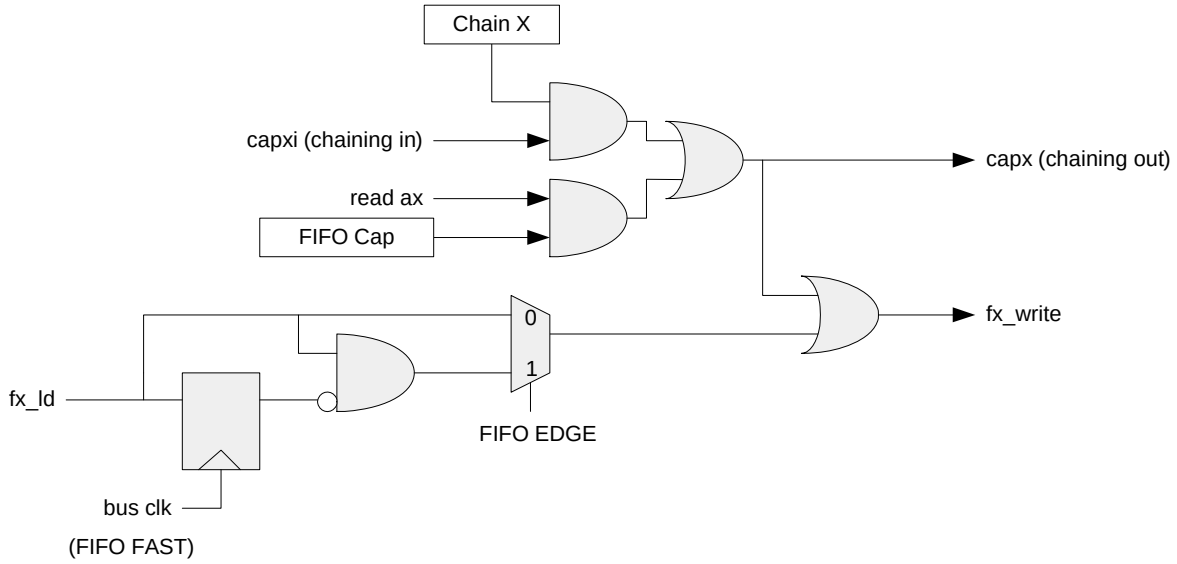
The routed FX_LD signal, which generates a FIFO load, is ORed with the software capture signal; the results can be unpredictable when both hardware and software capture are used at the same time. As a general rule these functions should be mutually exclusive, however, hardware and software capture can 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 21-12. 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 at 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 table below.

Table 21-5. 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 21-13 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.

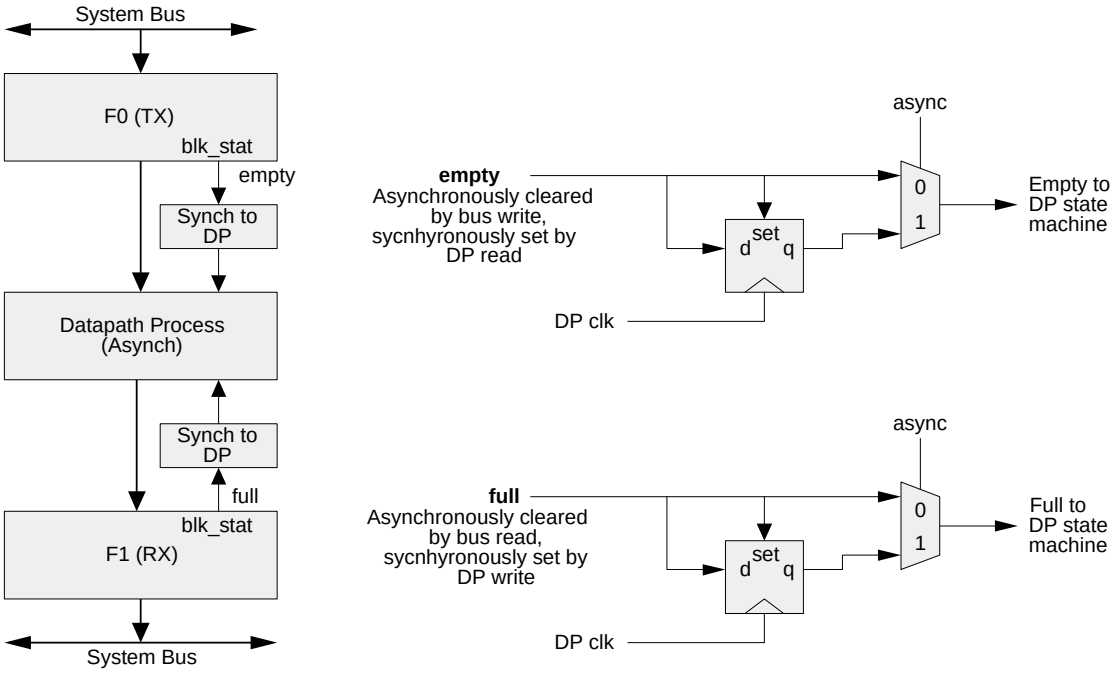
On the RX side, the datapath state machine uses “full” to determine whether there is a space left to write to the FIFO. Full is set synchronously to the DP state machine, but is cleared asynchronously due to a bus read. When cleared, the status is synchronized back to the DP state machine.

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 21-13. FIFO Asynchronous Operation



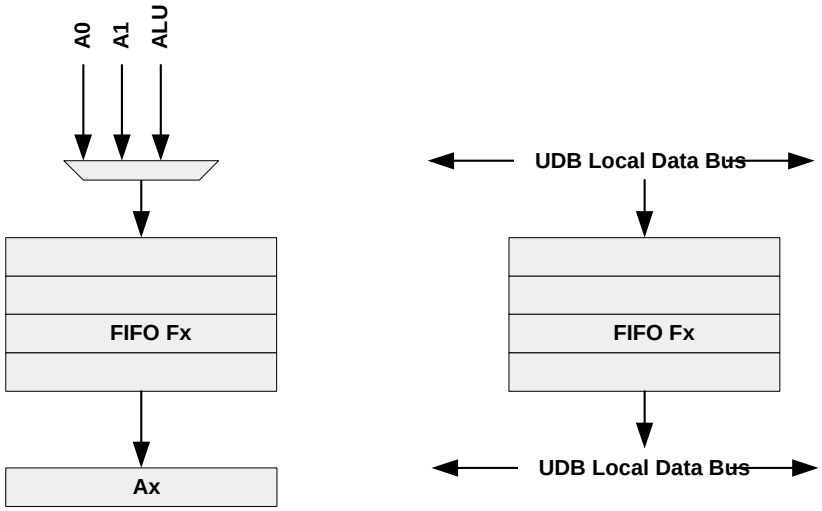
FIFO Clock Inversion Option

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 21-14 on page 197 shows the configurations available in dynamic FIFO mode.

Figure 21-14. FIFO Dynamic Mode



Internal Access **External Access**

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 21-6. 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

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

21.3.2.3 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 21.3.2.2 Datapath FIFOs on page 190.

21.3.2.4 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 21-7. 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 21-8.

Table 21-8. 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 21-9.

Table 21-9. Additional Carry In Functions

CI SEL A CI SEL B	Carry Mode	Description
00	Default	Default arithmetic mode as described in Table 21-8.
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 21-10. Note that in the case of the decrement and subtract functions, the carry is active low (inverted).

Table 21-10. 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.

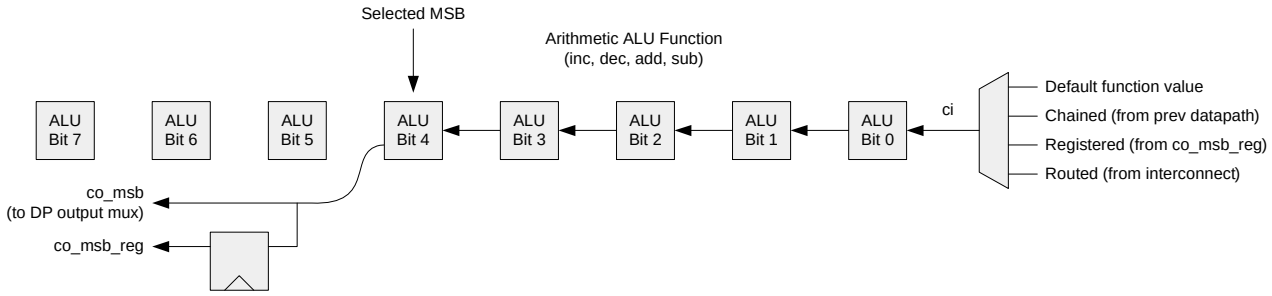
Table 21-11. Carry Out Functions

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 21-15 on page 199. 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 21-15. Carry Operation



Shift Operation

The shift operation occurs independently of the ALU operation, according to Table 21-12

Table 21-12. 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.

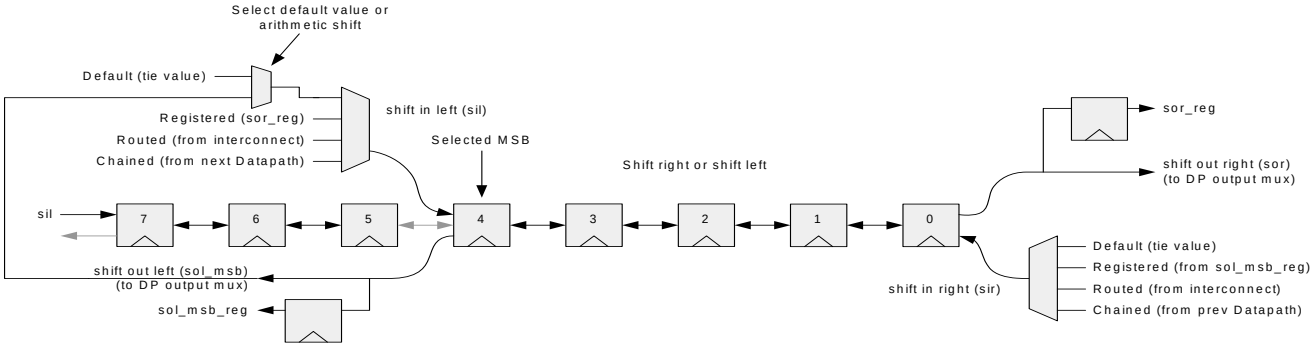
The SI SELA and SI SELB configuration bits determine the shift in data for a given operation. Dynamic configuration RAM selects the A or B configuration on a cycle-by-cycle basis. Shift in data is only valid for left and right shift; it is not used for pass and nibble swap. The selections and usage apply to both left and right shift directions and are shown in Table 21-13.

Table 21-13. 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 21-16. Shift Operation



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

ALU Masking Operation

An 8-bit mask register 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.

21.3.2.5 Datapath Inputs and Multiplexing

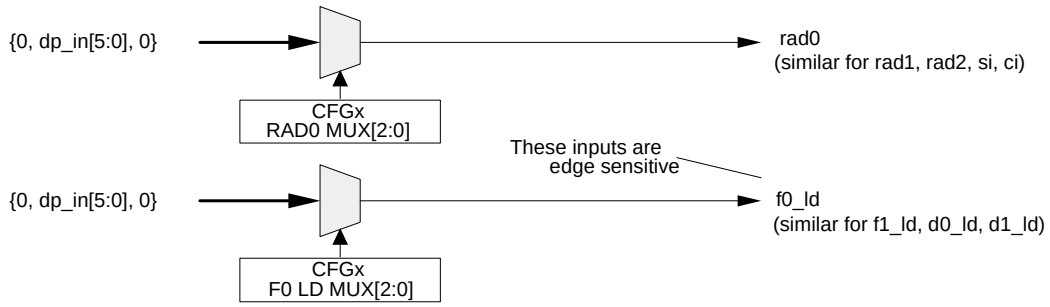
The datapath has a total of nine inputs as shown in Table 24-16, including six inputs from the channel routing. These consist of the configuration RAM address, FIFO and data register load control signals, and the data inputs shift in and carry in.

Table 21-14. 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 21-17, 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 21-17. Datapath Input Select



21.3.2.6 CRC/PRS Support

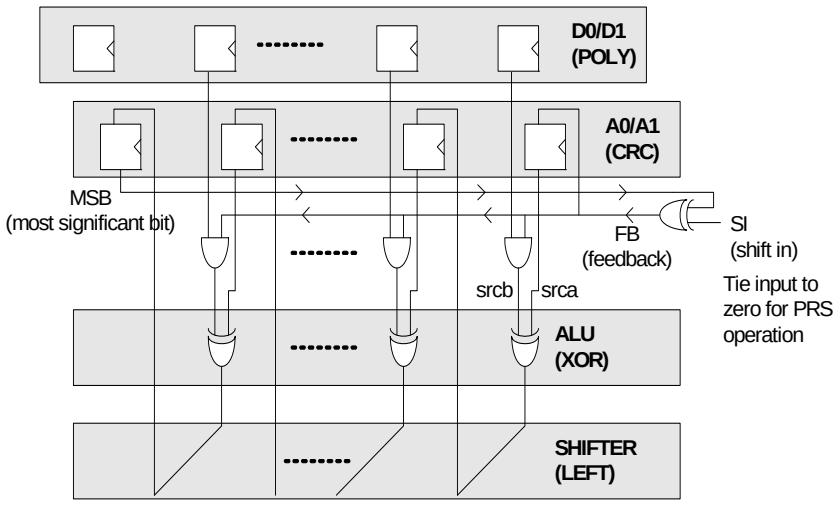
The datapath can support Cyclic Redundancy Checking (CRC) and Pseudo Random Sequence (PRS) generation. Chaining signals are routed between datapath blocks to support CRC/PRS bit lengths of longer than 8 bits.

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 21-18 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.

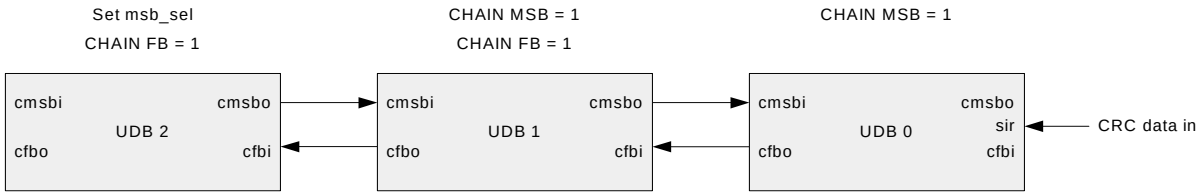
Figure 21-18. CRC Functional Structure



CRC/PRS Chaining

Figure 21-19 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 21-19. 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.

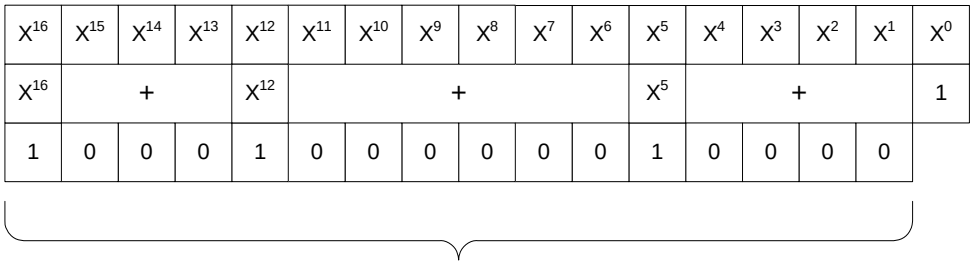
CRC/PRS Polynomial Specification

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 21-20.

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

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

Figure 21-20. 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 above.
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'.

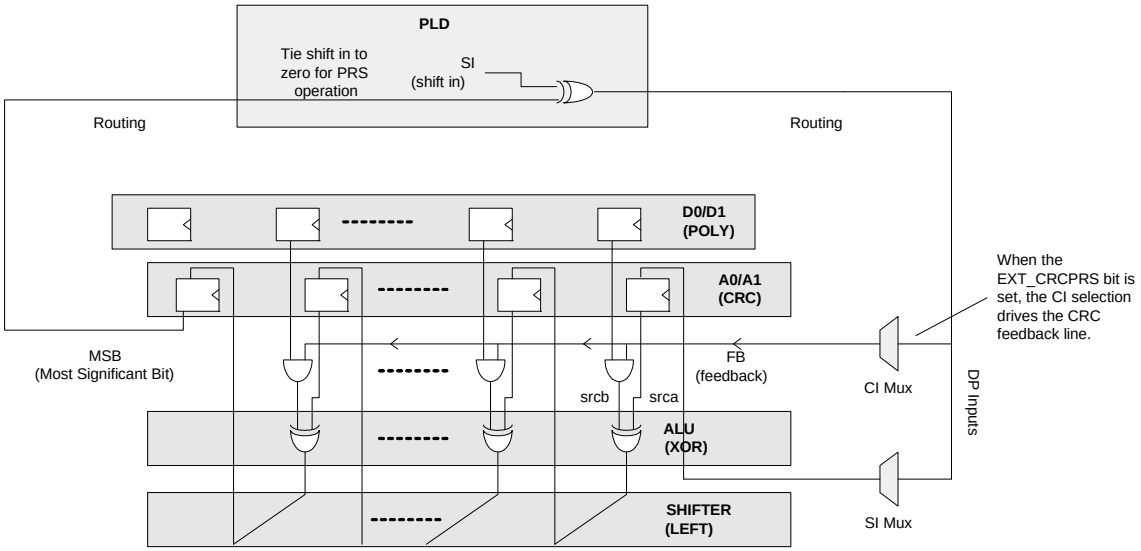
Clocking the UDB with this configuration generates the required CRC or outputs the MSB, which may be output to the routing for the PRS sequence.

External CRC/PRS Mode

A static configuration bit may be set (EXT CRCPRS) to enable support for external computation of a CRC or PRS. As shown in Figure 21-21, 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, as with the built-in CRC/PRS operation, the function can be interleaved with other functions if desired.

Figure 21-21. External CRC/PRS Mode



21.3.2.7 Datapath Outputs and Multiplexing

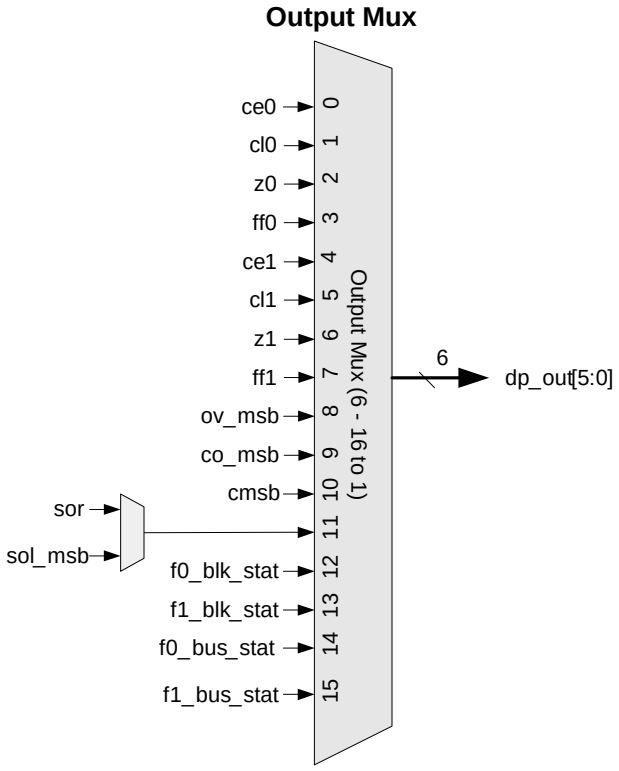
Conditions are generated from the registered accumulator values, ALU outputs, and FIFO status. These conditions can be driven to the digital routing for use in other UDB blocks, for use as interrupts or DMA requests, or to I/O pins. The 16 possible conditions are shown in the table below:

Table 21-15. 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 21-22, each output has a 16-1 multiplexer that allows any of these 16 signals to be routed to any of the datapath outputs.

Figure 21-22. 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 21-16. 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 21-23 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 21-23. Compare Equal Chaining

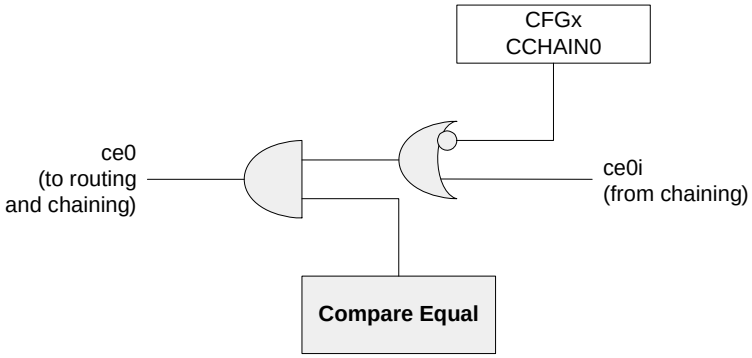
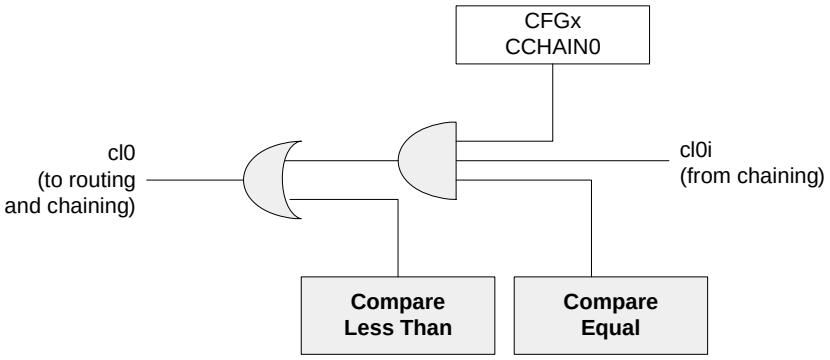


Figure 21-24 illustrates compare less than chaining. In this case, the “less than” is formed by the compare less than output in this block, which is unconditional. This is ORed with the condition where this block is equal, and the chained input from the previous block is asserted as less than.

Figure 21-24. 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.

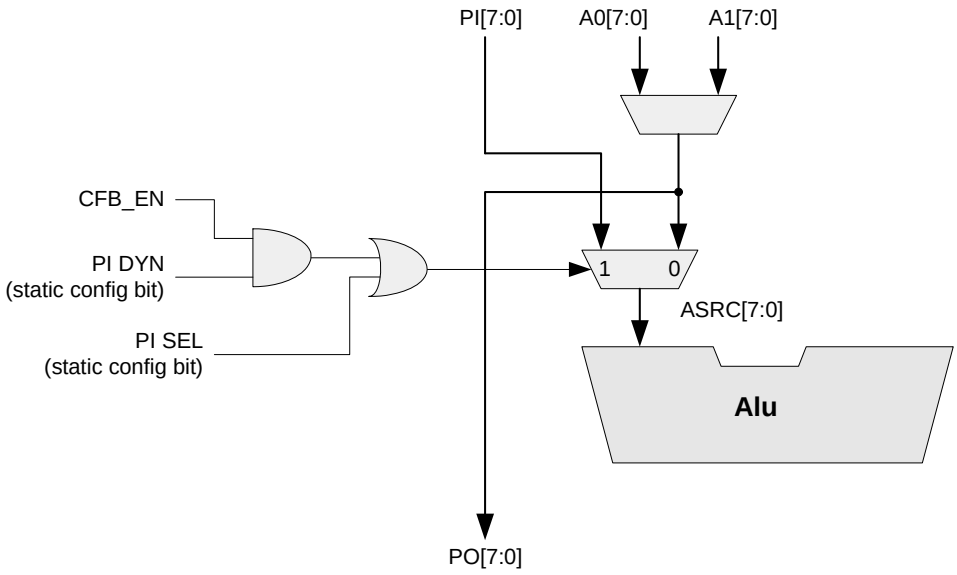
Overflow

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.

21.3.2.8 Datapath Parallel Inputs and Outputs

As shown in Figure 21-25, 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 21-25. Datapath Parallel In/Out

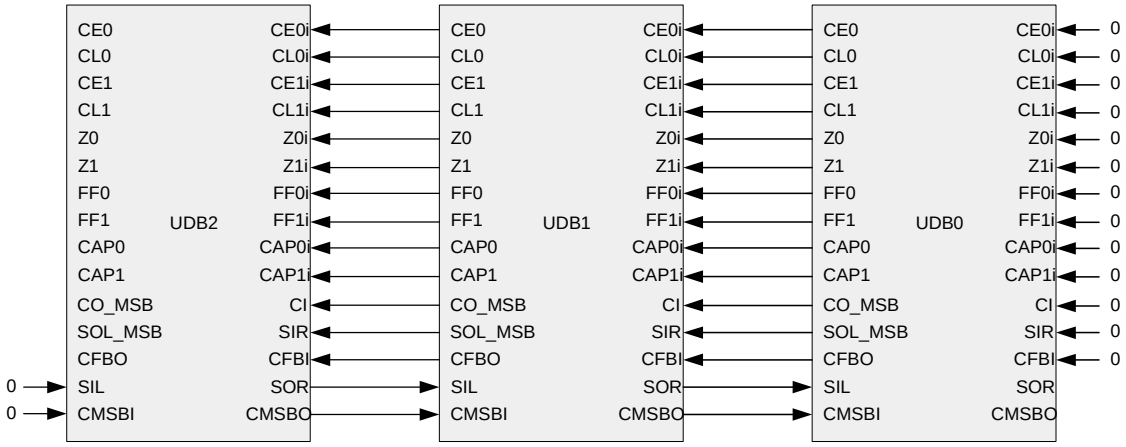


Parallel In needs to be selected for input to the ALU. There are two options, static operation or dynamic operation. For static operation, the PI SEL bit forces the ALU asrc to be PI. The PI DYN bit is used to enable the PI dynamic operation. When it is enabled, and assuming the PI SEL is 0, the PI multiplexer 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.

21.3.2.9 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 21-21, 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 21-26. Datapath Chaining Flow

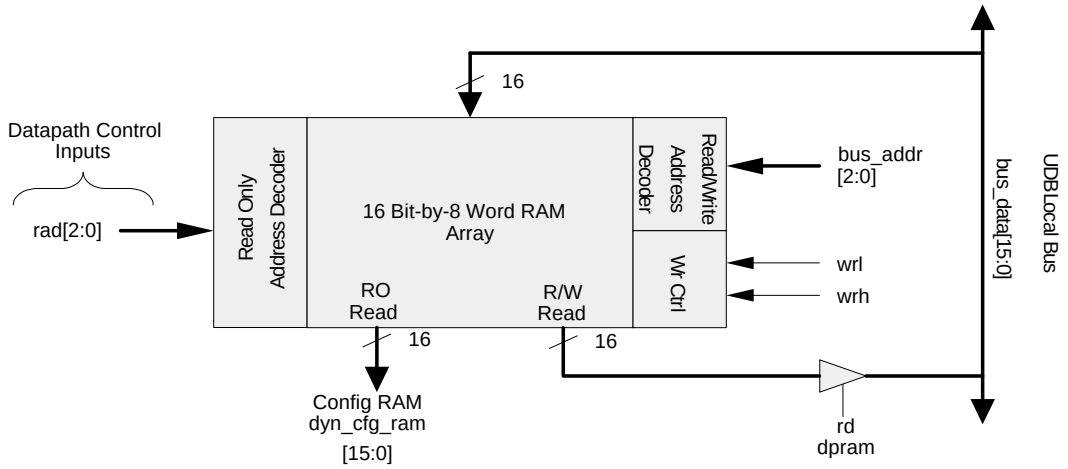


21.3.2.10 Dynamic Configuration RAM

Each datapath contains a 16 bit-by-8 word dynamic configuration RAM, which is shown in Figure 21-27. 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 21-27. 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

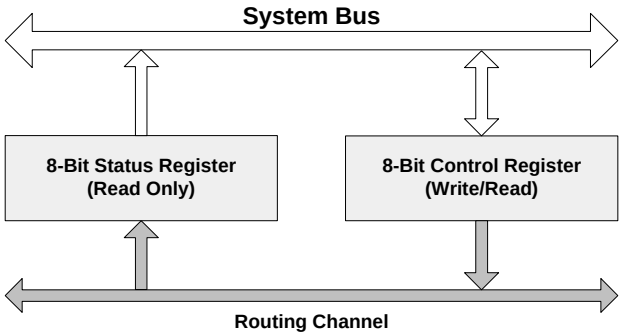
Table 21-17. Dynamic Configuration Quick Reference

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.			

21.3.3 Status and Control Module

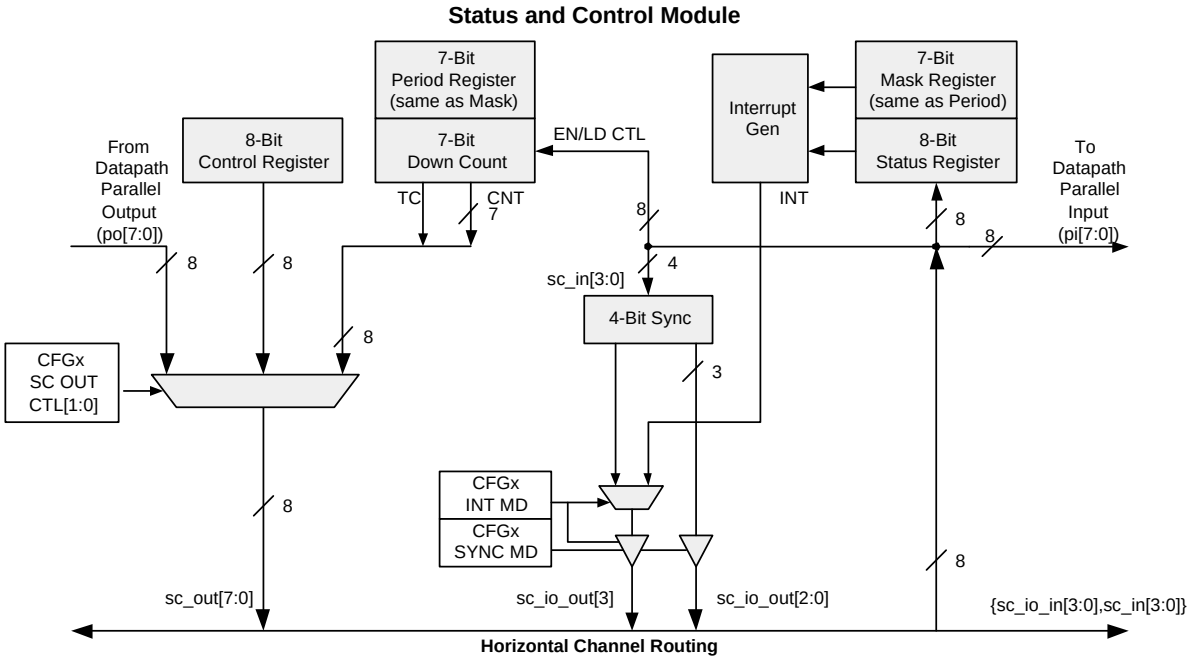
A high level view of the Status and Control module is shown in Figure 21-28. The Control register drives into the routing to provide firmware control inputs to UDB operation. The Status register read from routing provides firmware a method of monitoring the state of UDB operation.

Figure 21-28. Status and Control Registers



A more detailed view of the Status and Control module is shown in Figure 21-29. 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 21-29. Status and Control Module



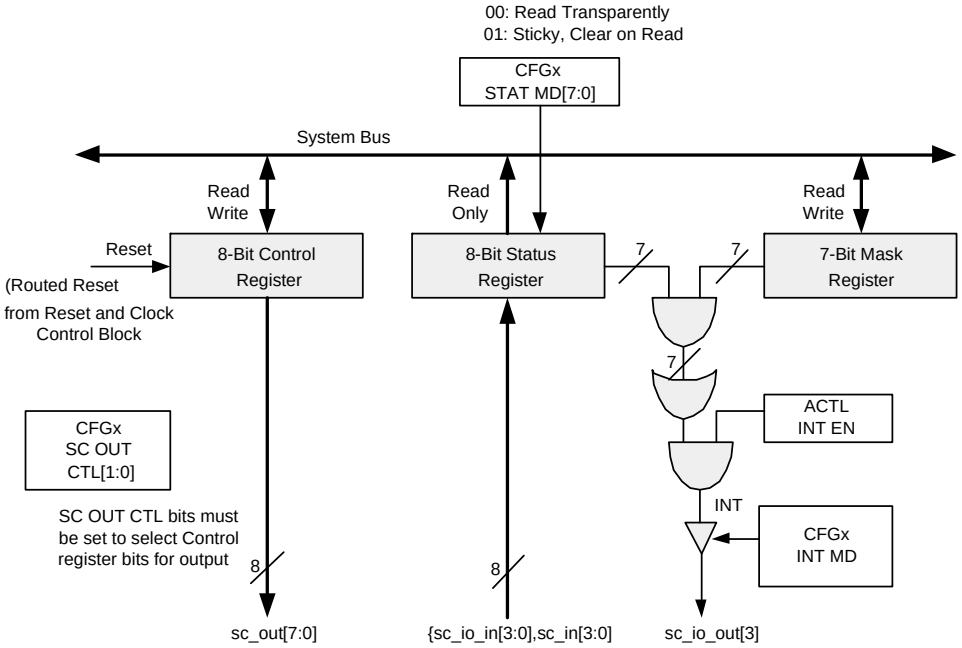
Modes of operation include:

- **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.

21.3.3.1 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 21-30 on page 209.

Figure 21-30. 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 below:

Table 21-18. 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.

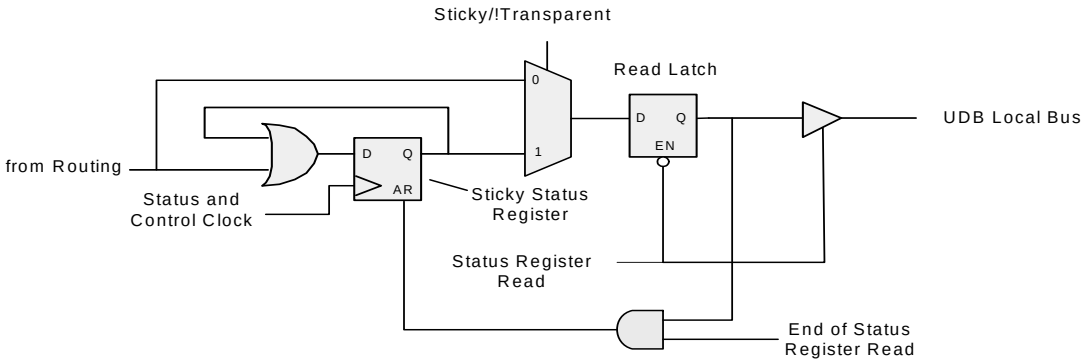
Sticky Status, with Clear on Read

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 21-31 on page 210 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 21-31. Status Read Logic



Interrupt Generation

In most functions, interrupt generation is tied to the setting of status bits. As shown in Figure 21-31, 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.

21.3.3.2 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 21-19.

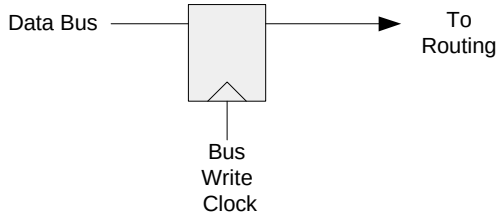
Table 21-19. Mode for Control Register Bit 0

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 21-32, 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.

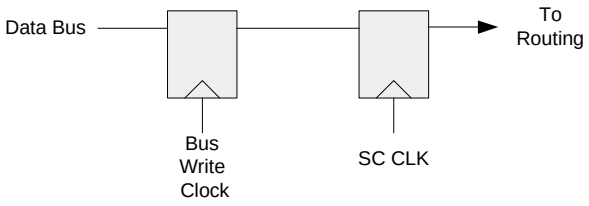
Figure 21-32. Control Register Direct Mode



Control Register Sync Mode

In Sync mode, as shown in Figure 21-33, 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 21-33. 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.

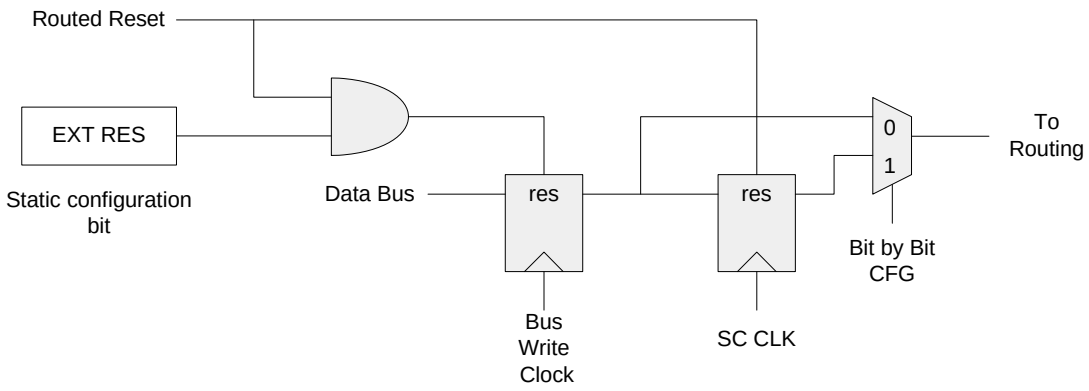
With this mode of operation, firmware can write a 1 to a control register bit to generate a pulse. After it is written as a 1 it

will be read back by firmware as a 1 until the completion of the pulse, after which it will be 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 21-34. 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.

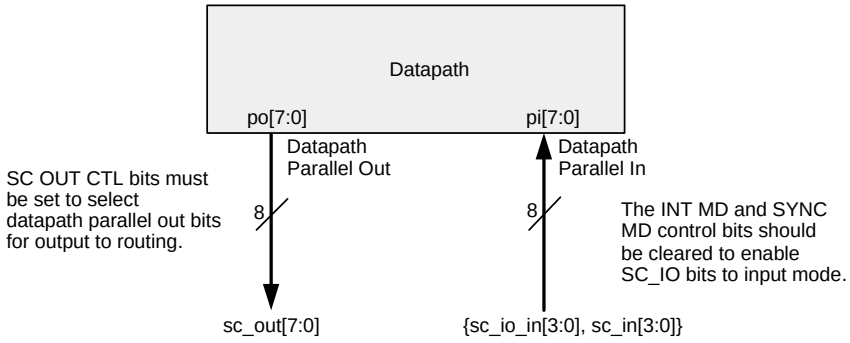
Figure 21-34. Control Register Reset



21.3.3.3 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 21-35. Parallel Input/Output Mode



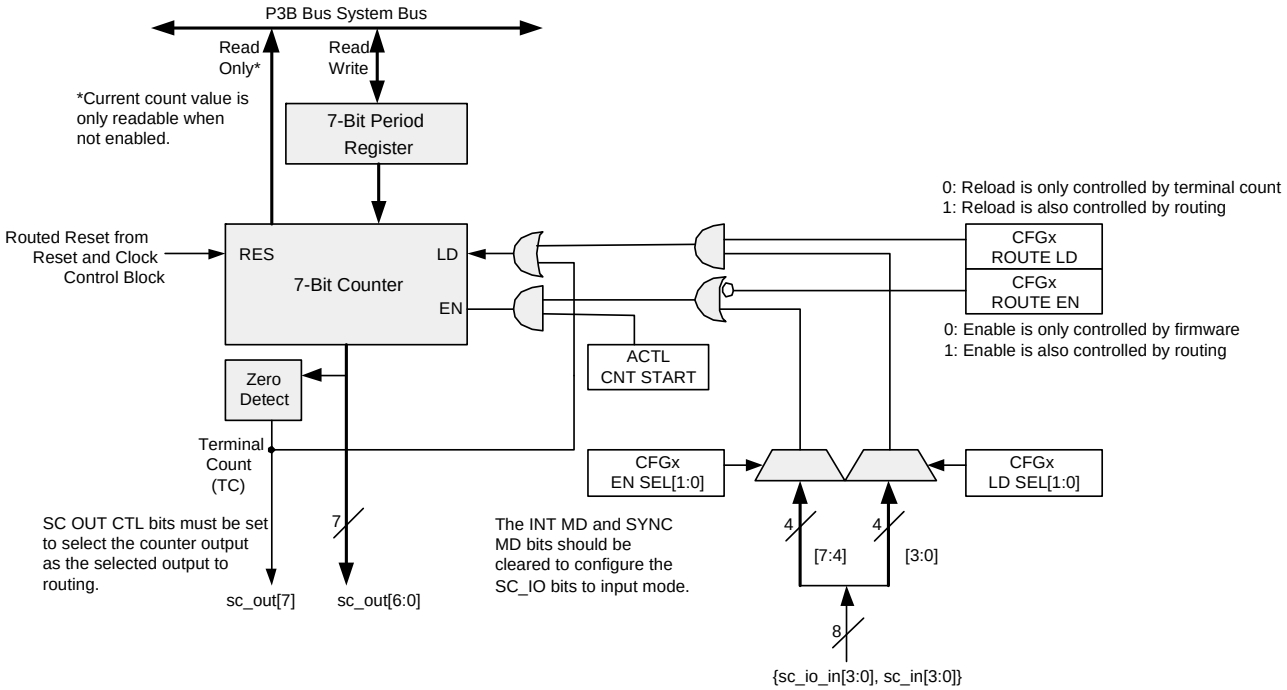
21.3.3.4 Counter Mode

As shown in Figure 21-36, 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. The Period register is retention and will maintain its state across sleep intervals. For a period of N clocks, the period value of N-1 should be loaded. N = 1 (period of 0) is not supported as a clock divide value, and will result in the terminal count output of a constant 1.The use of SYNC mode depends on whether or not 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.

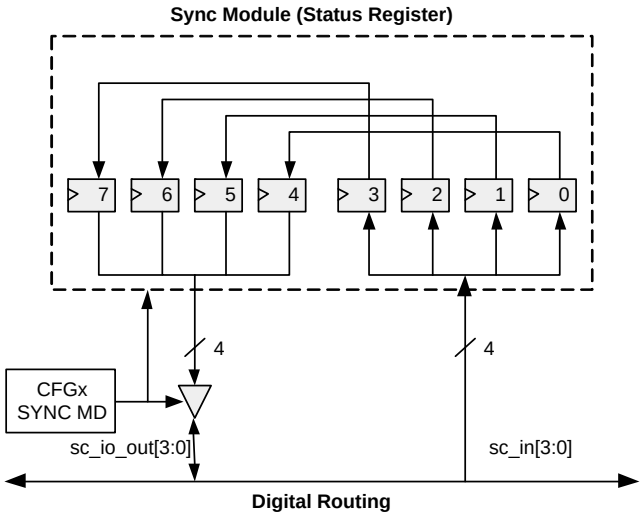
Figure 21-36. Counter Mode



21.3.3.5 Sync Mode

As shown in Figure 21-37, 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 21-37. Sync Mode



21.3.3.6 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 21.3.4 Reset and Clock Control Module on page 214.

21.3.3.7 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 table below.

Table 21-20. 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 may be used to enable and disable the counter (only valid when the SC_OUT_CTL[1:0] bits are configured for counter output mode).

21.3.3.8 Status and Control Register Summary

The table below 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 21-21. 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]

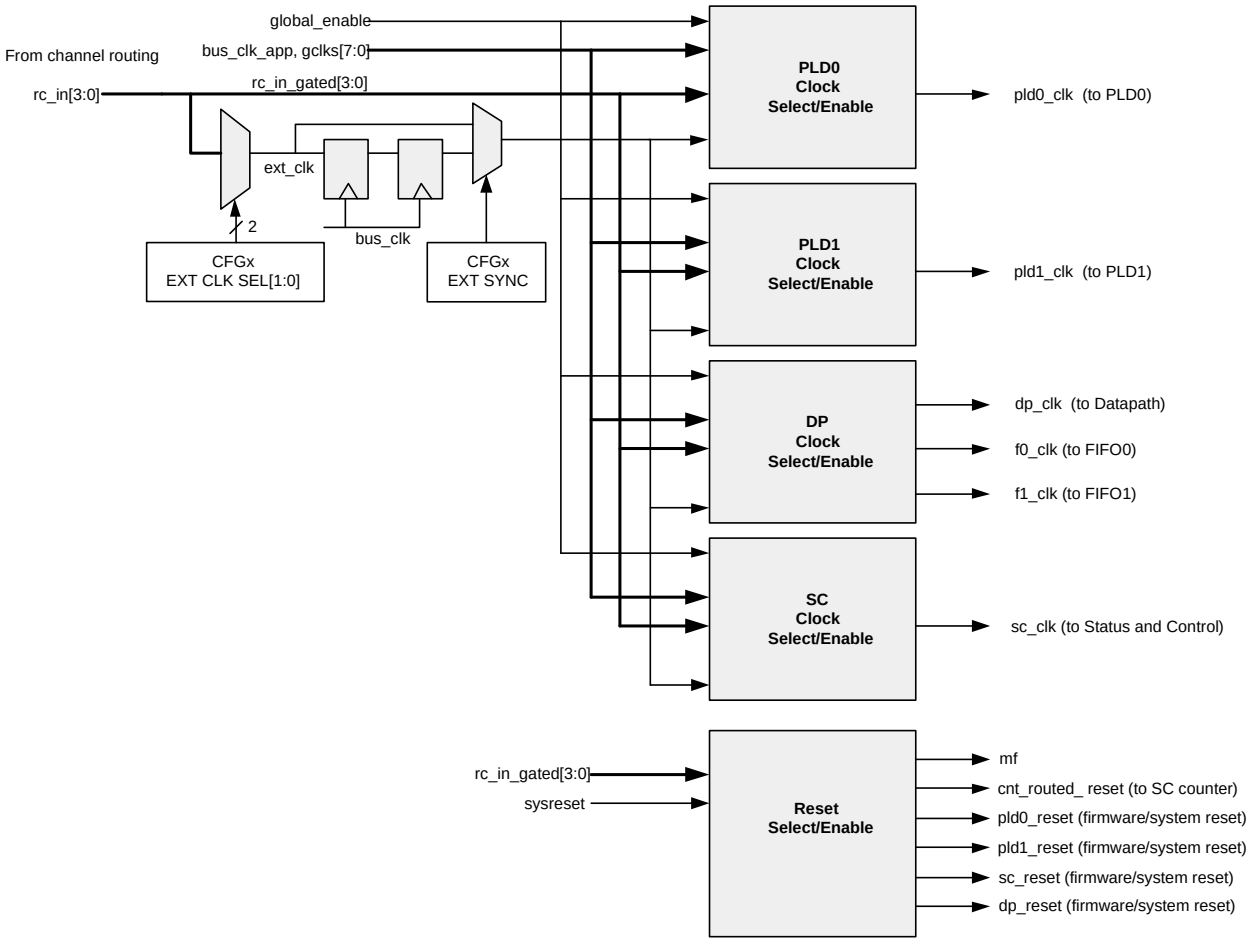
a. Note that in counter mode, the mask register is operating as a period register and cannot function as a mask register. Therefore, interrupt output is not available when counter mode is enabled.
b. Note that in SYNC mode, the status register function is not available, and therefore, the mask register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

21.3.4 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 21-38, 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: 8 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 just like 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.

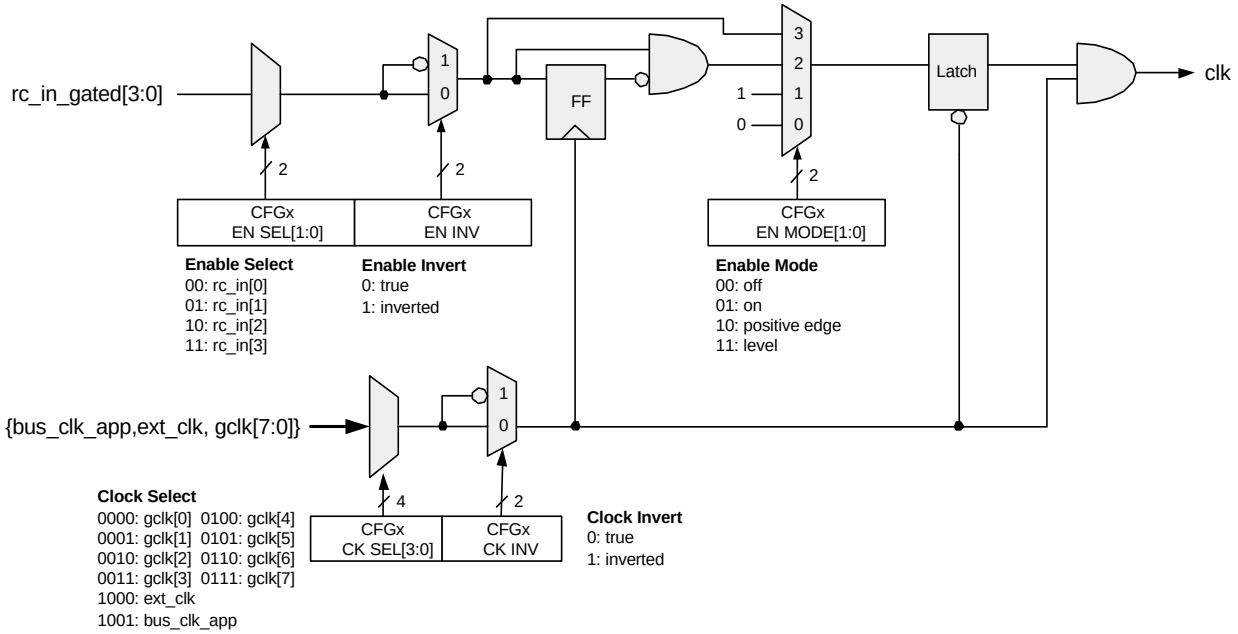
Figure 21-38. Reset and Clock Control



21.3.4.1 Clock Control

Figure 21-39 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 21-39. 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 123. 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 may be routed to any synchronous signal and can be selected from any of the four inputs from the routing matrix that are available to this block.

Clock Enable Inversion

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

Clock Enable Mode

By default, the clock enable is OFF. After configuring the target block operation, software can set the mode to one of the following using the CFGxEN MODE[1:0] register shown in Figure 21-39.

Table 21-22. 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. Because the boundary of a function mapped into the UDB array is arbitrary–it may span multiple UDBs and/or portions of UDBs–there must be a way to enable a given function atomically. This is typically implemented from a bit

in a control register routed to one or more clock enable inputs. This scenario also supports the case where applications require multiple, unrelated blocks to be enabled simultaneously.

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

Special FIFO Clocking

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

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

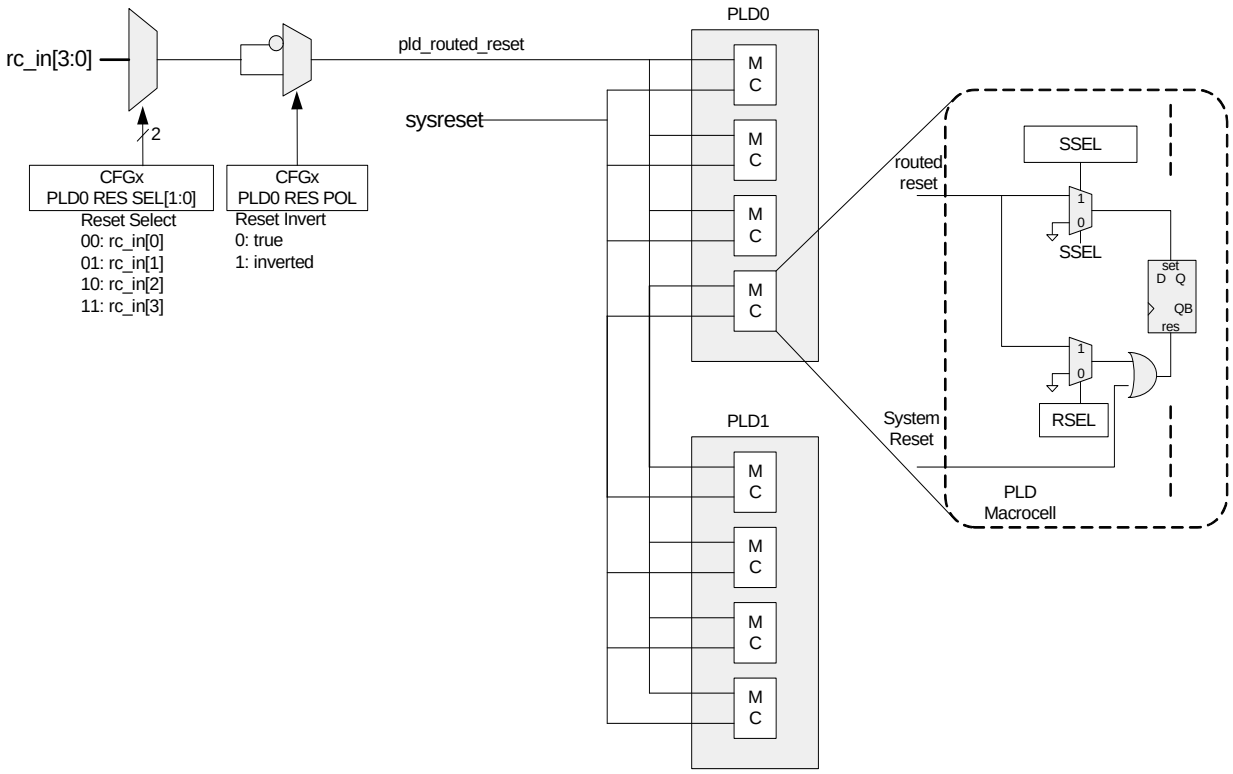
21.3.4.2 Reset Control

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 on legacy mode reset.

PLD Reset Control

Figure 21-40 shows the PLD reset system.

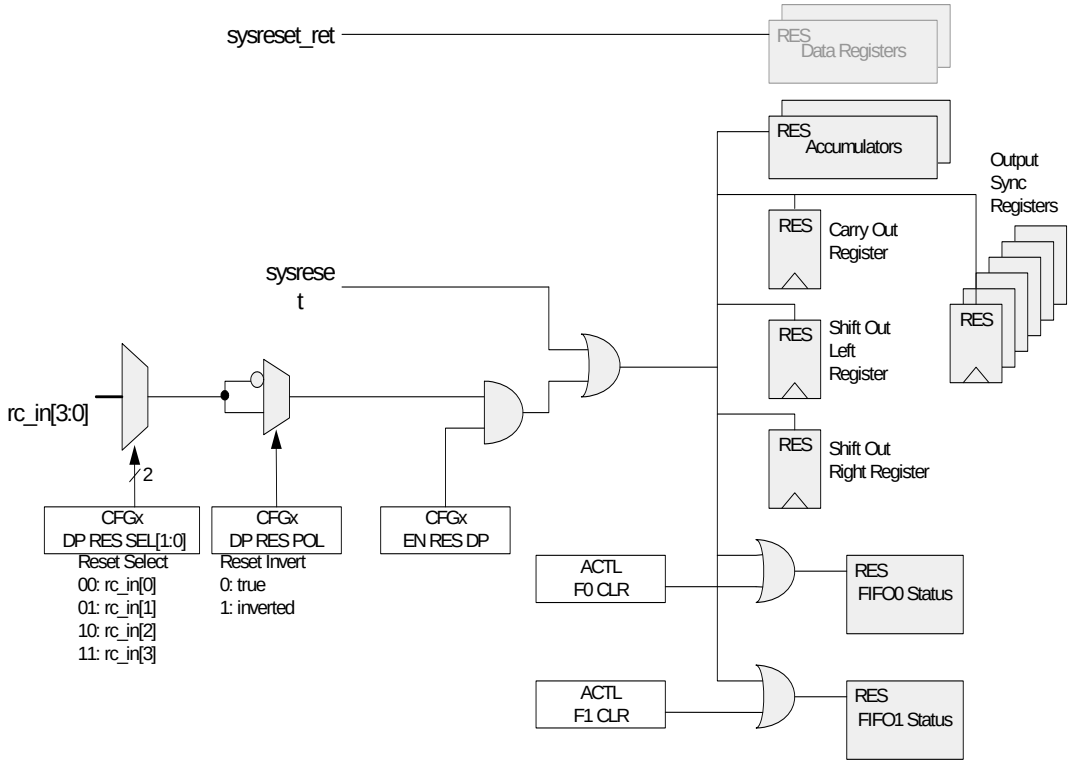
Figure 21-40. PLD Reset Structure



Datapath Reset Control

Figure 21-41 shows the datapath reset system. The routed reset is applied to all datapath registers and states except the data registers D0 and D1. The data registers are retention registers. The FIFO data is unknown after reset because it is RAM based.

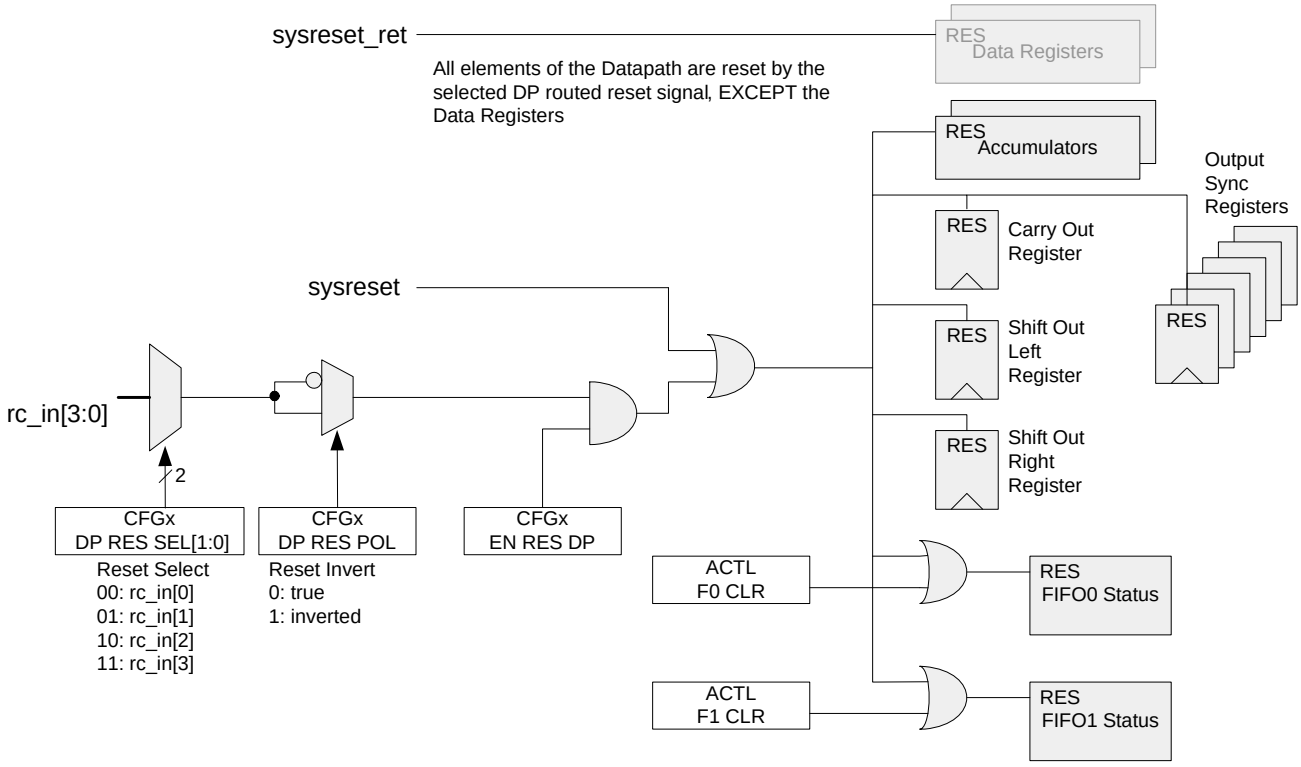
Figure 21-41. Datapath Reset Structure



Status and Control Reset Control

Figure 21-42 shows the status and control block reset system. The status and control/count registers share the routed reset, however they are individually enabled. The mask/period and auxiliary control registers are retention registers.

Figure 21-42. Status and Control Reset Control



21.3.4.3 UDB POR Initialization

Register and State Initialization

Table 21-23. 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.

21.3.5 UDB Addressing

There are three unique address spaces in a UDB pair:

- 8-Bit Working Registers – A bus master that can only access 8 bits of data per bus cycle, such as the PSoC 3 8051, 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, 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.

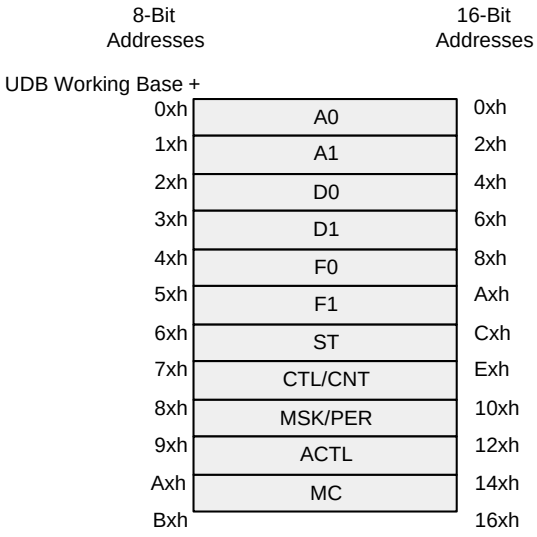
21.3.5.1 Working Register Address Space

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 21-43 shows the register map for one UDB.

On the right in Figure 21-43 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 21-43. UDB Working Registers

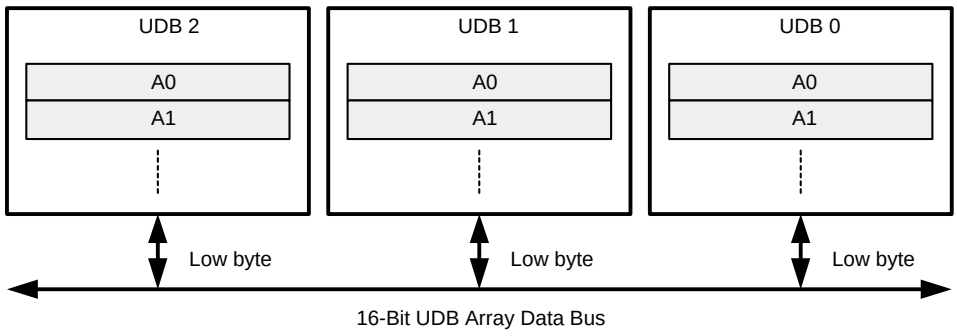


8-Bit Working Register Access

In this mode, all UDB registers are accessed on byte-aligned addresses. In 8-bit register access mode, as shown in Figure 21-44, 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.

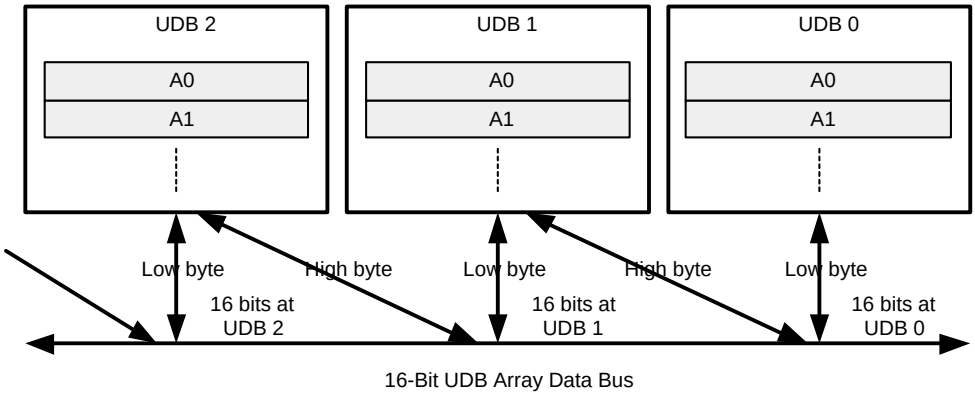
Figure 21-44. 8-Bit Working Register Access



16-Bit Working Register Address Space

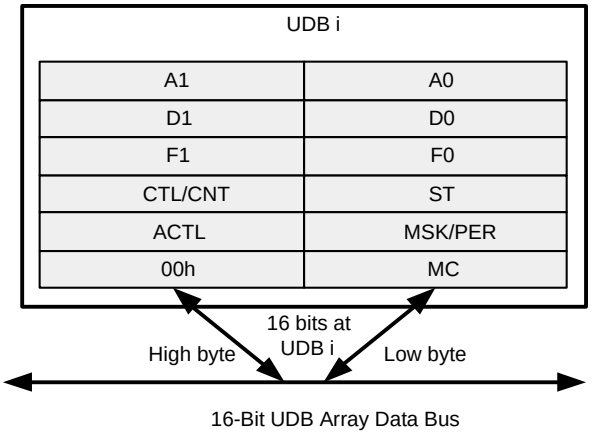
The 16-bit address space is designed for efficient DMA access and to provide support for CPU firmware access in processors that can support it. There are two modes of 16-bit register access, the “default” mode and the “concat” mode. As shown in Figure 21-45, the default mode accesses a given register 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 21-45. 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 21-46. 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 21-46. 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 21-24.

Table 21-24. Optimized Address Space for 16-Bit UDB Function

Address	Upper Byte Goes	Lower Byte Goes
0	UDB1	UDB0
2	UDB2	UDB1
4	UDB3	UDB2

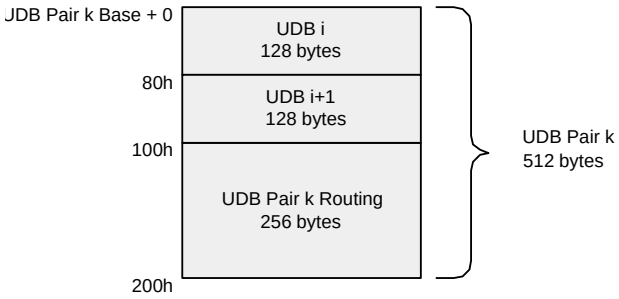
When the DMA transfers 16 bits to address 0, the lower and upper bytes are written to UDB0 and UDB1, respectively. On the next 16 bit DMA transfer at address 2, you overwrite the value in UDB1 with the lower byte of that transfer.

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.

21.3.5.2 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 21-47.

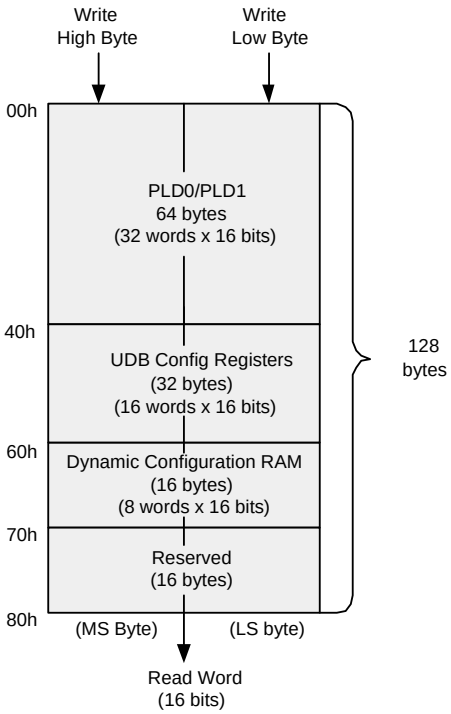
Figure 21-47. UDB Pair Configuration Address Map



21.3.5.3 UDB Configuration Address Space

Figure 21-48 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 21-48. UDB Configuration Address Space



21.3.5.4 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 225.

21.3.6 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.

21.3.6.1 Simultaneous System Bus Access

The following table lists the possible simultaneous access events and required behavior:

Table 21-25. 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.

21.3.6.2 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 194. 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.

21.4 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]			

22. 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.

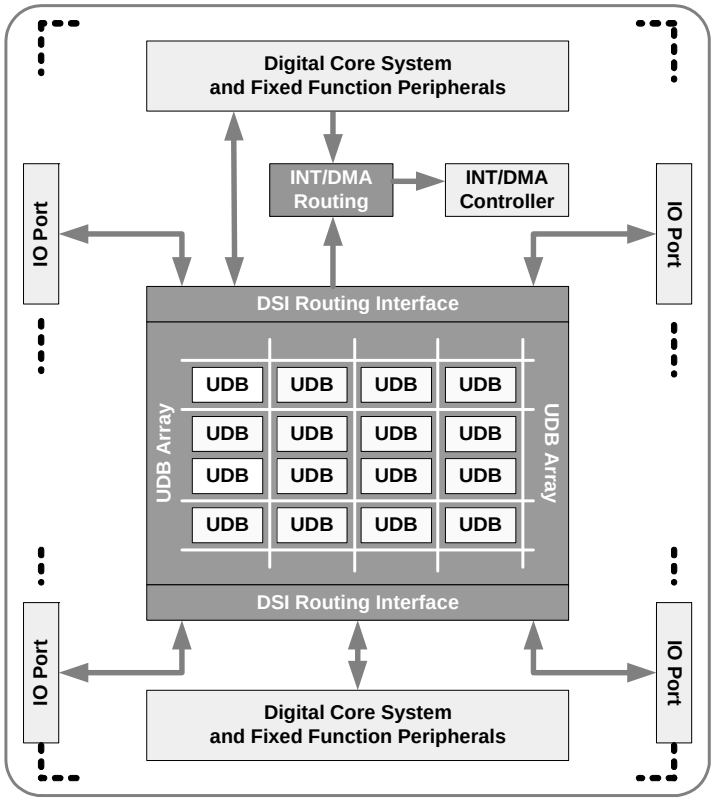
22.1 Features

- 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

22.2 Block Diagram

Figure 22-1 illustrates the programmable digital architecture for PSoC 3.

Figure 22-1. 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.

22.3 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 22-2 and Figure 22-4 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 22-2. DSI Example Connections to the Interrupt and DMA Controller

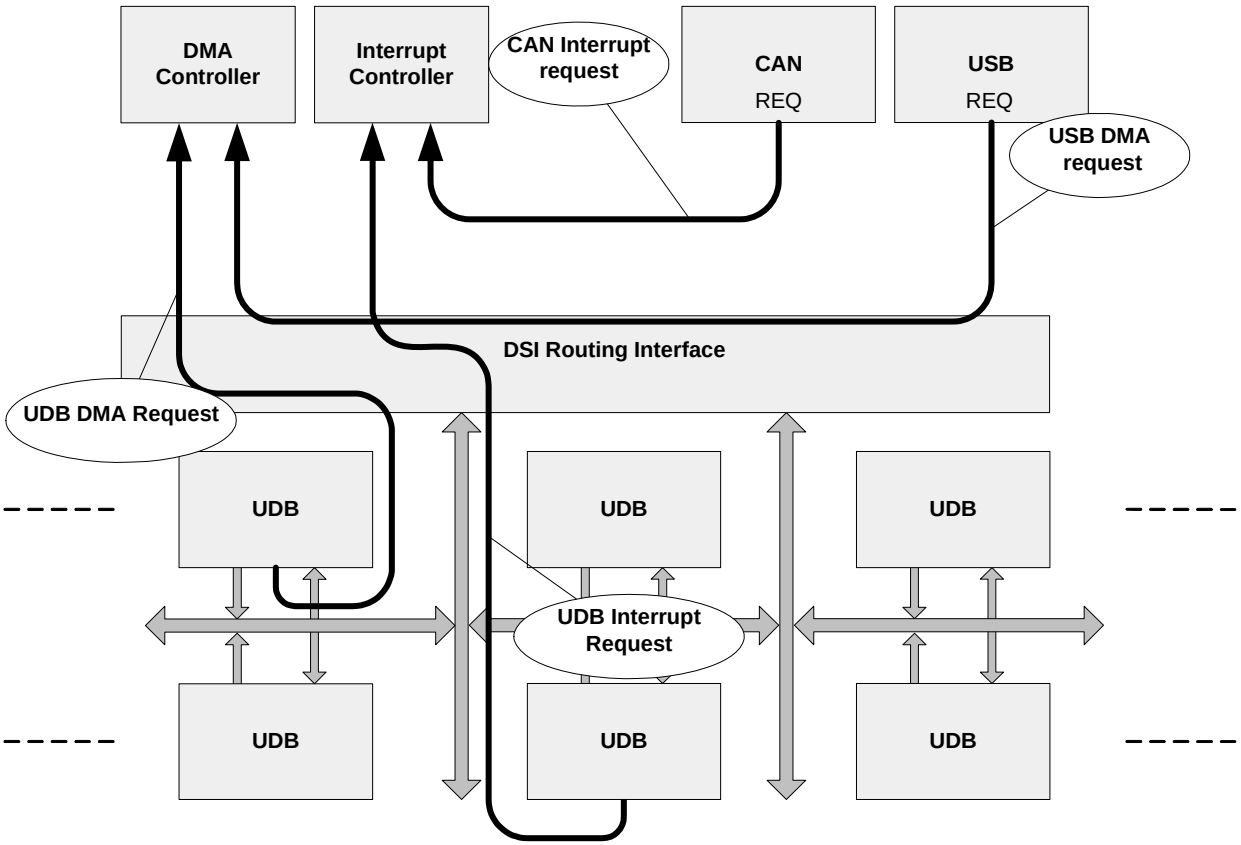
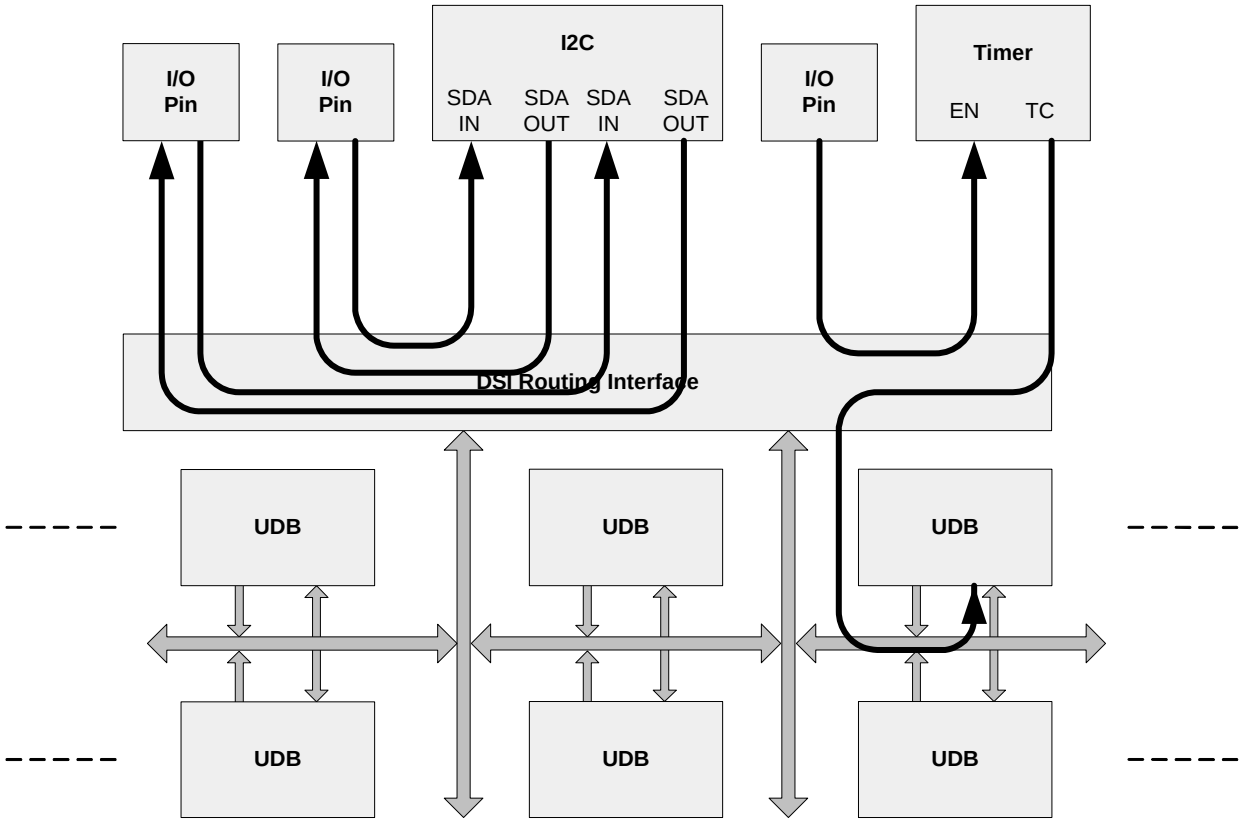


Figure 22-3. DSI Example Connections between Peripherals, I/O Pins, and UDBs



22.4 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.

22.4.1 UDB Array POR Initialization

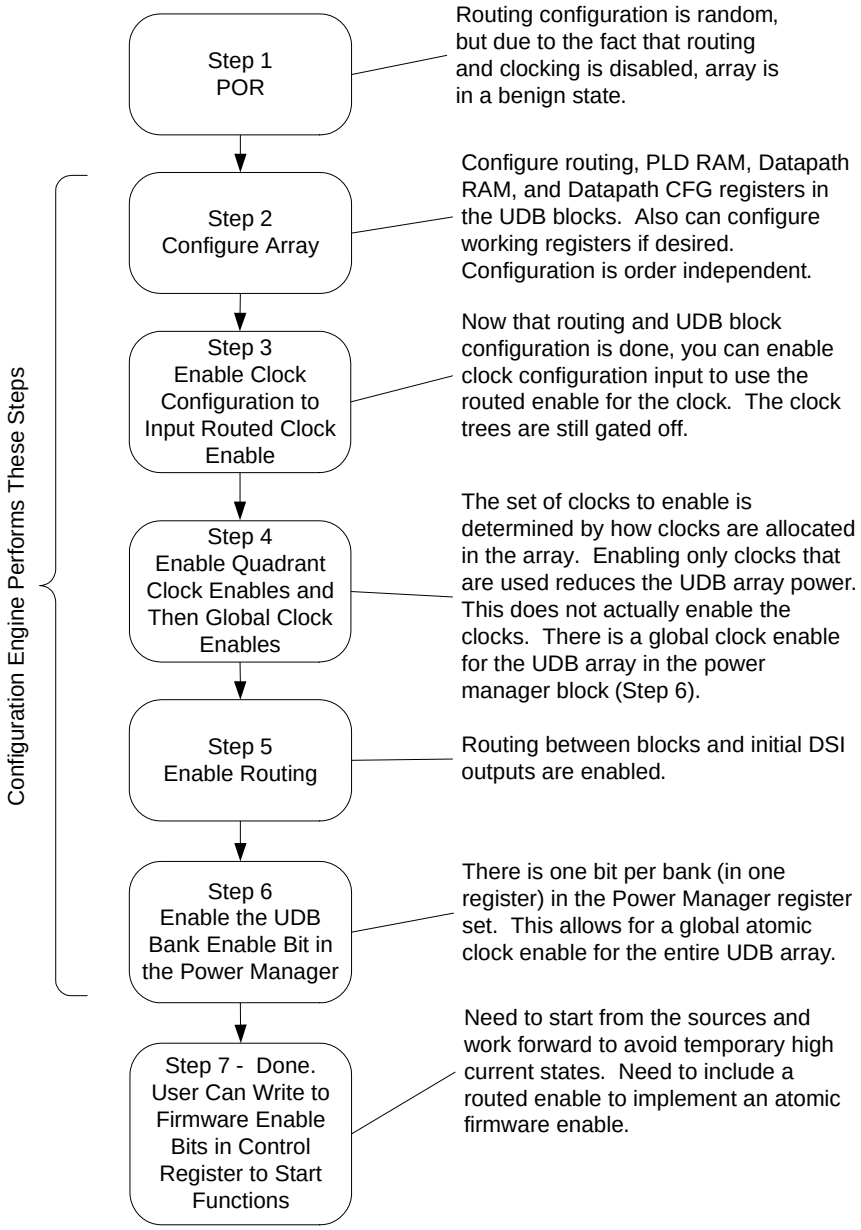
The key aspects of POR initialization are summarized as follows.

- 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.

22.4.2 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 22-4.

Figure 22-4. POR Configuration Sequence



22.4.2.1 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).

22.4.3 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.

22.4.4 UDB Register References and Address Mapping

UDB registers are classified as shown in the Figure 22-5. 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® 3 Registers TRM (Technical Reference Manual)*.

Figure 22-5. UDB Register Mapping

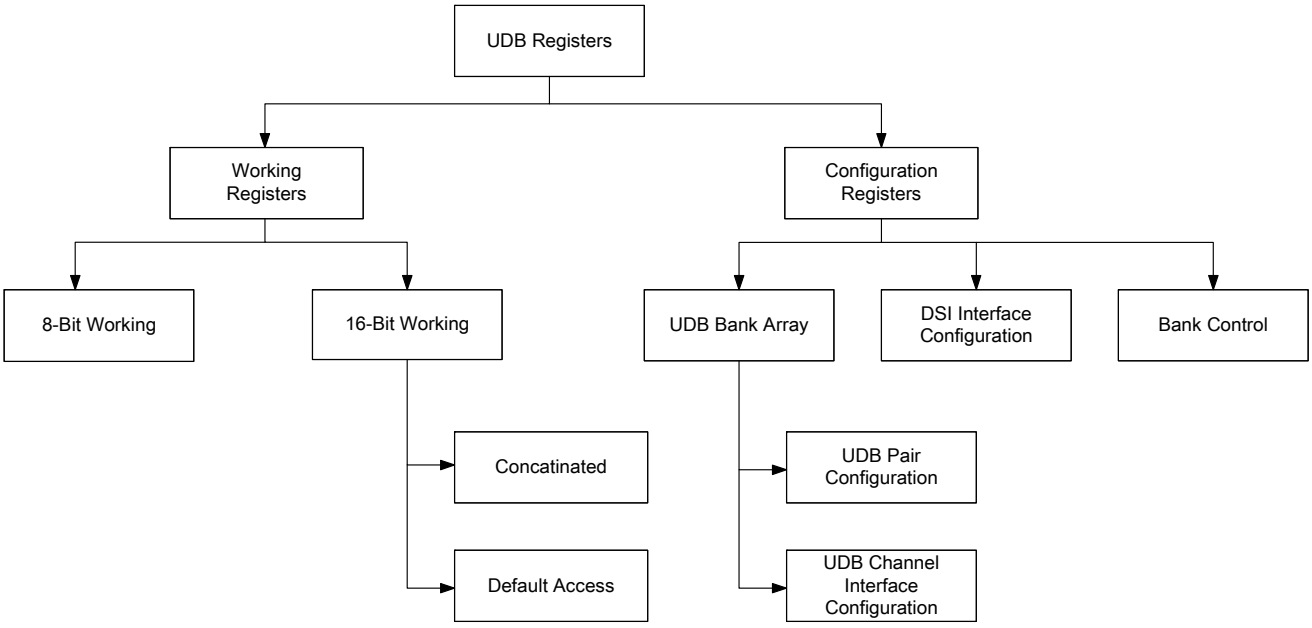
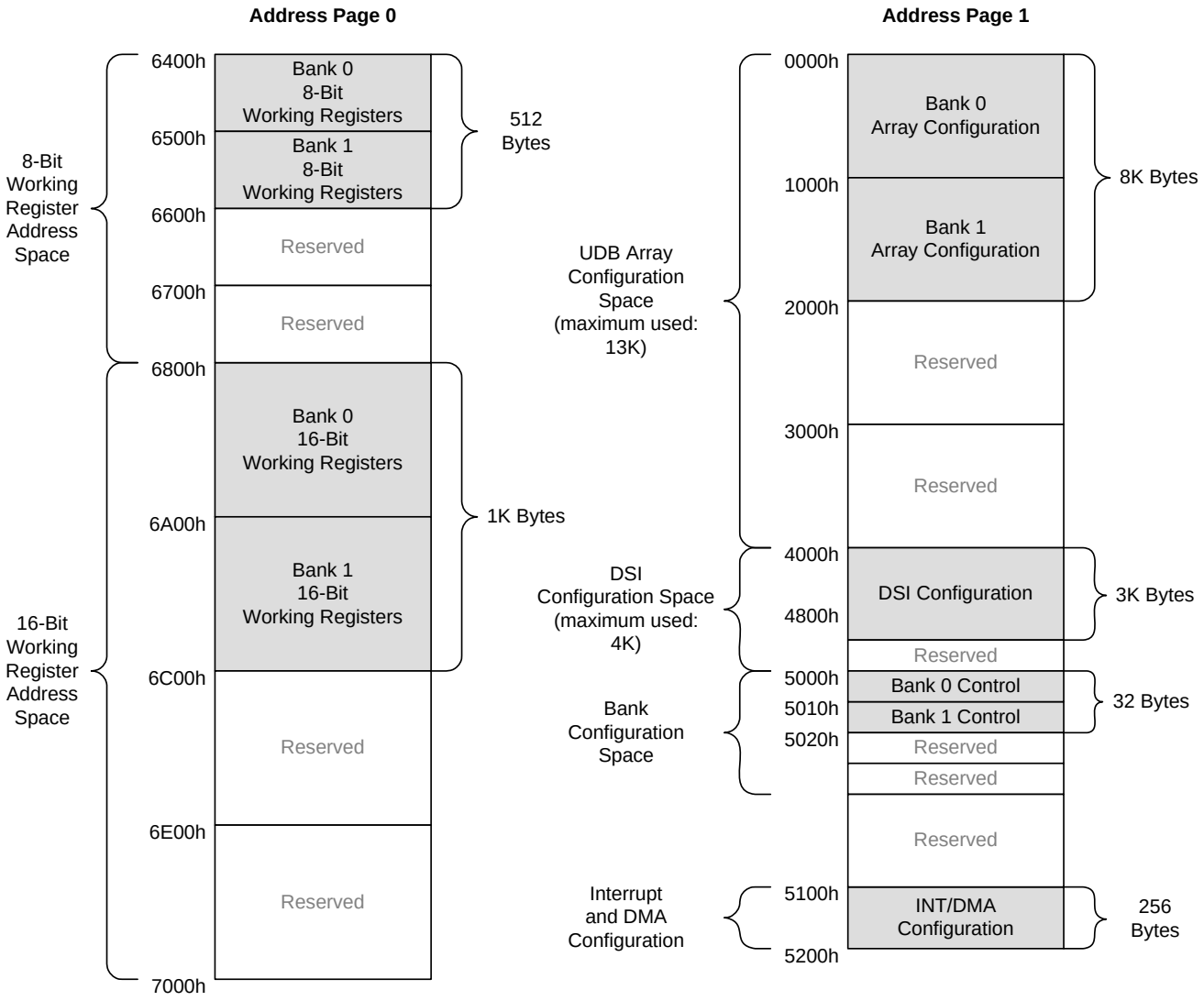


Figure 22-6 shows the register mapping for working and configuration registers of UDB and DSI.

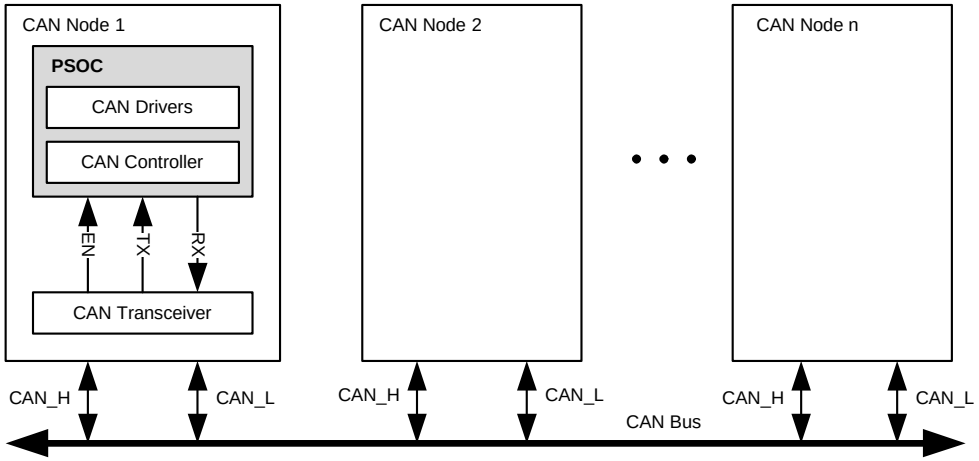
Figure 22-6. UDB Array Base Addresses



23. Controller Area Network (CAN)

The CAN peripheral is a fully functional Controller Area Network (CAN) supporting communication baud rates up to 1 Mbps. The CAN controller is CAN2.0A and CAN2.0B compliant per the ISO-11898 specification. The CAN protocol was originally designed for automotive applications with a focus on a high level of fault detection and recovery. This ensures high communication reliability at a low cost. Because of its success in automotive applications, CAN is used as a standard communication protocol for motion oriented embedded control applications (CANOpen) and factory automation applications (DeviceNet). The CAN features allow the efficient implementation of higher level protocols without affecting the performance of the microcontroller CPU.

Figure 23-1. CAN Bus System Implementation



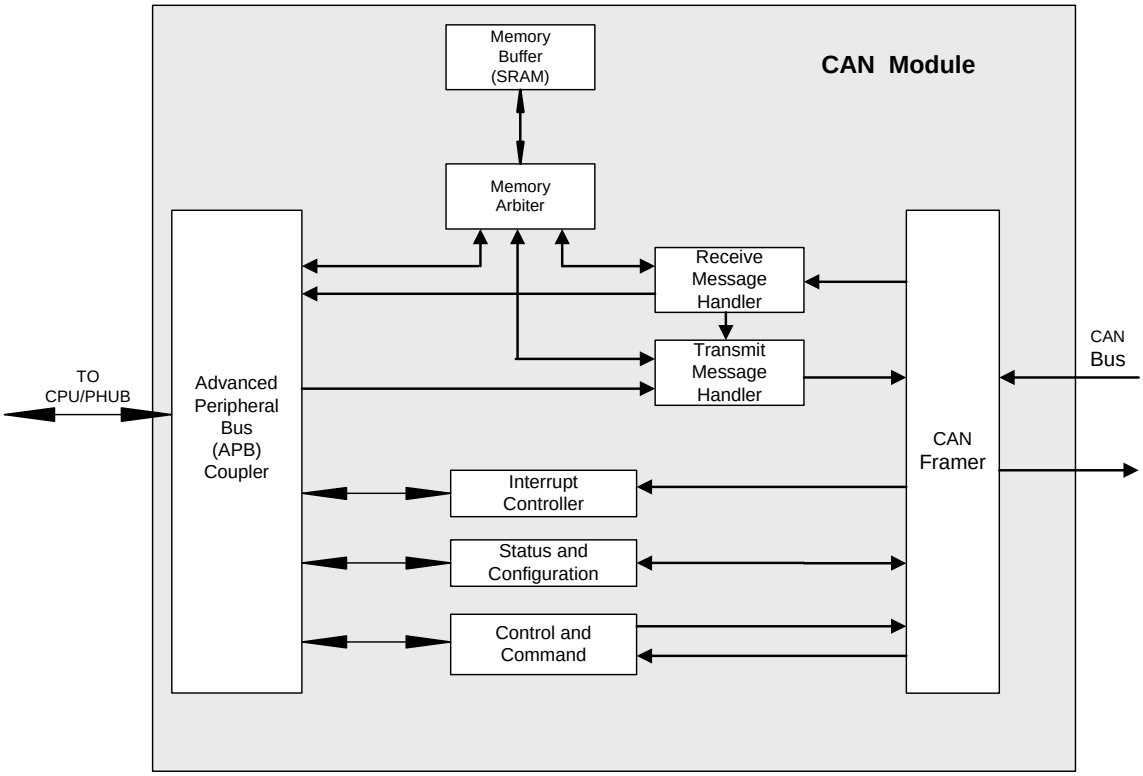
23.1 Features

- Compliant with CAN2.0A/B protocol specification:
  - Standard and extended frames
  - Remote Transmission Request (RTR) support
  - Programmable bit rate up to 1 Mbps
- Receive path:
  - 16 receive message buffers
  - 16 acceptance filters and acceptance masks
  - DeviceNet addressing support
  - Option to link multiple receive buffers to form a hardware FIFO
- Transmit path:
  - Eight transmit message buffers
  - Programmable priority for each transmit message buffer
- CAN Transmit (Tx), Receive (Rx), and EN can be routed to any I/O
- Listen Only mode for auto baud detection
- Ability to wake up the device from Sleep mode on bus activity

23.2 Block Diagram

To transmit a message, the host controller stores a message in the transmit message buffer and informs the transmit message handler which transmits the message. When a message is received, it is stored in the memory buffer and the host controller can process it on demand. The transmission and reception are mainly governed by the status and configuration registers. The various interrupts of the CAN module are handled by the interrupt controller unit. Figure 23-2 illustrates this process.

Figure 23-2. CAN Block Diagram



23.3 CAN Message Frames

In CAN the transmission and reception of messages are governed by four main frame types:

- Data frames
- Remote frame
- Error frame
- Overload frame

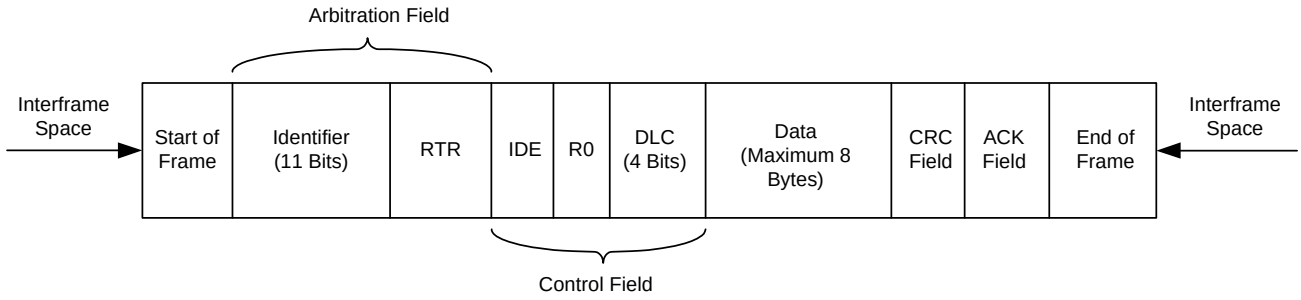
23.3.1 Data Frames

Data frames are mainly used to transfer data between transmitter and receiver. CAN supports mainly two types of data frames: Standard Data Frame and Extended Data Frame. For a CAN frame, '0' is referred to as the dominant bit and '1' as a recessive bit.

23.3.1.1 Standard Data Frame

The standard data frame for CAN is illustrated in Figure 23-3 on page 235.

Figure 23-3. Standard Data Frame



Start of frame. The beginning of a data frame is indicated by the start of frame bit. It is a single dominant bit.

Identifier. For a basic CAN data frame, the identifier is 11 bits long. It is mainly used to filter the data at the receiver side.

Remote Transmission Request Bit (RTR). Set the RTR bit '0' (dominant) for a data frame and set to '1' (recessive) for a remote frame. The identifier and RTR bit are known as the Arbitration Field.

Extended Identifier Bit (IDE). This bit must be a ‘0’ (dominant for a standard data frame and a ‘1’ (recessive) for extended CAN data frame.

R0. Reserved bit.

Data Length Code (DLC). These 4 bits indicate the number of data bytes in the data field. The IDE, R0, and DLC bits constitute the Control Field.

Data Field. This field contains the message data. It is of variable length and can have a maximum of 8 bytes.

Cyclic Redundancy Check (CRC). Frame checking is carried out by the method of cyclic redundancy check (CRC). The field consists of a 15-bit CRC code followed by a CRC delimiter.

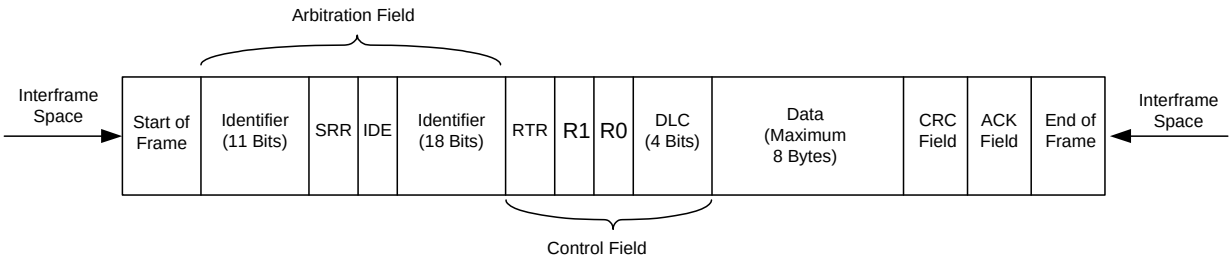
Acknowledgement Field (ACK). The ACK field is two bits long and recessive by default. When a receiver receives a message correctly, it overwrites the ACK field with a dominant bit.

End of Frame. The end of every frame is indicated by End of Frame field and it consists of seven recessive bits.

23.3.1.2 Extended Data Frame

The extended CAN frame format is illustrated in Figure 23-4. The extended CAN has a 29-bit identifier. It is arranged as an 11-bit identifier field and an 18-bit identifier field separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit is in the same position as the RTR bit in the standard frame, and is recessive. The IDE bit is set for extended frames. The Control Field of the extended data frame has an additional reserve bit ‘R1’ compared to the standard data frame.

Figure 23-4. Extended Data Frame



23.3.2 Remote Frame

The CAN bus allows a destination node to request data from the source by sending a Remote Frame. There are two differences between a Data Frame and a Remote Frame. First, the RTR bit is transmitted as a recessive bit in the remote Frame. Second, there is no Data Field in the Remote Frame.

For extended remote frame, the SRR bit is also transmitted as a recessive bit.

Interframe Space. Interframe space separates the data frames and remote frames from the preceding frames.

23.3.3 Error Frame

The Error frame is generated by a node when it detects any bus error. The error frame consists of an error flag and error delimiter. The error flag are classified into two types: error active flag and error passive flag.

Error Active Flag. When an error active station detects an error it sends six dominant bits as an active error flag. The format of the error flag thus violates the rule of bit stuffing thereby forcing all other nodes to send out error flags resulting in a series of six to twelve dominant bits on the bus.

Error Passive Flag. An error passive flag consists of six recessive bits. When an error passive station detects an error it sends a passive error flag. A passive error does not affect any other nodes and the error is detected only if the transmitting node detects a bus error. The Error Delimiter consists of eight recessive bits.

23.3.4 Overload Frame

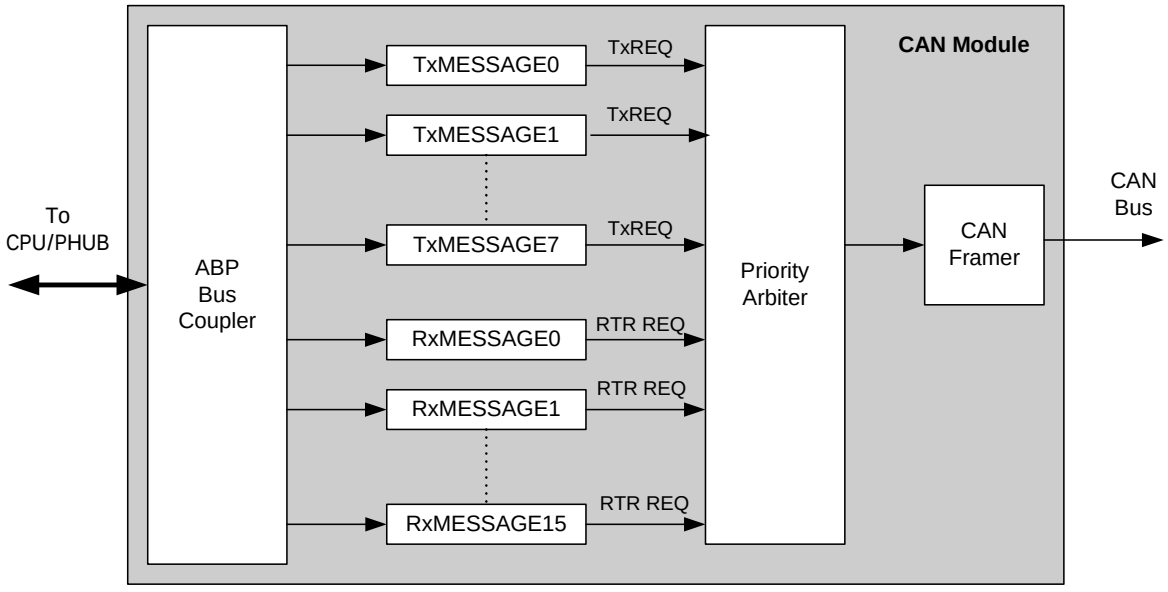
The overload frame (EOF) consists of an overload flag and an overload delimiter. CAN supports reactive overload frame which is activated when the following conditions occur:

- Detection of a dominant bit during first two bits of intermission
- Detection of a dominant bit in the last bit of EOF by a receiver
- Detection of a dominant bit by any node at the last bit of error delimiter or overload delimiter

23.4 Transmitting Messages in CAN

The CAN module supports eight transmit message holding buffers. An internal priority arbiter selects the message according to the chosen arbitration scheme. The arbitration scheme is either a round robin or fixed priority scheme. When a message is transmitted or when there is a message arbitration loss, the priority arbiter re-evaluates the message priority of the next message. The receive message buffers can also transmit remote transmit requests, which are explained later in this chapter.

Figure 23-5. Transmit (Tx) Block Diagram



23.4.1 Message Arbitration

The priority arbiter supports a round robin and fixed priority arbitration. The arbitration mode is selected using the configuration register.

Round Robin. In a round robin scheme, Buffer 0 is selected first, then Buffer 1 and so on till Buffer 7, and it continues again with Buffer 0 thus forming a cycle. A particular buffer is only selected if its TxREQ flag is set. This scheme guarantees that all buffers receive the same probability to send a message.

Fixed Priority. Buffer 0 has the highest priority. Designate Buffer 0 as the buffer for critical messages to guarantee that message is sent first. Priority arbitration is selected using the CFG_ARBITER bit in the Configuration register (CAN_CSR_CFG[12]).

Note RTR message requests are served before TxMessage buffers are handled. For example, RTRreq0, RTRreq15, TxMessage0, TxMessage1, and TxMessage7.

23.4.2 Message Transmit Process

Figure 23-6 shows the registers associated with a message that is transmitted.

The main steps in transmitting a standard data frame are:

1. Write the message into an empty transmit message holding buffer. An empty buffer is indicated by TxREQ flag equal to zero.
   a. For standard data frame, write '0' (dominant) to the RTR and IDE bit.
   b. Write the DLC bits appropriately to specify the number of data bytes to be transferred. The maximum number of data bytes is limited to eight. Data bytes with MSb (most significant bit) first in each byte are written in D0, D1…D7 locations.
   c. The 11-bit message identifiers are written to the ID[31:21] bit field.
2. Choose an appropriate priority arbitration scheme. The internal message priority arbiter selects the message according to the chosen arbitration scheme.
3. Request transmission by setting the respective TxREQ flag to ‘1’.
4. The TxREQ flag remains set as long as the message transmit request is pending. The content of the message buffer must not be changed while the TxREQ flag is set.

After the message is transmitted, the TxREQ flag is cleared and the TX_MSG interrupt status bit [CAN_CSR_INT_SR[11] in the interrupt status register CAN_CSR_INT_SR is asserted. The interrupt status bit is only asserted if the TxINT ENBL (CAN_TX[n]_CMD[2]) is set to ‘1’.

Figure 23-6. Transmit (Tx) Message Registers

REGISTERS											
COMMAND REGISTER (CAN_Txn_CMD)	Reserved [31:24]	WPN2 [23]	Reserved 1 [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:4]	WPN1 [3]	Tx INT ENBL [2]	Tx ABORT [1]	Tx REQ [0]
IDENTIFIER (CAN_Txn_ID)	ID [31:3]									Reserved [2:0]	
DATA REGISTER High (CAN_Txn_DH)	D0 [63:56]			D1 [55:48]			D2 [47:40]			D3 [39:32]	
DATA REGISTER Low (CAN_Txn_DL)	D4 [31:24]			D5 [23:16]			D6 [15:8]			D7 [7:0]	

n = 0,1,…,7

23.4.3 Message Abort

A message is aborted by setting the TxABORT flag (CAN_TX[n]_CMD[1]) in the CAN_TX[n]_CMD register. This bit is automatically cleared by the hardware when the message is aborted.

Note 1. The CAN Buffer register (CAN_CSR_BUF_SR) is used to read whether any transmission requests are pending.

Note 2. If the write protect bit wpn2 (CAN_TX[n]_CMD[23]) is ‘0’, then the bits [21:16] of the Command register cannot be modified because they are protected and provides an undefined value on read back.

Note 3. If the write protect bit wpn1 (CAN_TX[n]_CMD[3]) is ‘0’, then the bit [2] of the Command register cannot be modified. This bit gives a ‘0’ upon read back.

Note 4. Using the WPN flags(wpn1 and wpn2) enables simple retransmission of the same message by only having to set the TxREQ flag without taking care of the special flags (RTR,IDE,DLC and TxINTENBL).

23.4.4 Transmitting Extended Data Frames

For transmitting an extended data frame certain register settings must change compared to that of a standard data frame. These changes are as follows.

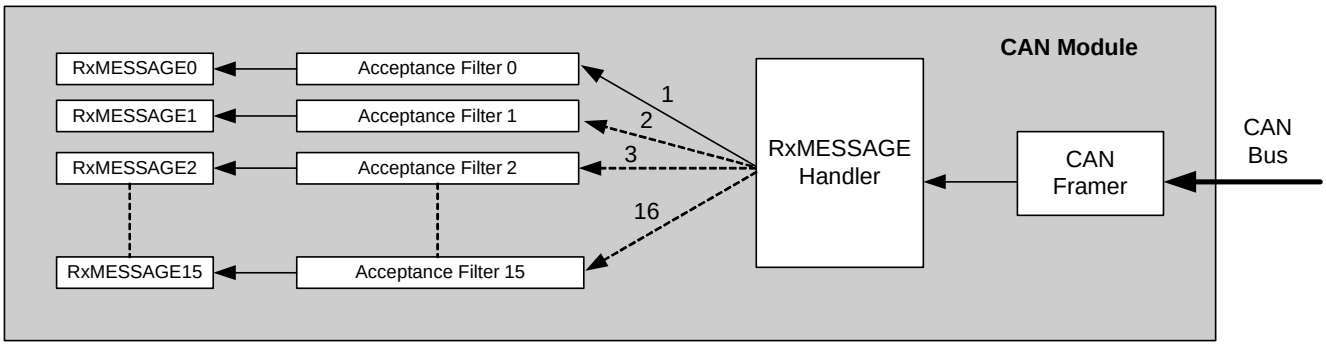
- For extended date frame, write '1' (recessive) to the IDE bit.
- The message identifiers are written to the ID[31:3] bit field.

23.5 Receiving Messages in CAN

The CAN module has 16 receive message buffers as illustrated in Figure 23-7. Each message buffer has a dedicated acceptance filter. The CAN message is received by the CAN framer and then the received message is simultaneously compared with all the acceptance filters and the accepted message is stored in the respective receive message buffer. The message available (MSG AV) bit in the message buffer is set to indicate the availability of the new message. Message receipt must be acknowledged by clearing the MSG AV flag to allow receipt of another message.

The acceptance filter is configured by the Acceptance Mask Register (AMR) and the Acceptance Code Register (ACR).

Figure 23-7. Receive (Rx) Block Diagram



23.5.1 Message Receive Process

Figure 23-8 shows the registers associated with a received message.

Figure 23-8. Receive (Rx) Message Registers

REGISTERS															
COMMAND REGISTER (CAN_Rxn_CMD)	Reserved [31:24]	WPN2 [23]	Reserved1 [22]	RTR [21]	IDE [20]	DLC [19:16]	Reserved [15:8]	WPNL [7]	LINK FLAG [6]	Rx INT ENBL [5]	RTR REPLY [4]	BUFF ENBL [3]	RTR ABORT [2]	RTR REPLY PNDG [1]	MSG AV [0]
IDENTIFIER (CAN_Rxn_ID)	ID [31:3]													Reserved [2:0]	
DATA REGISTER High (CAN_Rxn_DH)	D0 [63:56]					D1 [55:48]			D2 [47:40]				D3 [39:32]		
DATA REGISTER Low (CAN_Rxn_DL)	D4 [31:24]					D5 [23:16]			D6 [15:8]				D7 [7:0]		

n = 0,1,…,15

The main steps in receiving a message are:

1. After receipt of a new message, the RxMessageHandler hardware (as seen in Figure 23-7) searches all receive buffer starting from RxMessage0 until it finds a valid buffer. A valid buffer is indicated by:
   a. Receive buffer is enabled indicated by BUFF ENBL = ‘1’ (CAN_RX[n]_CMD[3]).
   b. Acceptance filter of the receive buffer matches incoming message.
2. If the RxMessageHandler finds a valid buffer that is empty, then the message is stored and the MSG AV flag of this buffer is set to ‘1’.
3. If the Rx INT ENBL flag is set, then the RX_MSG flag (CAN_CSR_INT_SR[12]) of the interrupt controller is asserted.
4. If the receive buffer already contains a message indicated by MSG AV = ‘1’ and the Link Flag is not set, then the RX_MSG_LOSS interrupt flag (CAN_CSR_INT_SR[10]) is asserted. The existing message is overwritten with the new received message.

Note The CAN Buffer register (CAN_CSR_BUF_SR) determines if any receive message buffer is available.

23.5.2 Acceptance Filter

Each receive buffer has its own acceptance filter that is used to filter incoming messages. An acceptance filter is configured by the Acceptance Mask register (AMR) and the Acceptance Code register (ACR).

AMR: ‘0’. The incoming bit is checked against the respective ACR bit. The message is not accepted when the incoming bit does not match respective ACR bit.

AMR: ‘1’. The incoming bit is Do Not Care.

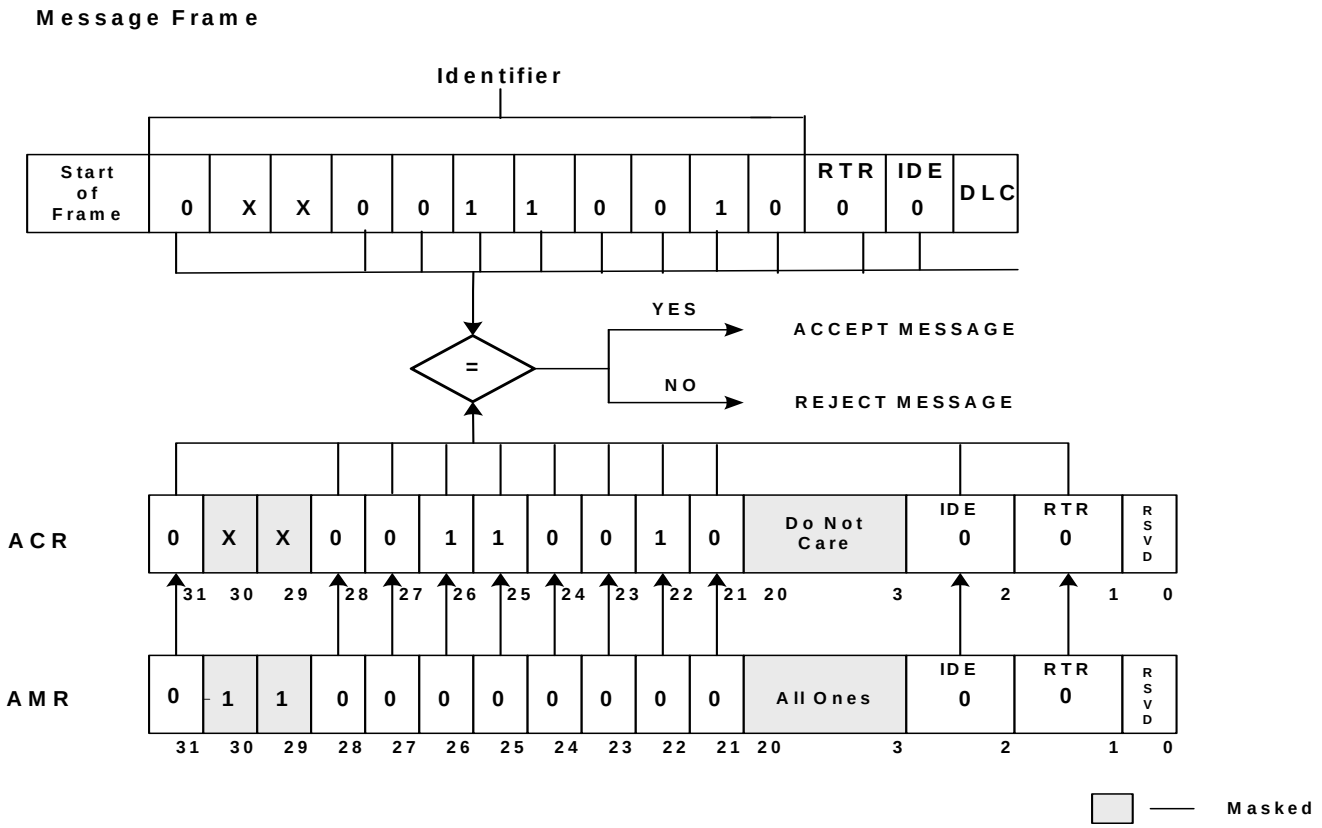
Following message fields are covered:

- Identifier
- IDE
- RTR
- Data byte 1 and data byte 2
  For a standard CAN message when IDE=0, the 11 bit identifier are the bits [31:21] of AMR and ACR.

23.5.2.1 Example

A message and the acceptance filter settings to accept that message are shown in Figure 23-9 on page 240.

Figure 23-9. Acceptance Filter



As seen in the Figure 23-9, the shaded areas are masked bits. When a bit is set to '1' in the AMR register, the corresponding bit in the ACR register is not checked against the received message frame. In the example, bits 30, 29, and bits from 3 to 20 are set to '1' and are masked. Because other bits in the AMR register are written as '0', the respective bits in the ACR register are compared with message bits as shown in Figure 23-9. If the corresponding bits in ACR match with that of the message, the message is then stored in the receive message buffer. If the corresponding bits in ACR do not match with the message, the incoming message is rejected.

AMR Settings:

ID[28:21],ID[31] = 0
ID[30],ID[29]= 1
ID[20:3] = All Ones
IDE = 0
RTR = 0

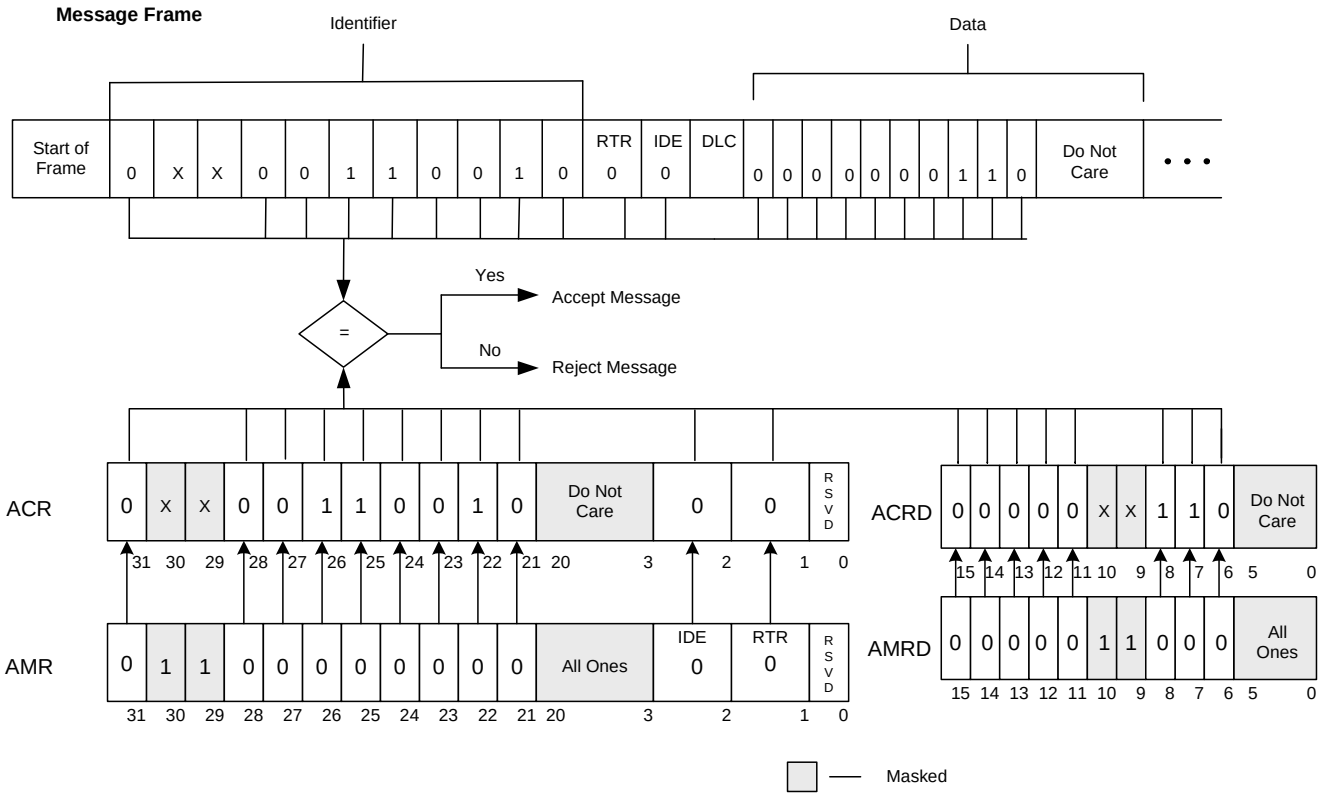
ACR Settings:

ID[31:21] = 182h
ID[20:3] = Do Not Care
IDE = 0
RTR = 0

23.5.3 DeviceNet Filtering

For some CAN high level protocols such as DeviceNet, additional protocol related information is contained in the first and second data bytes. The acceptance filters provide additional coverage of these two bytes for a more efficient implementation of the protocol. The data bits of the first two bytes of the incoming message are compared with the ACRD register (CAN_RX[n]_ACRD) and the respective bits that are compared are specified using AMRD register (CAN_RX[n]_AMRD). Using the Example on page 239, DeviceNet filtering is illustrated in Figure 23-10.

Figure 23-10. DeviceNet Filter



In Figure 23-10 the data field of the message frame is compared with those bits of the ACRD register, which are not masked by the AMRD register.

To accept this message, the acceptance filter settings are as follows.

AMR Settings:

ID[28:21],ID[31] = 0
ID[30],ID[29] = 1
ID[20:3] = All Ones
IDE = 0
RTR = 0
AMRD[15:11], AMRD[8:6] = 0
AMRD[10:9], AMRD[5:0] = All Ones

ACR Settings:

ID[31:21] = 182h
ID[20:3] = Do Not Care
IDE = 0
RTR = 0
ACRD[15:6] = 06h
ACRD[5:0] = Do Not Care

The example in Figure 23-10 shows the filtering using 10 data bits. Using AMRD, up to 16 data bits, can be used for filtering.

23.5.4 Filtering of Extended Data Frames

Filtering the extended data frame is very similar to the standard date frame with the following exception.

- IDE bit in AMR and ACR registers must be set to check for extended data frame.

23.5.5 Receiver Message Buffer Linking

Several receive buffers can link together to form a receive buffer array that acts almost like a receive FIFO. To accomplish this, do the following:

- Set the Link flag CAN_RX[n]_CMD[6] for the buffers that need to be linked.
- Make sure that all buffers of the same array have the same message filter setting (AMR and ACR are identical).
- Do not set the Link flag of the last buffer of an array.

When a receive buffer already contains a message (MSG AV='1') and a new message arrives for this buffer, then this message is discarded (RX_MSG_LOSS Interrupt). To avoid this situation, several receive buffers are linked together. When the CAN controller receives a new message, the RxMessageHandler searches for a valid receive buffer. If one is found that is already full (MSG AV = '1') and the 'Link Flag' is set, the search for a valid receive buffer is continued. If found, the message is transferred to that buffer thereby forming an array. If no other buffer is found, then the RX_MSG_LOSS interrupt is set.

It is possible to build several message arrays. Each of these arrays must use the same AMR and ACR.

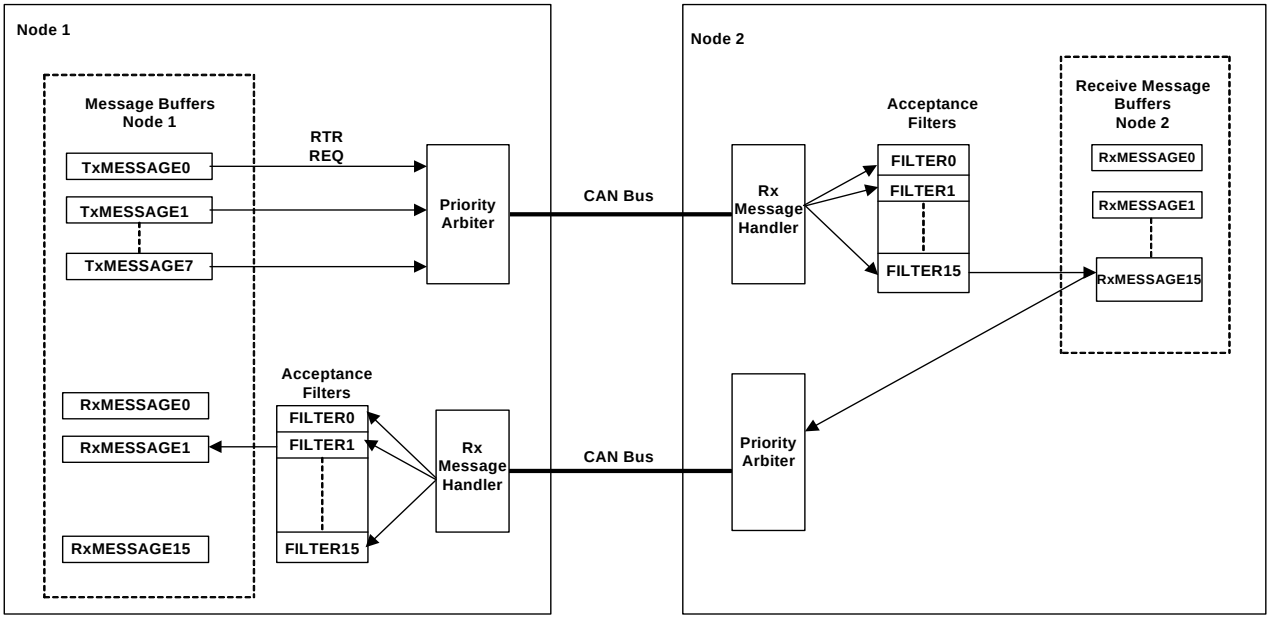
23.6 Remote Frames

Remote frames are used for initiating transmission between two nodes and the node acting as a receiver sends the remote frame. A remote frame can use either standard format or extended format. A remote frame is different from a data frame in that the RTR bit is always equal to '1' and the data field is absent, independent of the value of DLC field. The flow of a remote transmit request is illustrated in Figure 23-11.

As shown in Figure 23-11:

- The message buffer0 of node1 transmits a remote frame into the CAN bus.
- The RTR request is received by the RxMessageHandler of node 2 and sends it to the acceptance filters.
- The acceptance filter settings of the receive message buffer 15 matches with that of the message and then the message is moved to the receive message buffer 15.
- If the RTR Auto Reply feature is enabled, the receive message buffer 15 will transmit the message with the same identifier as it received (without CPU intervention).
- The acceptance filter of the receive message buffer 1 of node1 has the same identifier settings as that of the transmitted message node 1. Hence, the RTR message will be stored in the receive message buffer 1 of node 1.

Figure 23-11. Remote Transmit Request



23.6.1 Transmitting a Remote Frame by the Requesting Node

The process to transmit a remote frame by a requesting node (Node 1 as shown in Figure 23-11 on page 242) is as follows.

1. Write a message to an empty transmit buffer. An empty buffer is indicated by Tx_REQ = ‘0’ (CAN_TX[n]_CMD[0]).
2. Set the RTR bit (CAN_TX[n]_CMD[21]) to ‘1’.
3. Choose an appropriate priority arbitration scheme.
4. Set the transmit request flag to initiate transmission.
5. The Identifier transmitted in a message must be the same as the identifier of receiving message.

23.6.2 Receiving a Remote Frame

The process to receive a remote frame is as follows.

1. The acceptance filter must be configured to receive the desired message ID.
2. Enable the automatic RTR message handling by setting bit ‘RTR REPLY’ to ‘1’.
   a. If enabled, it will automatically transmit the remote frame with the same identifier.
   b. Else the remote frame must be transmitted following the standard routine as that of a data frame.
3. Set the requesting node that receives the replied RTR message to receive a normal message. Do not set the RTR Reply bit.

23.6.3 RTR Auto Reply

The CAN module supports automatic answering of RTR message requests. All 16 receive buffers support this feature. If an RTR message is accepted in a receive buffer where the RTR REPLY FLAG is set, then this buffer automatically replies to this message with the content of the receive buffer. The ‘RTR REPLY PNDG FLAG’ is set when the RTR message request is received. It is reset when the message is sent or when the message buffer is disabled. To abort a pending RTRreply message, use the RTR ABORT-command.

23.6.4 Remote Frames in Extended Format

The transmission and reception of remote frames in extended format is similar to standard format except for the following.

- The IDE bit (CAN_TX[n]_CMD[20]) is set to ‘1’ to make it an extended data frame.
- The identifier is 29 bits long compared to the 11 bits of a standard data frame.

23.7 Bit Time Configuration

The CAN module operates on a single clock input CLK_BUS. This section explains how to configure the programmable bit-rate divider to achieve the desired bit rate and its relationship with CLK_BUS.

23.7.1 Allowable Bit Rates and System Clock (CLK_BUS)

Across the industry, most implementations of CAN-Bus use one of 10 bit rates:

- 1 Mbps
- 800 Kbps
- 500 Kbps
- 250 Kbps
- 125 Kbps
- 100 Kbps
- 50 Kbps
- 20 Kbps
- 10 Kbps
- 5 Kbps

These bit rates are configurable if CLK_BUS is 8 MHz or a multiple. All except 800 Kb are configurable if CLK_BUS is 10 MHz or a multiple. With a very few exceptions, all 10 bit rates are not possible if CLK_BUS is not evenly divisible by 1,000,000 Hz. From a bit rate generation point of view, the accuracy for CLK_BUS must be at least 1.58% for 125 Kbps and slower bit rates, and 0.5% or better for bit rates faster than 125 Kbps. Figure 23-12 on page 244 shows a table of the 10 bit rates that are supported for any given fclk frequency from 8 MHz to 100 MHz. Note that maximum possible frequency for PSoC 3 is 67 MHz.

Figure 23-12. Bit Rate Versus CLK_BUS

clk_bus Freq (MHz)	1 Mb	800 Kb	500 Kb	250 Kb	125 Kb	100 Kb	50 Kb	20 Kb	10 Kb	5 Kb
8	■	■	■	■	■	■	■	■	■	■
9					■	■	■	■	■	■
10	■		■	■	■	■	■	■	■	■
11					■	■	■	■	■	■
12			■	■	■	■	■	■	■	■
13					■	■	■	■	■	■
14				■	■	■	■	■	■	■
15			■	■	■	■	■	■	■	■
16	■	■	■	■	■	■	■	■	■	■
17					■	■	■	■	■	■
18				■	■	■	■	■	■	■
19					■	■	■	■	■	■
20	■		■	■	■	■	■	■	■	■
21					■	■	■	■	■	■
22				■	■	■	■	■	■	■
23					■	■	■	■	■	■
24	■	■	■	■	■	■	■	■	■	■
25			■	■	■	■	■	■	■	■
26				■	■	■	■	■	■	■
27					■	■	■	■	■	■
28			■	■	■	■	■	■	■	■
29					■	■	■	■	■	■
30	■		■	■	■	■	■	■	■	■
31					■	■	■	■	■	■
32	■	■	■	■	■	■	■	■	■	■
33					■	■	■	■	■	■
34				■	■	■	■	■	■	■
35			■	■	■	■	■	■	■	■
36			■	■	■	■	■	■	■	■
37					■	■	■	■	■	■
38				■	■	■	■	■	■	■

clk_bus Freq (MHz)	1 Mb	800 Kb	500 Kb	250 Kb	125 Kb	100 Kb	50 Kb	20 Kb	10 Kb	5 Kb
39					■	■	■	■	■	■
40	■	■	■		■	■	■	■	■	■
41					■	■	■	■	■	■
42				■	■	■	■	■	■	■
43					■	■	■	■	■	■
44			■	■	■	■	■	■	■	■
45			■	■	■	■	■	■	■	■
46				■	■	■	■	■	■	■
47					■	■	■	■	■	■
48	■	■	■	■	■	■	■	■	■	■
49					■	■	■	■	■	■
50	■		■	■	■	■	■	■	■	■
51					■	■	■	■	■	■
52			■	■	■	■	■	■	■	■
53					■	■	■	■	■	■
54				■	■	■	■	■	■	■
55			■	■	■	■	■	■	■	■
56	■	■	■	■	■	■	■	■	■	■
57					■	■	■	■	■	■
58				■	■	■	■	■	■	■
59					■	■	■	■	■	■
60	■		■	■	■	■	■	■	■	■
61					■	■	■	■	■	■
62				■	■	■	■	■	■	■
63					■	■	■	■	■	■
64	■	■	■	■	■	■	■	■	■	■
65			■	■	■	■	■	■	■	■
66				■	■	■	■	■	■	■
67					■	■	■	■	■	■
68			■	■	■	■	■	■	■	■
69					■	■	■	■	■	■

clk_bus Freq (MHz)	1 Mb	800 Kb	500 Kb	250 Kb	125 Kb	100 Kb	50 Kb	20 Kb	10 Kb	5 Kb
70	■		■	■	■	■	■	■	■	■
71					■	■	■	■	■	■
72	■	■	■	■	■	■	■	■	■	■
73					■	■	■	■	■	■
74				■	■	■	■	■	■	■
75			■	■	■	■	■	■	■	■
76			■	■	■	■	■	■	■	■
77					■	■	■	■	■	■
78				■	■	■	■	■	■	■
79					■	■	■	■	■	■
80	■	■	■	■	■	■	■	■	■	■
81					■	■	■	■	■	■
82				■	■	■	■	■	■	■
83					■	■	■	■	■	■
84			■	■	■	■	■	■	■	■
85			■	■	■	■	■	■	■	■
86				■	■	■	■	■	■	■
87					■	■	■	■	■	■
88	■	■	■	■	■	■	■	■	■	■
89				■	■	■	■	■	■	■
90	■		■	■	■	■	■	■	■	■
91					■	■	■	■	■	■
92			■	■	■	■	■	■	■	■
93				■	■	■	■	■	■	■
94				■	■	■	■	■	■	■
95			■	■	■	■	■	■	■	■
96	■	■	■	■	■	■	■	■	■	■
97				■	■	■	■	■	■	■
98				■	■	■	■	■	■	■
99					■	■	■	■	■	■
100	■		■	■	■	■	■	■	■	■

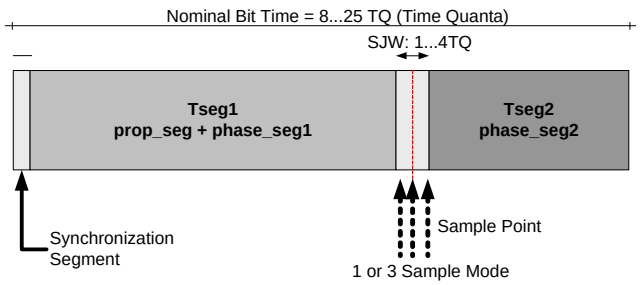
■ Configurable Bit Rates

□ Non Configurable Bit Rates

23.7.2 Setting Bit Rate TSEG1 and TSEG2

The bit rate is defined as the number of bits transmitted on a CAN bus per second. Bit time is the reciprocal of bit rate. Bit time is divided into three segments as shown in Figure 23-13. Each segment is represented in terms of fixed units of time called Time Quanta (TQ) which is derived from the oscillator clock.

Figure 23-13. Bit Time



$$BitTime = (1 + (tseg1 + 1) + (tseg2 + 1)) \times TQ$$

Equation 1

$$TQ = \frac{BRP + 1}{\text{clk\_bus}}$$

Equation 2

Note Bit rate pre scaler is a register that performs a pre scaling function on CLK_BUS to generate the clock for CAN module. See Figure 23-14.

Synchronization Segment. This is the first segment with 1 TQ length and is mainly used for synchronization. An edge is expected to fall within this segment.

Tseg1, Tseg2. These segments compensate for the edge phase shift errors. The tseg1 also takes in the propagation time which includes any delays in the network. The length of the segments is increased or decreased to compensate for the error due to phase shift of edges which is known as resynchronization.

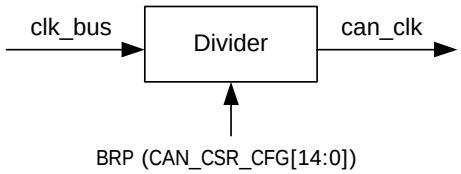
Sample Point. This is the point at which the state of the bus is read and the bit is interpreted. It is located at the end of tseg1.

Synchronization Jump Width. By resynchronization, the tseg1 is lengthened or tseg2 is shortened. Synchronization jump width puts a limit to this resynchronization. The length

of tseg2 must be greater than the synchronization jump width.

The Configuration register CAN_CSR_CFG is used for setting the bit rate prescaler (BRP), tseg1, tseg2, and the synchronization jump width. CAN peripheral clock (CAN_CLK) is generated by dividing the system clock (CLK_BUS) by (BRP+1). See the Clocking section for detailed information on available options to generate the system clock. For N time quanta in a bit time, the CAN peripheral clock frequency must be configured to N time the CAN bus bit rate.

Figure 23-14. Bit Timing Block Diagram



23.7.2.1 Example

An example to achieve 1 Mbps speed with 40 MHz is described as follows.

1. The speed is 1 MHz and the bit time is 1 µs.
2. Choosing a minimum value of 8 TQ in the bit time, 1TQ = 0.125 µs.
3. BRP = ((time quanta * clk_bus) – 1) = 4.
4. Therefore write a value of ‘4’ into the CFG_BITRATE bits in the configuration register.
5. Choose the sampling point to be 60% of the bit time, which is approximately equal to 5TQ. Because the sampling point is at the end of tseg1, this implies that (tseg2+1) = 3TQ or tseg2 = 2TQ.
6. To fix the sampling point synchronization jump width, use a value ‘1’ by writing to the bits CFG_SJW = ‘1’.
7. Write to the bits cfg_tseg2 a value of ‘2’ to set the value of tseg2 to 2TQ.
8. Now tseg1 is calculated using the following equation:

   tseg1 = ((BitTime - (1TQ + tseg2 + 1TQ)) - 1TQ)

   ...which is tseg1 = 3TQ.
9. Therefore, write a value of ‘3’ into the bits cfg_tseg1 in the configuration register.

This procedure above is applied to achieve the standard bit rates using the clock frequencies as specified in the table in Figure 23-12 on page 244.

Observe the following conditions for setting tseg1 and tseg2:

- tseg1 = 0 or tseg1 = 1 are not allowed.
- tseg2 = 0 is not allowed; tseg2 = 1 is only allowed in direct sampling mode.

Note 1. Sampling_mode bit in the Configuration register (CAN_CSR_CFG) specifies whether or not one sampling point is used in the receiver path or three sampling points with majority decision are used.

Note 2. Edge_mode bit in the Configuration register (CAN_CSR_CFG) specifies whether or not the high to low edge is used for synchronization or both edges are used.

23.8 Error Handling and Interrupts in CAN

According to the CAN protocol specification, there are five different types of errors. Each CAN node in the bus tries to detect an error, and when it does, it sends out an error frame. The different types of errors and the process of error handling are explained in the following sections.

23.8.1 Types of Errors

23.8.1.1 BIT Error

A CAN unit sending a bit on the bus also monitors the bus. When the bit value that is monitored is different from the bit value that is sent, a BIT error is detected. An exception is the sending of a ‘recessive’ bit during the stuffed bit stream of the Arbitration Field or during the ACK Slot. A Transmitter sending a Passive Error Flag and detecting a ‘dominant’ bit does not interpret this as a BIT error.

23.8.1.2 FORM Error

A FORM error is detected when there is an error in the CAN message format. The fixed format fields in the message frame such as End of Frame, Interframe Space, etc., contains illegal bits.

23.8.1.3 ACKNOWLEDGE Error

A transmitter sending a recessive bit during the ACK slot monitors the ACK slot for a dominant bit. If a receiver receives a message correctly, a dominant bit is written in the ACK slot. Therefore, if the transmitter does not find a dominant bit in the ACK slot after transmission, then an ACKNOWLEDGE error is detected.

23.8.1.4 CRC Error

A transmitting node performs certain calculations to generate a CRC code and transmits it in the CRC field. A receiving node also performs the same calculations to generate a CRC code. If the code generated by the receiver does not match the code transmitted then a CRC error is detected.

23.8.1.5 STUFF Error

When there are six consecutive equal bit levels in a message field that is coded by the message of bit stuffing, a STUFF error is detected during the bit time of the sixth consecutive bit level.

23.8.2 Error States in CAN

There are three main error states in CAN:

Error Active. An error active node can take part in normal bus communication. When it detects an error it sends out an ERROR ACTIVE FLAG.

Error Passive. An error passive node takes part in bus communication. When it detects an error it sends out an ERROR PASSIVE FLAG. After sending out the ERROR PASSIVE FLAG, it waits before proceeding with further transmission. An error passive node sends additional 8 recessive bits during the interframe space. This period is also known as suspend transmission because no transmission takes place.

Bus Off. A node that is in Bus Off does not take part in any bus communication. It has no effect on the bus.

The error status in CAN is indicated by the error status register CAN_CSR_ERR_SR. The bits ERR_STATE (CAN_CSR_ERR_SR[17:16]) indicate which error state the CAN node is in. The error states in CAN are determined according to the values of two counters:

- Transmit Error Counter (CAN_CSR_ERR_SR[7:0])
- Receive Error Counter (CAN_CSR_ERR_SR[15:8])

The error counters are modified according to the CAN 2.0B Specification.

A node is in ‘error active’ state if the Transmit Error Counter or the Receive Error Counter are less than or equal to 127 decimal. A node is in ‘error passive’ state if the Transmit or Receive Error Counter value exceeds or equals 128 decimal. A node is in ‘Bus Off’ state if the Transmit Error Counter exceeds or equals the value of 256 decimal.

An ‘error passive’ node becomes ‘error active’ again when both the Transmit Error Count and the Receive Error Count are less than or equal to 127.

A node which is in ‘Bus Off’ state becomes ‘error active’ with its error counters both set to ‘0’ after 128 occurrences of 11 consecutive ‘recessive’ bits are monitored on the bus.

There are two bits in the error status register:

- ‘txgte96’ (CAN_CSR_ERR_SR[18])
- ‘rxgte96’ (CAN_CSR_ERR_SR[19])

that indicate if the Transmit Error Counter and Receive Error Counter, respectively, are greater than or equal to 96 decimal. This is a feature. It serves as an error warning because an error count value greater than and around 96 indicates a heavily disturbed bus.

23.8.3 Interrupt Sources in CAN

The interrupt controller governs the various interrupt sources in CAN.

The interrupt controller contains an interrupt status and an interrupt enable register. The interrupt status register (CAN_CSR_INT_SR) stores internal interrupt events. When a bit is set, it remains set until it is cleared by writing a '1' to it. The interrupt enable register has no effect on the interrupt status register.

The interrupt enable register (CAN_CSR_INT_EN) controls which particular bits from the interrupt status register are used to assert the interrupt output INT_N. INT_N is asserted if a particular interrupt status bit and the respective enable bit are set.

The various interrupt sources in CAN are as follows:

rx_msg. Indicates a message received.

tx_msg. Indicates a message sent.

rx_msg_loss. Is set when a new message arrives but the RxMessage flag MSG AV is set.

bus_off. The CAN has reached the bus off state.

crc_err. A CAN CRC error detected.

form_err. A CAN message format error detected.

ack_err. A CAN message acknowledge error detected.

stuff_err. A bit stuffing error detected.

bit_err. A bit error detected.

ovr_load. An overload frame received.

arb_loss. The arbitration lost while sending a message.

23.9 Operating Modes in CAN

The CAN module operates mainly in three different modes. The command register CAN_CSR_CMD is used to select the operating modes by setting the corresponding bit for each mode. The three operating modes are as follows:

- SRAM Test Mode: CAN_CSR_CMD[2]
- Listen Only Mode: CAN_CSR_CMD[1]
- Run/Stop Mode: CAN_CSR_CMD[0]

23.9.1 Listen Only Mode

In Listen Only mode, the CAN controller only listens to the CAN receive line without acknowledging the received messages on the bus. It does not send any messages in this mode. However, the error flags are updated so that the bit timing is adjusted until no error occurs.

The various steps involved in automatic baud rate detection are as follows.

1. The CAN controller is initialized for acceptance of all messages (i.e., the global/local mask is set to ‘0’).
2. The bit timing values of the first possible CANOpen bit rate (10 Kbps) is loaded and the controller is switched into “Listen Only” mode.
3. Assuming that there is traffic on the network and the bit rate is correct, the message is accepted.
4. The error registers will not change and the flag for message reception is set inside the CAN controller. This means the correct bit rate is detected.
5. Assuming the bit rate is not correct, the error flags are updated (stuff-, CRC, or form-error).
6. In this scenario, the CAN controller is switched off and the next possible bit timing values are loaded from the bit rate table.

23.9.2 Run/Stop Mode

The CAN controller is in Run mode when it is operating normally. The CAN controller is stopped while it is in the SRAM Test mode.

24. 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.

24.1 Features

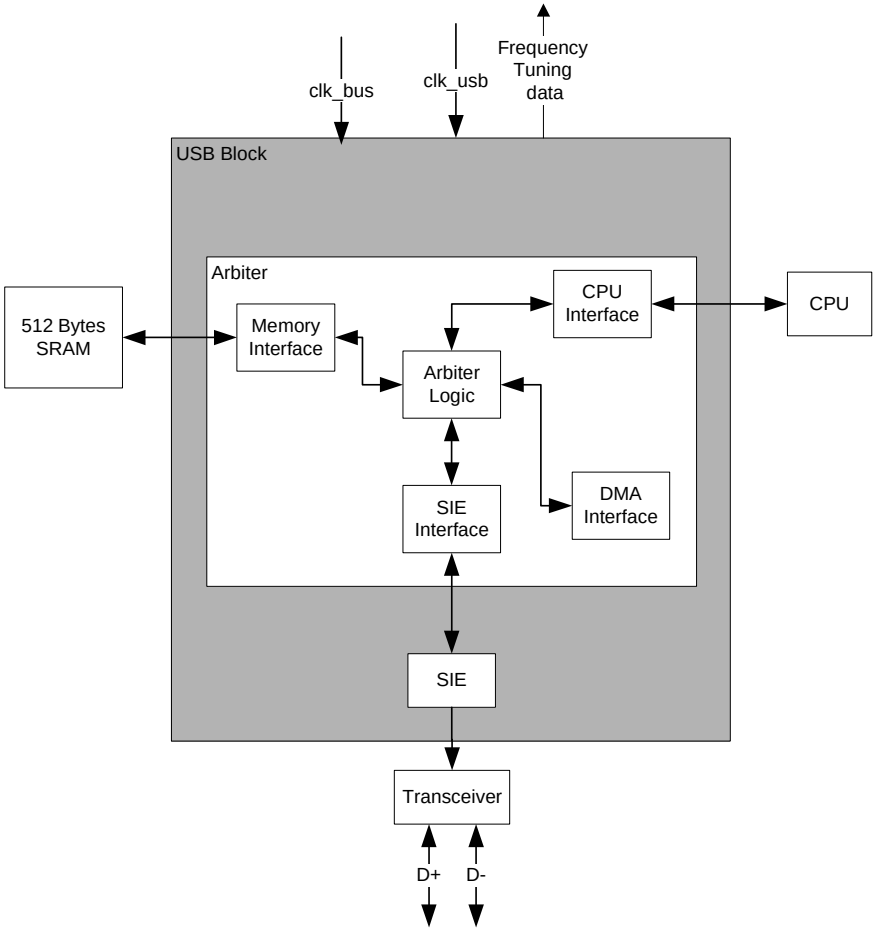
The PSoC USB has these features:

- Complies with USB Specification 2.0
- Supports full speed peripherals device operation with a signaling bit rate of 12 Mbps
- Supports 8 data endpoints and 1 control endpoint
- Supports four types of transfers – bulk, interrupt, isochronous, and control
- Supports Plug and Play
- Supports two types of logical transfer modes:
  - Store and Forward mode
  - Cut Through mode
- Differential signal (D+ and D-) output
- Supports maximum packet size of 64 bytes using the Store and Forward mode and maximum packet size of 1023 for Isochronous transfer using the Cut through mode
- Capable of supplying PS/2 and CMOS signals
- Supports two Vccd voltage ranges, with a nominal voltage of 3.3 V

24.2 Block Diagram

Figure 24-1 illustrates the architecture of the USB block. It consists of the Serial Interface Engine (SIE) and Arbiter.

Figure 24-1. USB Block Diagram



24.2.1 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:

- Conforms to 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. See Table 24-1 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 and the USB block is in Store and Forward mode, the hardware automatically retransmits the same data when it receives another IN token from the host. In Cut Through mode, this bit can be read by the firmware to determine if the data should be retransmitted.

Table 24-1. 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

Table 24-1. Mode Values in the MODE bits of the SIE_EPx_CR0 Register *(continued)*

Mode	Encoding	SETUP	IN	OUT	Comments
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

24.2.2 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).

24.2.2.1 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.

24.2.2.2 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.

24.2.2.3 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.

24.2.2.4 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.

24.2.2.5 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 memory management is either “Manual” or “Automatic.” In the case of Manual Memory Management, the read and write address manipulations are done by the firmware. In the case of Automatic management, all the memory handling is done by this block itself. 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.

24.2.2.6 Synchronization Block

The USB block uses 2 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.

24.3 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.

24.3.1 Operating Frequency

The USB block needs two different clocks to work: the System Clock which controls the Arbiter, memory and the register block, and the USB clock which controls the SIE and the OsClock.

- Minimum system clock – 24 MHz
- USB Clock for Full Speed operation – 48 MHz (+0.25% tolerance)

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 on the clock for this block, see the Clocking System chapter on page 123. 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 as per 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.

24.3.2 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.

24.3.3 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–.

24.3.4 Endpoints

The SIE and Arbiter support 8 data endpoints (EP1 to EP8) and one control endpoint (EP0). The data endpoints share the SRAM memory area of 512 bytes. The endpoint memory management can be either “Manual” or “Automatic.” The endpoints are configured for direction and other configuration using the SIE and arbiter registers. The endpoint “read address” and “write address” registers are accessed through the Arbiter. 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. In the Auto Management mode, the register USB_EP_ACTIVE is written to control the active state of the endpoint. The endpoint activation cannot be dynamically changed during runtime. In Manual Memory Management mode the firmware decides the memory allocation, so it is not required to specify the active endpoints. The EP_ACTIVE register is ignored during the manual memory management mode. The USB_EP_TYPE register is used to control the transfer direction (IN, OUT) for the endpoints. The control endpoint has a separate 8 bytes for its data.

24.3.5 Transfer Types

The PSoC USB supports Full Speed transfers and is compliant with the USB 2.0 Specification. It supports four types of transfers:

- Interrupt Transfer
- Bulk Transfer
- Isonchronous Transfer
- Control Transfer

For further details about these transfers, refer to the USB Specification 2.0.

24.3.6 Interrupts

The interrupts are generated by the SIE and the 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 the SIE and the 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.

24.4 Logical Transfer Modes

The USB block in PSoC devices supports two types of logical transfers. The logical transfers can be configured using the register setting for each endpoint. Any of the logical transfer methods can be adapted to support the three types of data transfers (Interrupt, Bulk, and Isochronous) mentioned in the USB 2.0 Specification. The Control transfer is mandatory in any USB device.

The logical transfer mode is a combination of memory management and DMA configurations. The Logical Transfer modes are related to the data transfer within the USB block (i.e., to/ from the SRAM memory unit for each endpoint). It does not represent the transfer methods between the device and the host (i.e., the transfer types specified in the USB 2.0 Specification).

The USB block supports two basic types of transfer modes and are detailed in Table 24-2 on page 255.

- Store and Forward mode
- Cut Through mode

Table 24-2. USB Transfer Modes

Feature	Store and Forward Mode	Cut Through Mode
SRAM Memory Usage	Requires more memory	Requires less memory
SRAM Memory Management	Manual	Auto
SRAM Memory Sharing	512 bytes of SRAM shared between endpoints. Sharing is done by firmware.	Each endpoint is allocated less share of memory automatically by the block. Rest of memory is available as “Common Area.” This Common Area is used during the transfer.
IN Command	Entire packet present in SRAM memory before the IN command is received.	Memory filled with data only when SRAM IN command is received. Data is given to host when enough data is available (based on DMA configuration). Does not wait for the entire data to be filled.
OUT Command	Entire packet is written to SRAM memory on OUT command. After entire data is available, it is copied from SRAM memory to the USB device.	Waits only for enough bytes (depends on DMA configuration) to be written in SRAM memory. When enough bytes are present, it is immediately copied from SRAM memory to the USB device.
Transfer of Data	Data is transferred when all bytes are written to the memory.	Data is transferred when enough bytes are available. It does not wait for the entire data to be filled.
Types Based on DMA	No DMA mode Manual DMA mode	Only Auto DMA mode
Supported Transfer Types	Best suited for Interrupt and Bulk transfers	Best suited for Isochronous transfer

Every endpoint has a set of registers that need to be handled during the modes of operation, as detailed in Table 24-3.

Table 24-3. Endpoint Registers

Register	Comment	Content	Usage
ARB_RWx_WA	Endpoint Write Address register	Address of the SRAM	This register indicates the SRAM location to which the data in the Data register is to be written.
ARB_RWx_RA	Endpoint Read Address register	Address of the SRAM	This register indicates the SRAM location from which the data must be read and stored to the Data register.
ARB_RWx_DR	Endpoint Data Register	8-Bit Data	Data register is read/ written to perform any transaction. IN command: Data written to the Data register is copied to the SRAM location specified by the WA register. After write, the WA value is automatically incremented to point to the next memory location. OUT command: Data available in the SRAM location pointed by the RA register is read and stored to the DR. When the DR is read, the value of RA is automatically incremented to point to the next SRAM memory location that must be read.
SIE_EPx_CNT0 and SIE_EPx_CNT1	Endpoint Byte Count Register	Number of Bytes	Holds the number of bytes that can be transferred. IN command: Holds the number of bytes to be transferred to host. OUT command: Holds the maximum number of bytes that can be received. The firmware programs the maximum number of bytes that can be received for that endpoint. The SIE updates the register with the number of bytes received for the endpoint.
“Mode” bits in SIE_EPx_CR0	Mode Values	Response to the Host	Controls how the USB device responds to the USB traffic and the USB host. Some examples of mode include ACK, NAK, STALL, etc. See Table 24-1 on page 251 for additional details.

In the Manual Memory Management case, 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 Automatic memory management case, the endpoint read and endpoint write address registers are updated by the USB block. The block assigns memory to the endpoints that are activated using the EP_ACTIVE register. The size of memory allocated depends on the value in the BUF_SIZE register. The rest of the memory, after allocation, is called the “Common Area” memory and used for the transfer of data.

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 3). 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.

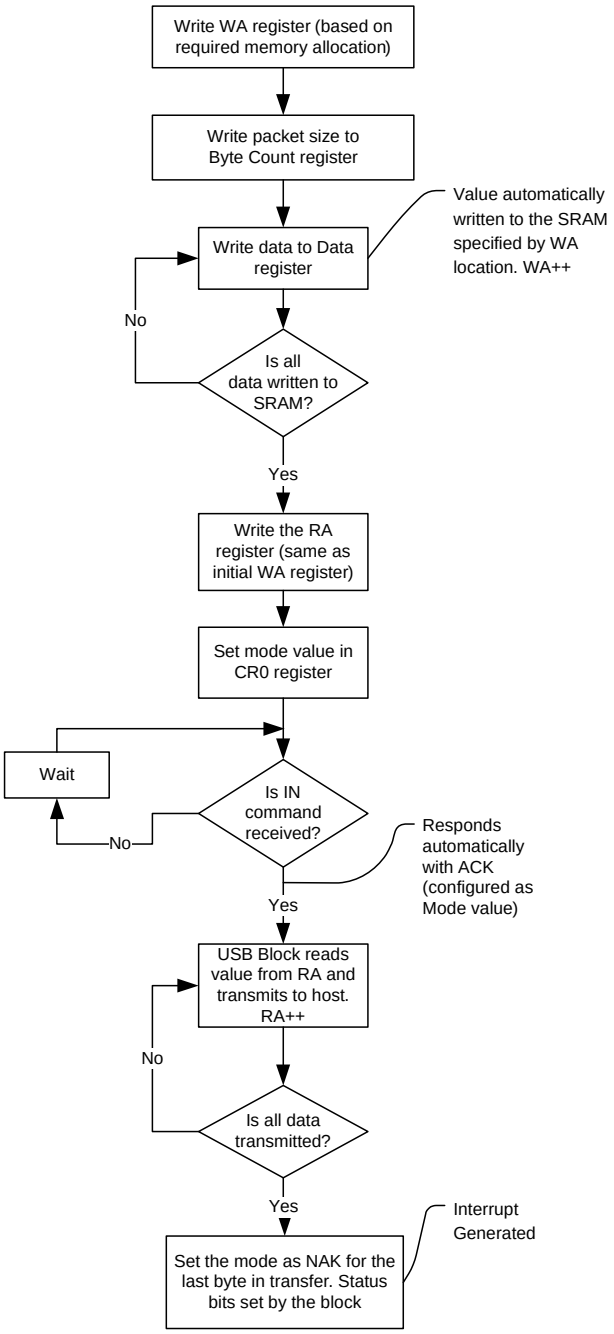
24.4.1 Store and Forward Mode

24.4.1.1 No DMA Access

This is the Manual Memory Management mode with no DMA access.

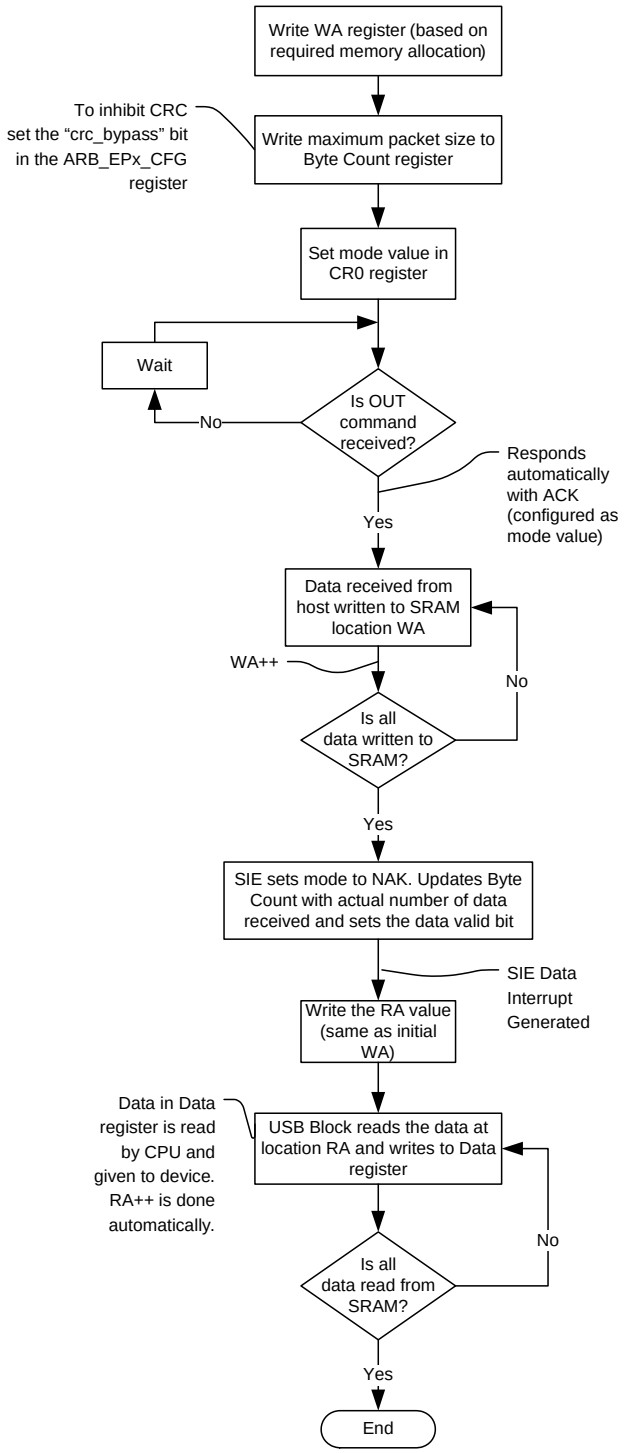
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 24-2.

Figure 24-2. 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 24-3.

Figure 24-3. No DMA Access OUT Transaction



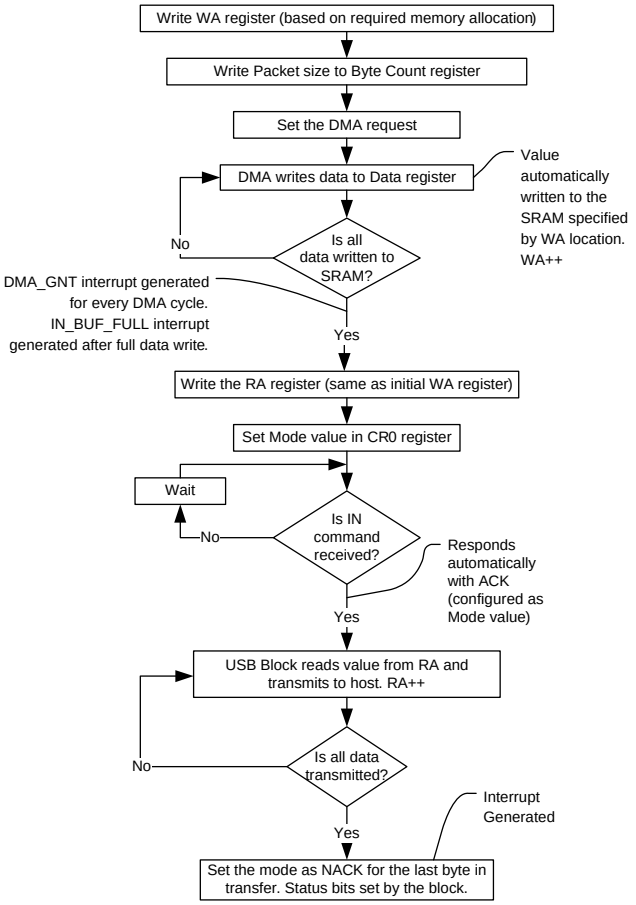
24.4.1.2 Manual DMA Access

This is the Manual Memory Management mode with Manual DMA Access. This mode requires the configuration of the DMA controller. See DMA Interface on page 252 for details and constraints regarding 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.

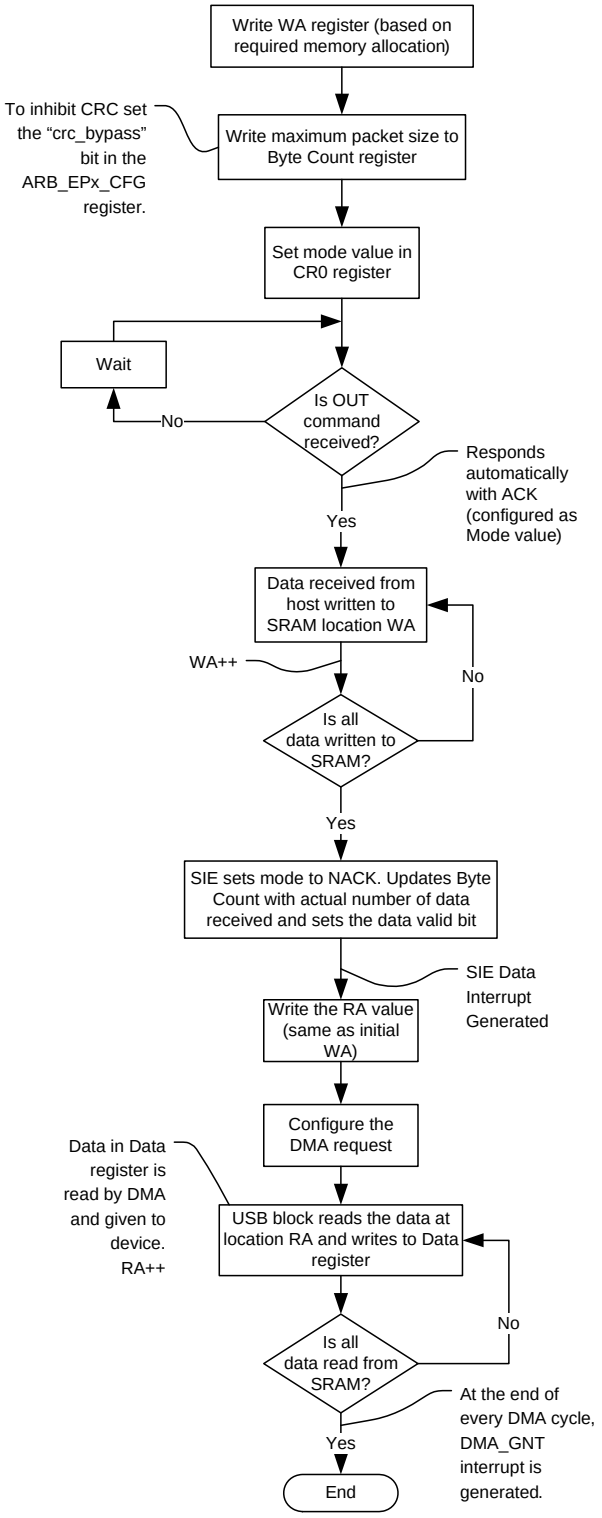
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 24-4.

Figure 24-4. Manual DMA IN Transaction



OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in Figure 24-5.

Figure 24-5. Manual DMA OUT Transaction



24.4.2 Cut Through Mode

This is the Auto Memory Management mode with Auto DMA Access. The CPU programs the initial buffer size requirement for IN/OUT packets and informs the Arbiter block of the endpoint configuration details for the particular application being considered. The block then controls memory partitioning and handling of all memory pointers. During the memory allocation, each active IN endpoint (set by the EP_ACTIVE and EP_TYPE registers) is allocated a small amount of memory configured using the BUF_SIZE register. The remaining memory is left as “Common Area” and is common for all endpoints.

In this mode, the memory requirement is less and it is suitable for the Full Speed “Isochronous Transfer” up to 1023 bytes.

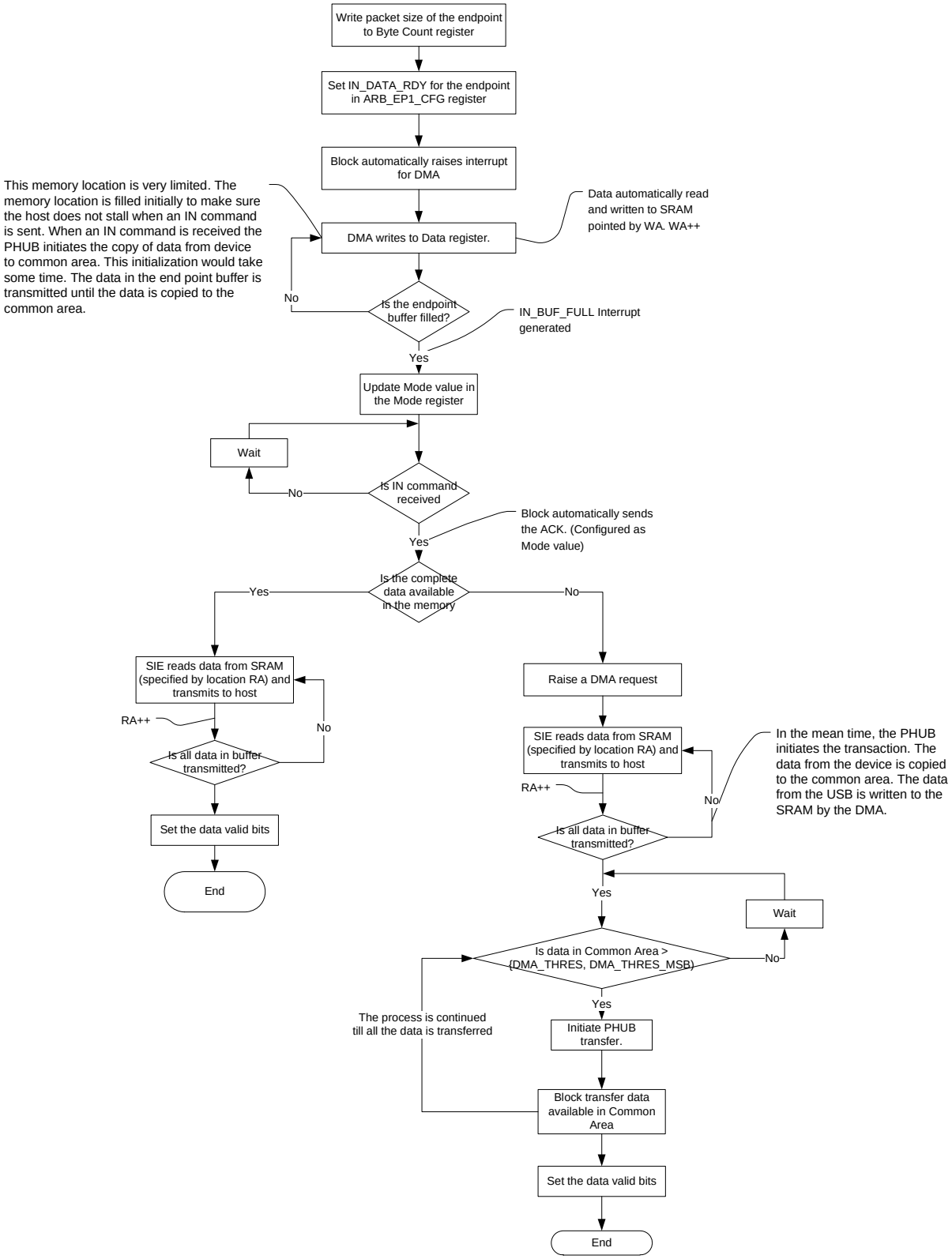
When an IN command is sent by the host, the device responds with the data present in the dedicated memory area for that endpoint. It simultaneously issues a DMA request for more data for that EP. This data fills up in the Common Area. The device does not wait for the entire packets of data to be available. It only waits for the (USB_DMA_THRES_MSB, USB_DMA_THRES) number of data available in the SRAM memory and begins the transfer from the common area. See DMA Interface on page 252 for details and constraints regarding DMA transfers to the USB block.

Similarly, when an OUT command is received, the data for the OUT endpoint is written to the common area. When some data (data greater than (USB_DMA_THRES_MSB, USB_DMA_THRES)) is available in the common area, the Arbiter block initiates a DMA request to the PHUB and the data is immediately written to the device. The device does not wait for the common area to be filled.

This mode requires the configuration of the DMA_THRES and DMA_THRES_MSB registers to hold the number of bytes that can be transferred in one DMA transfer. Similarly, the PHUB register must be configured for the BURSTCNT values. The BURSTCNT value must always be equal to the value set in the DMA_THRES registers. The block sends the Termin signal to the PHUB along with the last data byte of the packet. Apart from the DMA registers, this mode also needs the configuration of the BUF_SIZE for the IN and the OUT buffers and the EP_ACTIVE and the EP_TYPE registers.

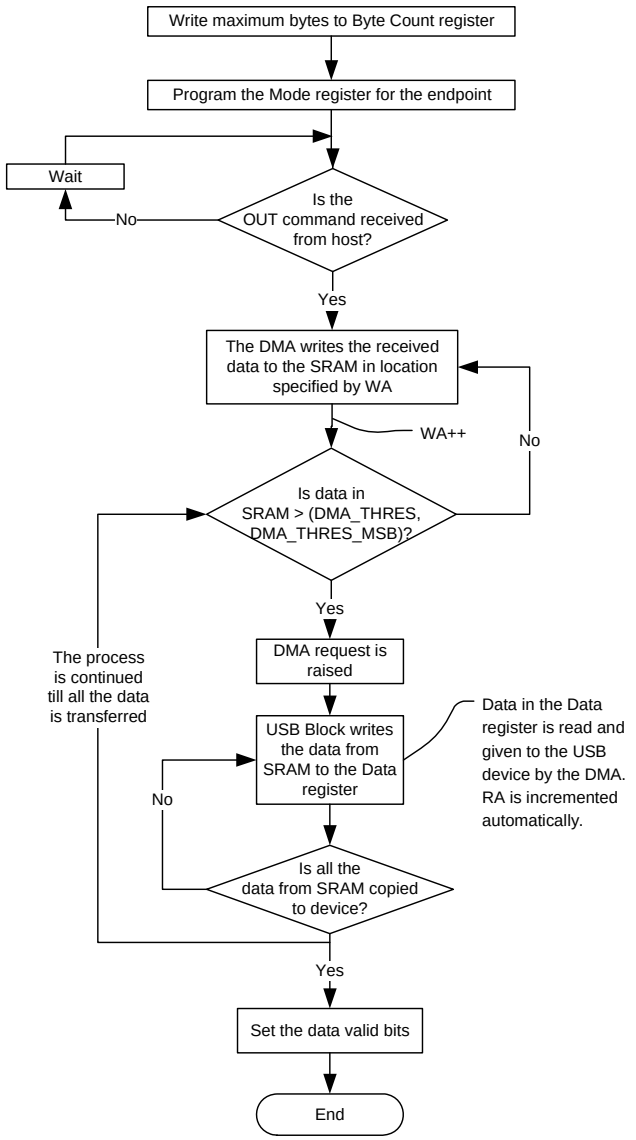
IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in Figure 24-6 on page 260.

Figure 24-6. Cut Through Mode IN Transaction



OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in Figure 24-7.

Figure 24-7. Cut Through Mode OUT Transaction



24.4.3 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 below:

Figure 24-8. IN Transaction

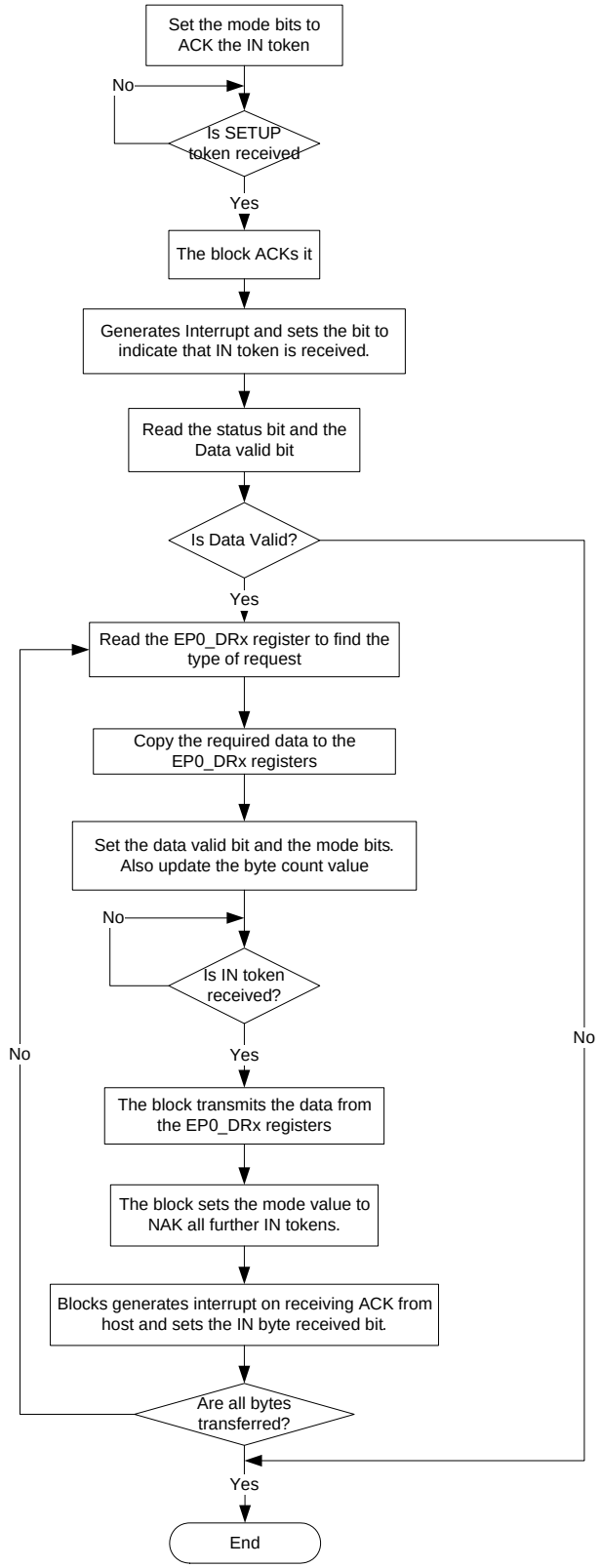
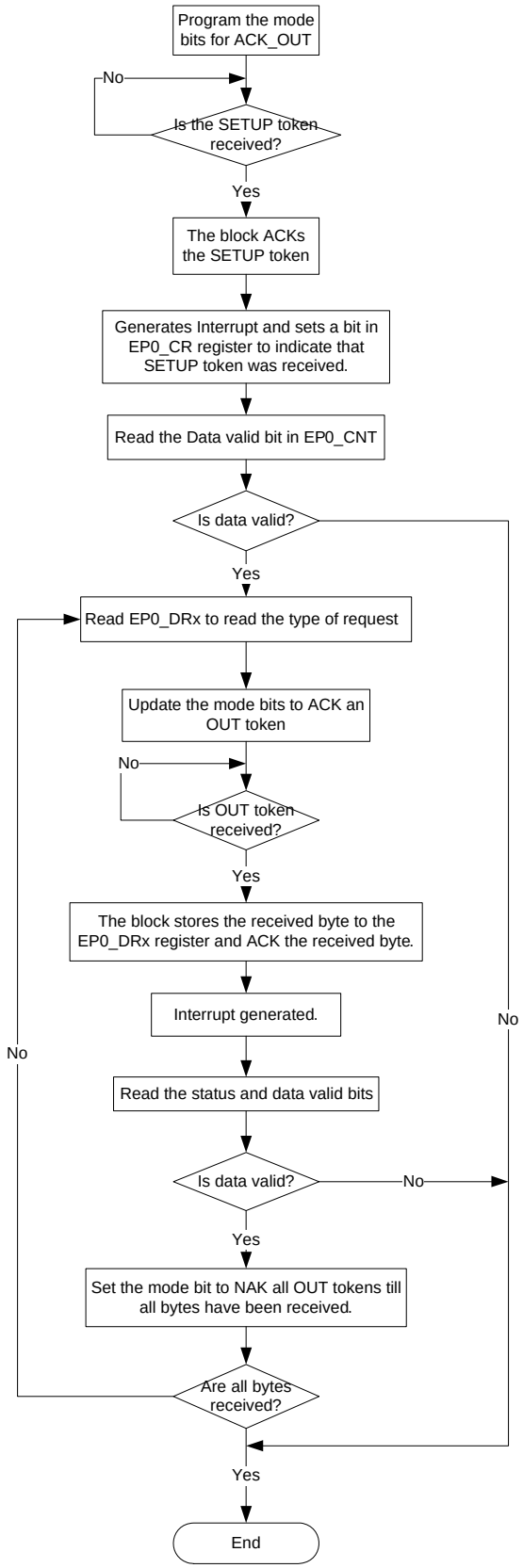


Figure 24-9. OUT Transaction



24.5 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 “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.

24.6 Register List

Table 24-4. 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_CNT1 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		
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 like 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

Table 24-4. USB Register List *(continued)*

Register Name	Comments	Features
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

25. 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).

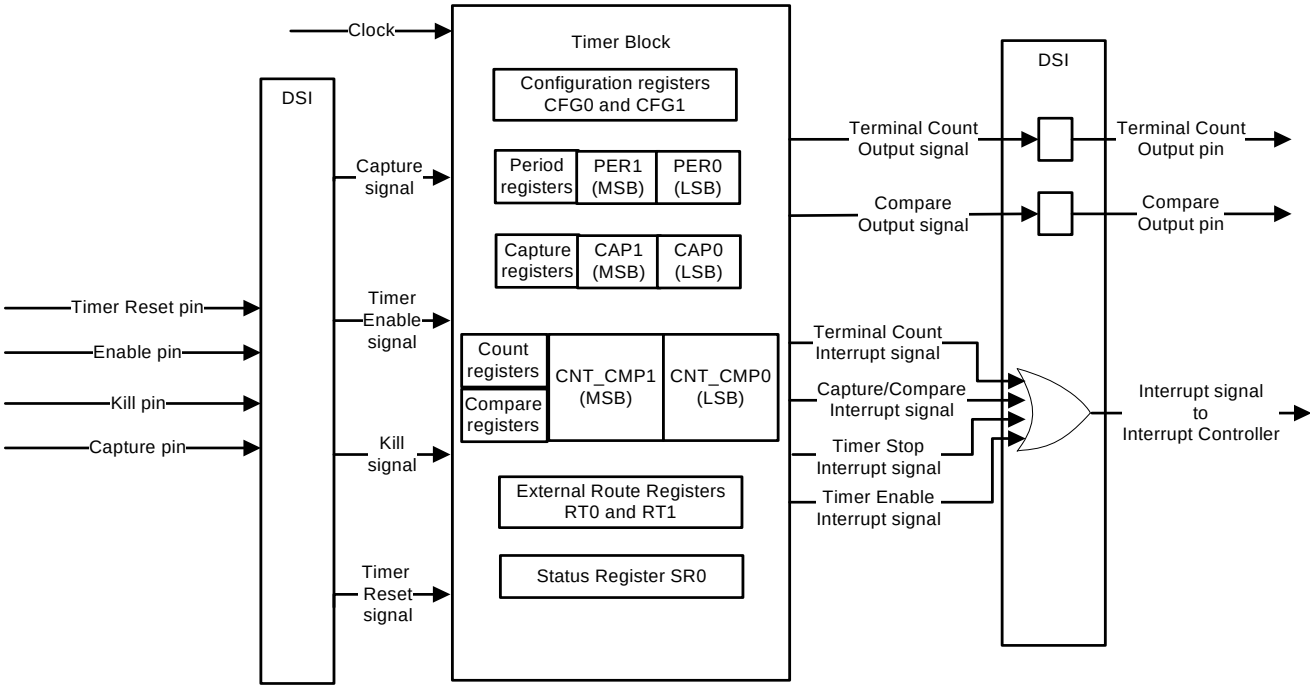
25.1 Features

- 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

25.2 Block Diagram

Figure 25-1 on page 266 shows one timer block.

Figure 25-1. Timer Block Diagram



25.3 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.

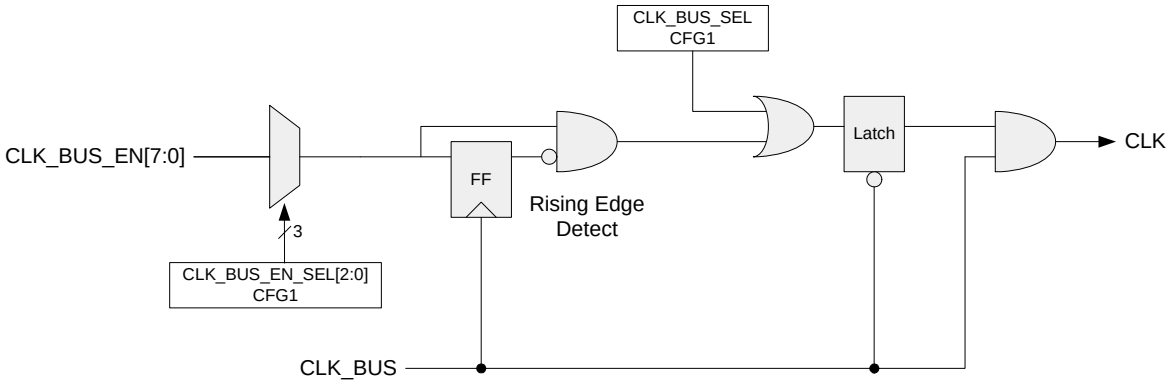
25.3.1 Clock Selection

The block supports the flexibility to select the required clock source. As shown in Figure 25-2 on page 267, 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 25-2. Clock Selection



25.3.2 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.

25.3.3 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 25-1 on page 266.

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.

25.3.3.1 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.

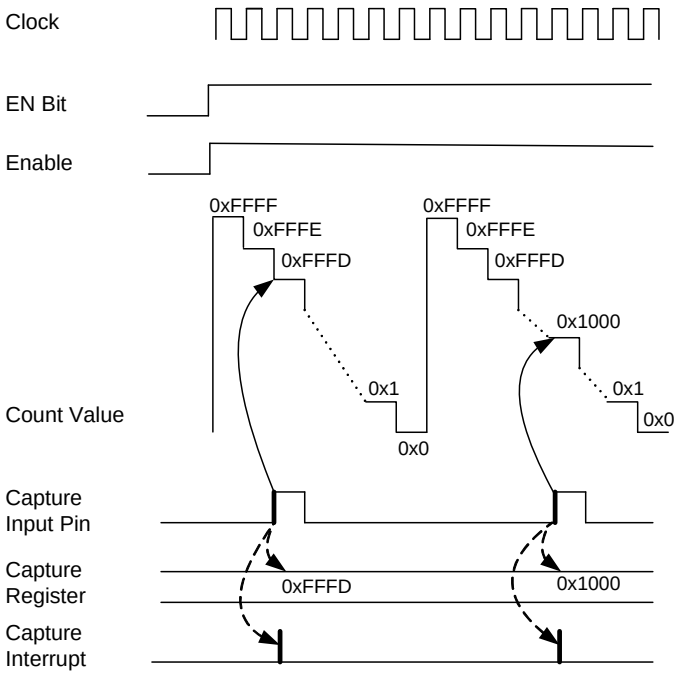
25.3.3.2 Capture Signal

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 25-3 shows the effect of the capture signal (period register = 0xFFFF).

Figure 25-3. Capture Mode Timing Diagram

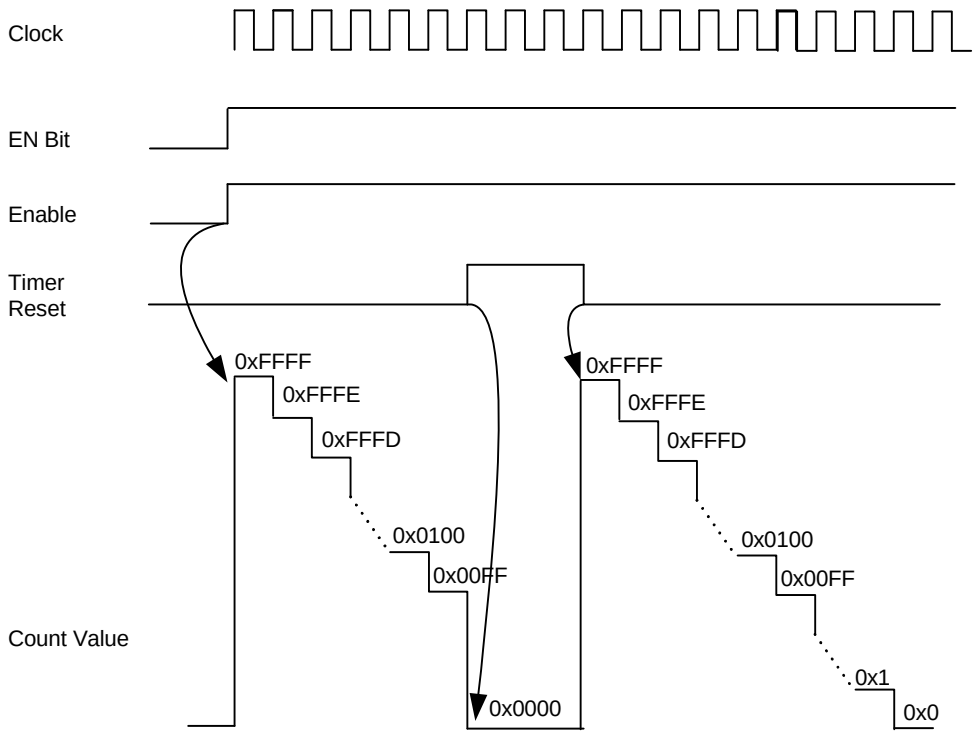


25.3.3.3 Timer Reset Signal

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 25-4 is a timing diagram for the timer reset signal (Period register = 0xFFFF).

Figure 25-4. Timer Reset Signal Timing Diagram



25.3.3.4 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.

25.3.4 Operating Modes

25.3.4.1 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
- TMRx_CFG2[1:0] = 0 – Timer runs in continuous mode

  Note These bits cannot be modified by the user.

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.

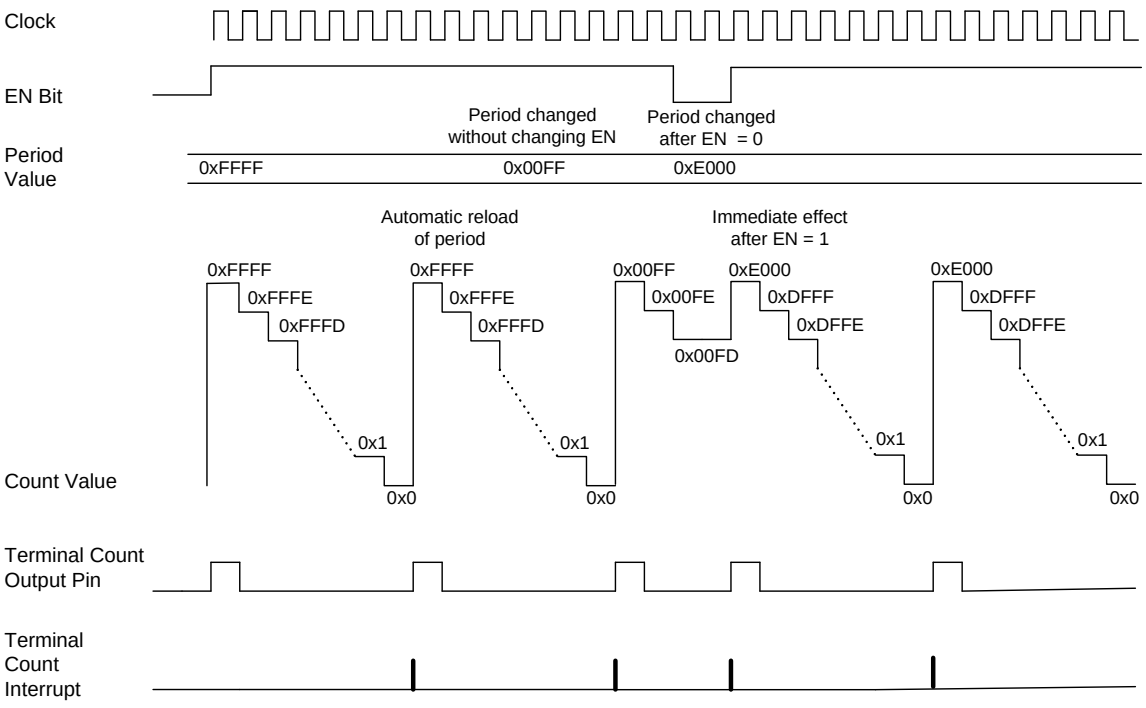
The following describes the process:

- 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 25-5 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 with 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 25-5. Free Run Mode Timing Diagram



25.3.4.2 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 of TMRx_CFG2[1:0] – gated timer runs in various modes

These modes are shown in Table 25-1.

Table 25-1. TMRx_CFG2[1:0] Bit Settings in Gated Timer Mode

TMRx_CFG2[1:0]	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 TIMEREN
10	Period count – counts from one positive edge to the next positive edge of TIMIEREN
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

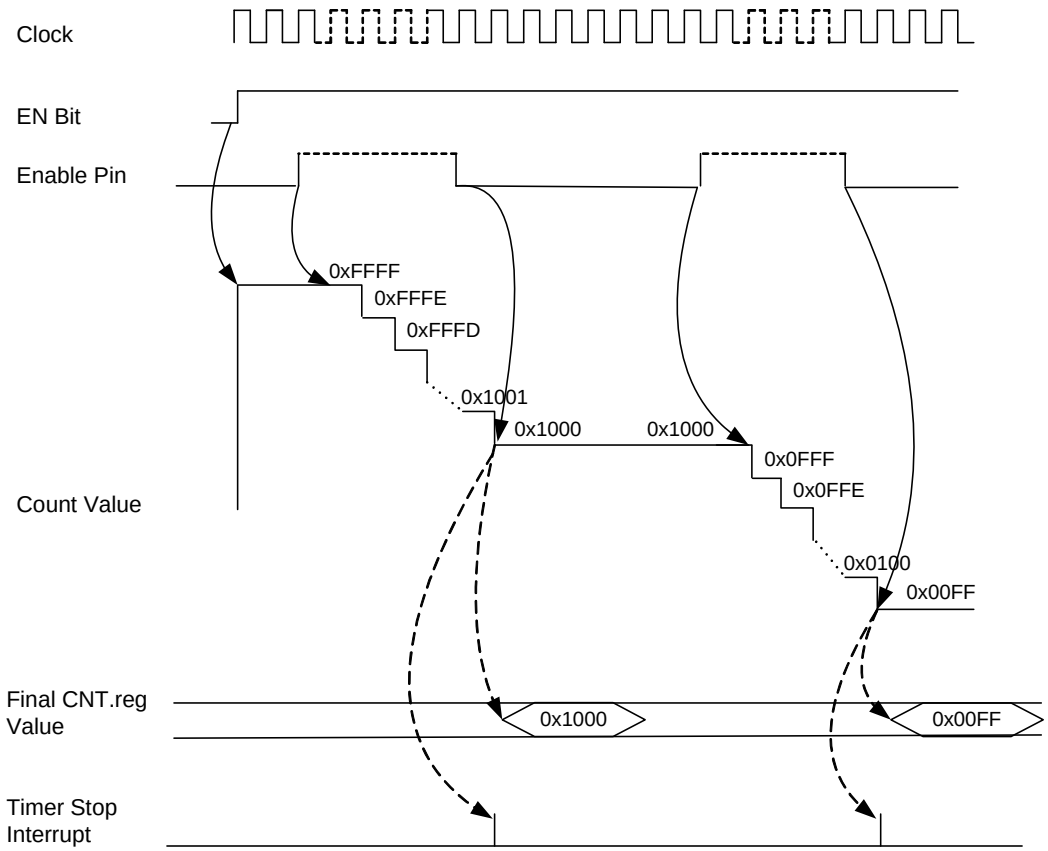
The input signal is given to the Enable pin. The timer begins counting at the rising edge of the Enable signal and stops counting at the falling edge of the Enable signal. There is a latency of one clock cycle for the block to detect the edges.

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 25-6 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 25-6. Gated Timer in Pulse Width Mode



Period Mode

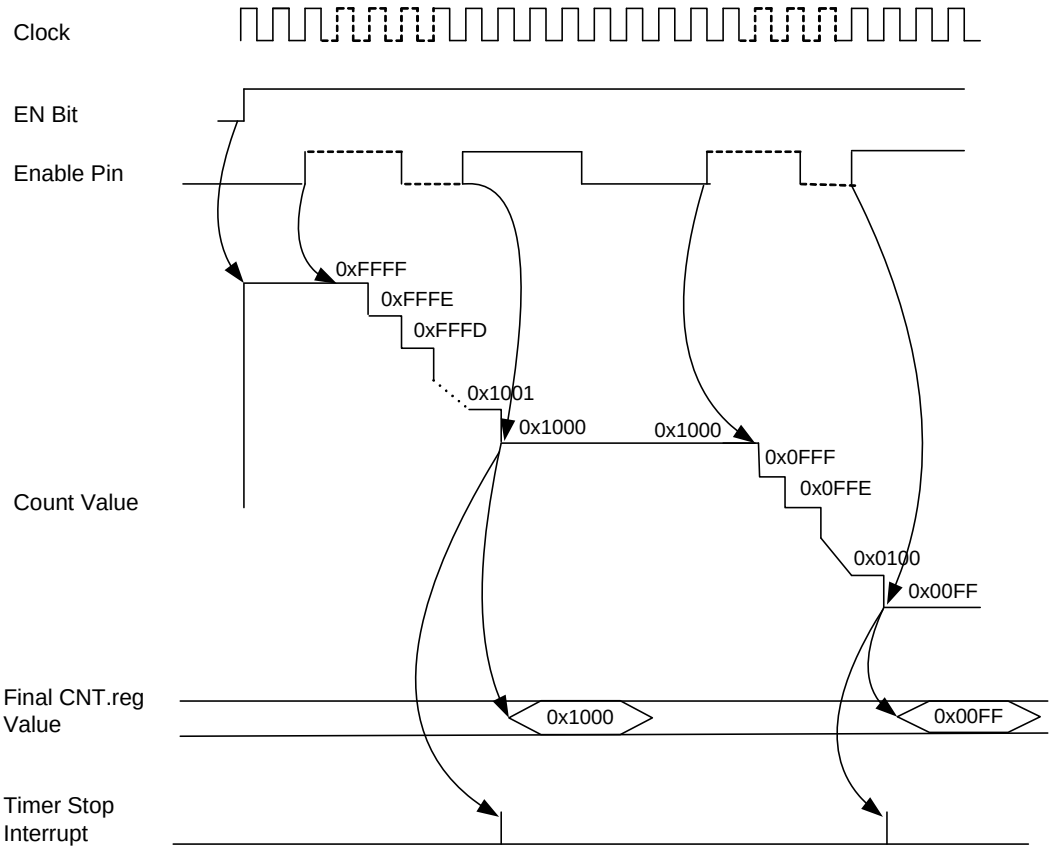
The input signal is given to the Enable pin. In this mode, the timer begins counting at the rising edge of the Enable signal and stops counting at the next rising edge. There is a latency of one clock cycle for the block to detect the edges.

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 25-7 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 25-7. Gated Timer in Period Mode



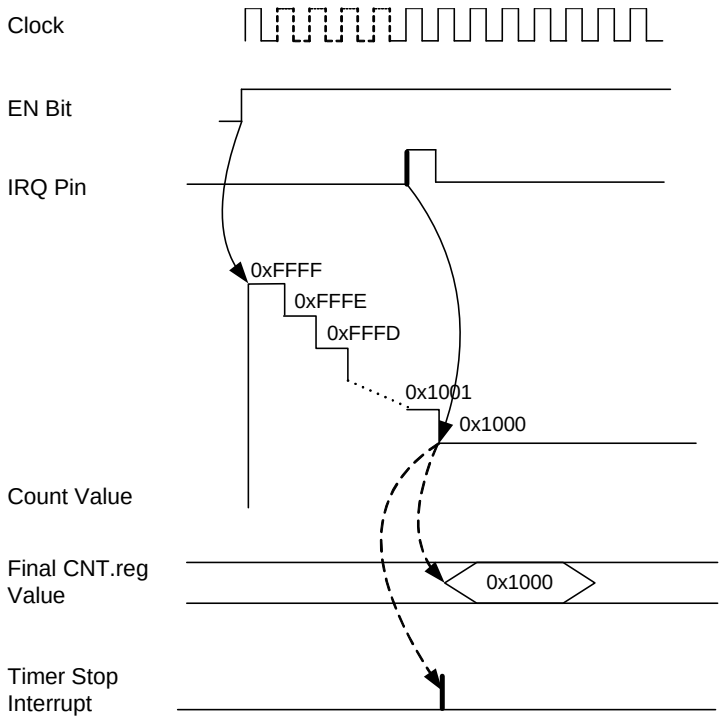
Stop on Interrupt Mode

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 25-8 shows the Gated Timer in IRQ mode.

Figure 25-8. Gated Timer in IRQ Mode



25.3.4.3 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, TMRx_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 25-2. Register Settings for Compare Type

CMP_CGF	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 278). 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.

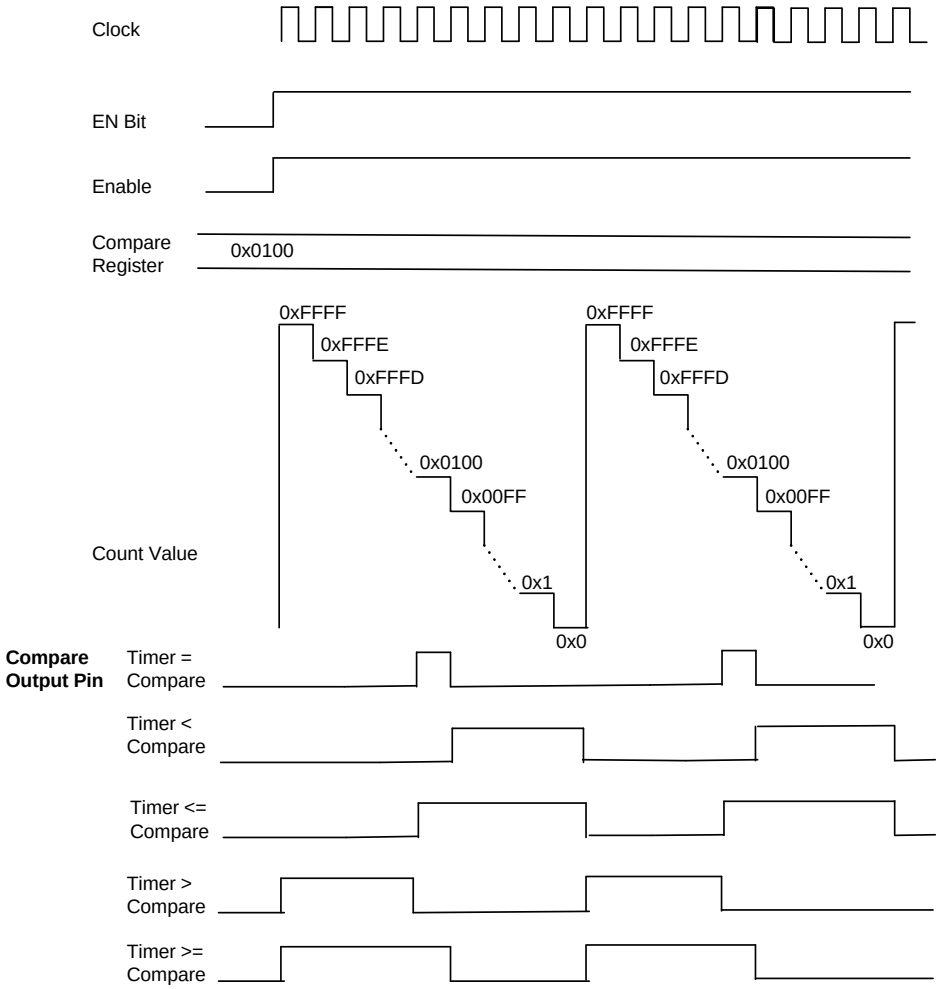
CMP_CFG = 100

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 25-9 shows the compare output for various Compare types. The Period register is loaded with 0xFFFF, and the Compare register is loaded with 0x1000.

Figure 25-9. 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.

25.3.4.4 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

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

The following table shows end criteria (where the block stops) 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 25-3. 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

25.3.5 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 25-1 on page 266. 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.

  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.

25.3.6 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.

25.4 Register Listing

The following table lists the registers.

Table 25-4. 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

26. I²C

PSoC® 3 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 on 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 see 26.3 Background Information.

26.1 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 or not other masters are on the bus. For multimaster mode lock synchronization is supported.

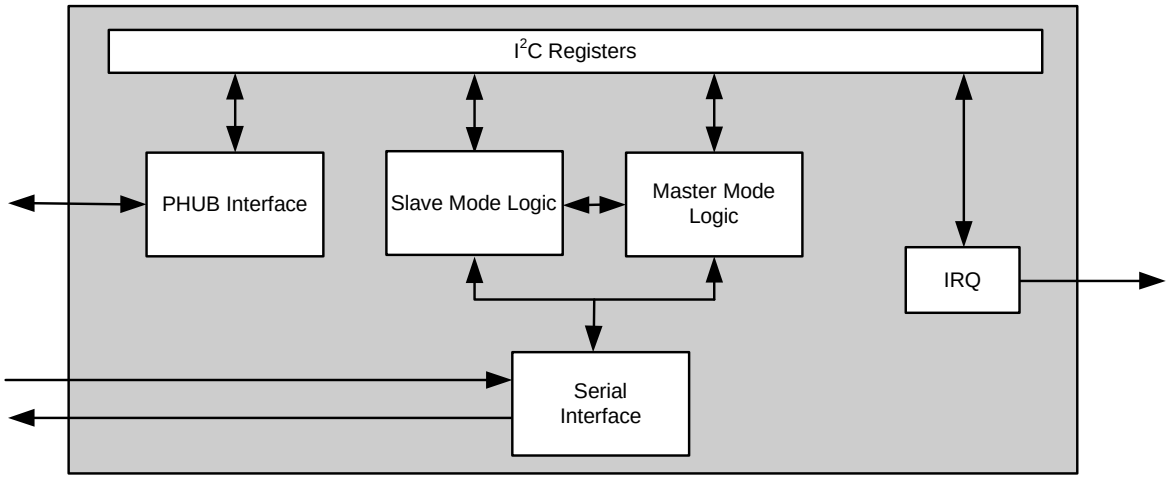
Basic I²C features include:

- 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 HW address compare, and wake from sleep on address match
- Provides 50 ns glitch filtering

26.2 Block Diagram

Figure 26-1 is a block diagram of the PSoC I²C interface.

Figure 26-1. Block Diagram of the PSoC I²C Interface



26.3 Background Information

The following information is provided to familiarize the user with the I²C bus and the way it transfers data.

26.3.1 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 26-2 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.

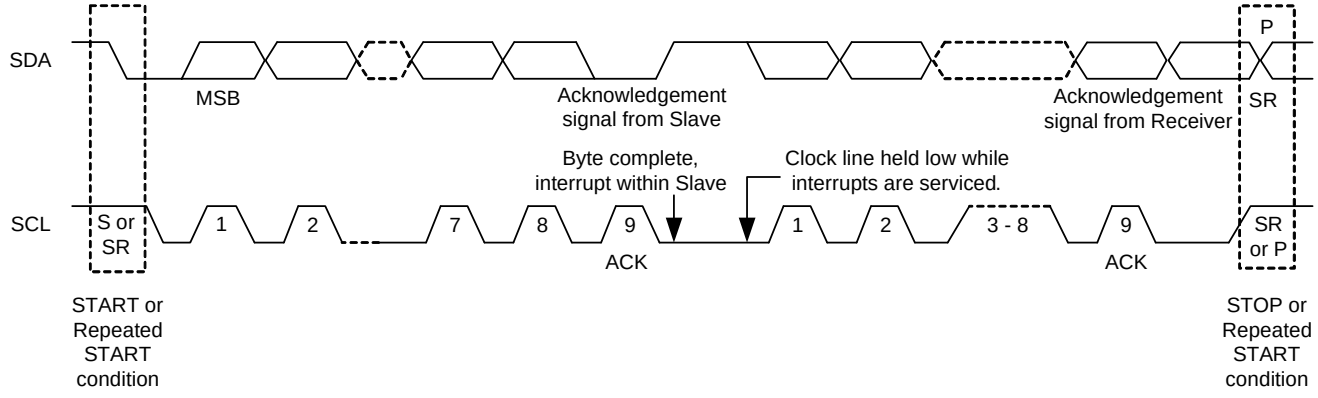
26.3.2 Typical I²C Data Transfer

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 26-2. I²C Transfer of a Single Data Byte, With Clock Stretching by a Non-PSoC Slave



26.4 How It Works

The PSoC 3 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 26.4.6 Operating the I²C Interface on page 284 and sections

26.7 Slave Mode Transfer Examples on page 289, 26.8 Master Mode Transfer Examples on page 291, and 26.9 Multi-Master Mode Transfer Examples on page 293.

The I²C interface supports either 7-bit or 10-bit addressing. The hardware supports 7-bit address compare. In slave mode, 7-bit address detection is done by using either a hardware address compare or 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.

26.4.1 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.

26.4.2 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.

26.4.3 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 19.3.2.5 Open Drain, Drives High and Drives Low on page 164).

The I²C must be routed to the SIO pins to use the block in sleep.

26.4.4 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.

26.4.5 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 26-1. 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.

26.4.6 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 26-2.
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 26-3. 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 26.4.6.1 Slave Mode on page 286, 26.4.6.2 Master Mode on page 287, or 26.4.6.3 Multi-Master Mode on page 288.

Table 26-2. 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 26-3. Configuration For I2C Baud Rate[a]

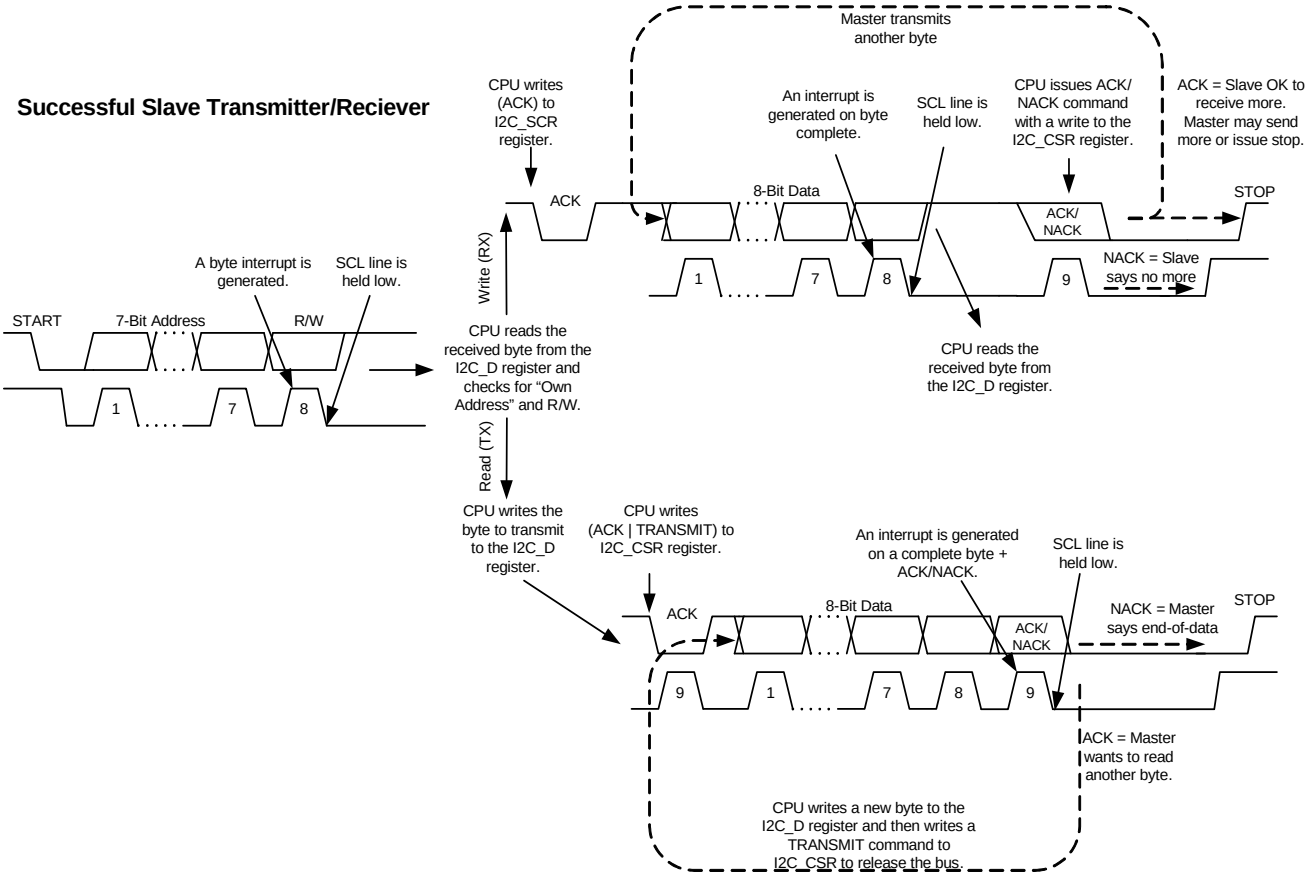
IMO Bus Clock (MHz)	I2C Mode	Oversample Rate	Divide Factor		SCL (kHz)
			I2C_CLK_DIV2[1:0]	I2C.CLK_DIV1[7:0]	
3	Standard	16	(0)2'b00	(2)8'b00000010	93.75
6	Standard	32	(0)2'b01	(2)8'b00000010	93.75
6	Fast	16	(0)2'b02	(1)8b'00000001	375
12	Standard	32	(0)2'b03	(4)8'b00000100	93.75
12	Fast	16	(0)2'b04	(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.

26.4.6.1 Slave Mode

To enable slave mode operation, set I2C_CFG bit 0, Enable Slave. See Figure 26-3.

Figure 26-3. 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 are 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 NACK 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 26-3. 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.
4. When transmitting bytes, the slave receives an ACK / NACK from the master for each byte sent.

   ACK is a signal that the master wants another byte.

   NACK or a Stop condition is a signal that the master does not want any more bytes – the CPU should let the I2C interface go to an idle state.
5. When receiving bytes, the slave ACKs / NACKs each byte received from the master.

   ACK is a signal that the slave can accept another byte.

   NACK is a signal that no more bytes can be accepted – after generating a NACK 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.

Slave Address Recognition

The slave address recognition feature can be enabled in hardware to reduce CPU usage. To enable hardware address recognition:

1. Set the 7-bit slave address in I2C_ADR, bits 0 to 6.
2. Set I2C_XCFG, bit 0, HW Addr En.

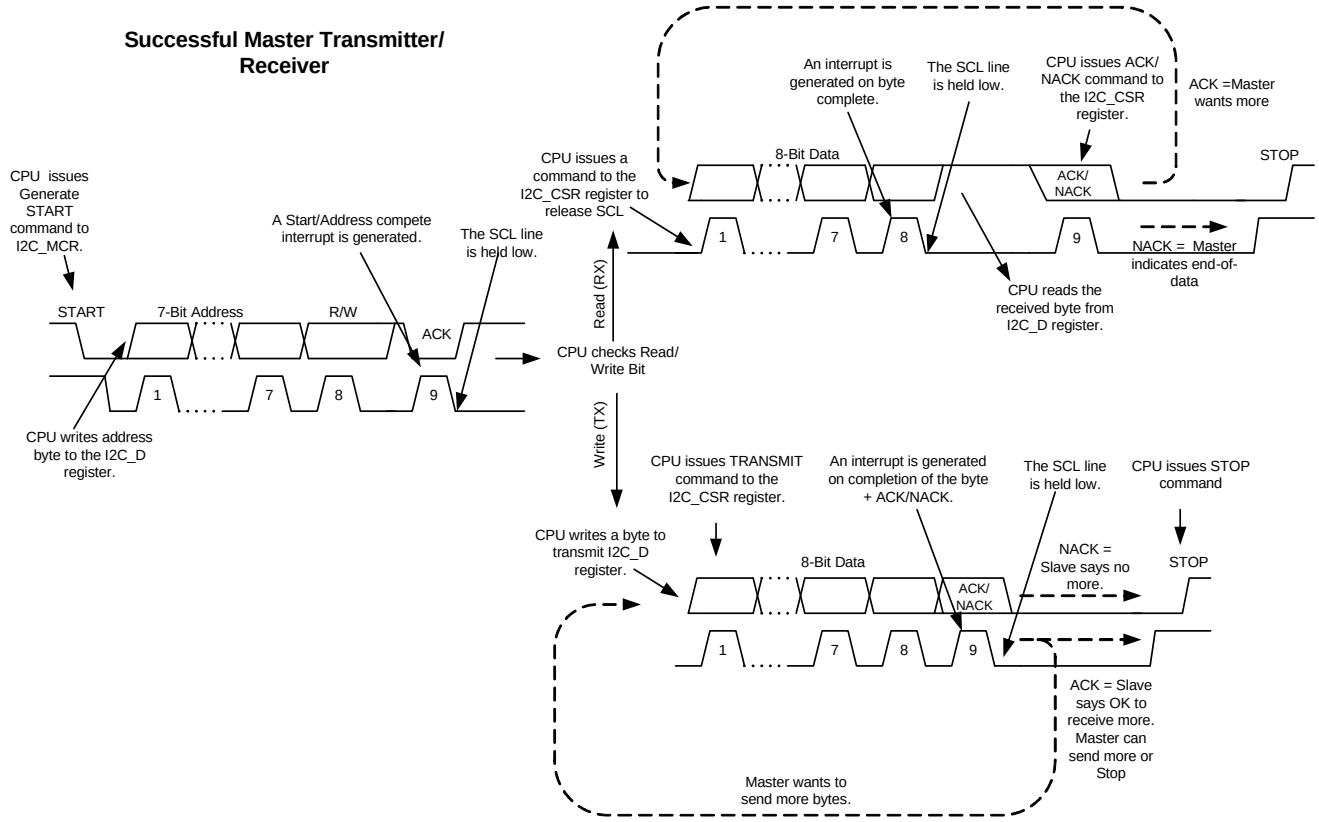
When hardware address recognition is enabled, the address portion of the first byte received after a bus Start condition is compared to the value in I2C_ADR.

- If no match is detected, the byte is automatically NAKed.
- If a match is detected, the byte is automatically ACKed, a byte complete interrupt is generated, and the remainder of the transfer is performed as described above.

26.4.6.2 Master Mode

To enable master mode operation, set the I2C_CFG bit 1, Enable Master. See Figure 26-4.

Figure 26-4. 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 26-4.

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.

26.4.6.3 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 is generated and the address byte is 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.

26.5 Hardware Address Compare

The hardware has the ability to compare the seven address bits received on the SDA line with that configured in the I²C.ADR register. On a true compare, the address is automatically ACKed, the SCL line held low, and a byte complete interrupt is issued. On reception of the byte complete interrupt from the hardware, the firmware needs to read bit [0] of the data register to determine Read/Write direction for the transfer. The firmware must then set the transmit bit in the I2C.CSR register to release the SCL. On an mismatch the address is automatically NAKed and the hardware revert to an idle state waiting for the new Start detection.

26.6 Wake from Sleep

When the HW address compare is enabled and the device is put to sleep, the slave can be used to wake the device on an I²C HW address match (only when either of the SIO pairs are used as I²C pins). While in sleep, the master clock is disabled. The incoming SCL clock is used to latch the address into the block. When the address matches, the wakeup interrupt is asserted to wake the system up, and the SCL is pulled low until the master clock is operational. After the system wakes, the I²C block is switched back to normal operation, and all other transaction proceed.

The I²C Block only responds to transactions during sleep if and only if:

- The I²C block is enabled in slave mode and slave mode only
- Hardware address compare is enabled
- There is an address value written in the I²C.ADR
- The I2C_ON bit in I2C.XCFG is set to 1'b1

Follow this procedure before putting the part to sleep, to ensure proper sleep mode I²C operation.

1. The CPU must set the Force NACK bit of the I2C.XCFG register when it wants to put the part into sleep.
2. The FW must poll the Ready_To_Sleep bit in the I2C.XCFG bit. One the bit is high, FW can put the part to sleep.

If I2C address match wake up is required in standby mode, then clear the EN_I2C bit in the PM_STBY_CFG5 register before entering the standby mode. Clearing this bit asserts the power down signal to the I2C block, which is required for an I2C address match wake up.

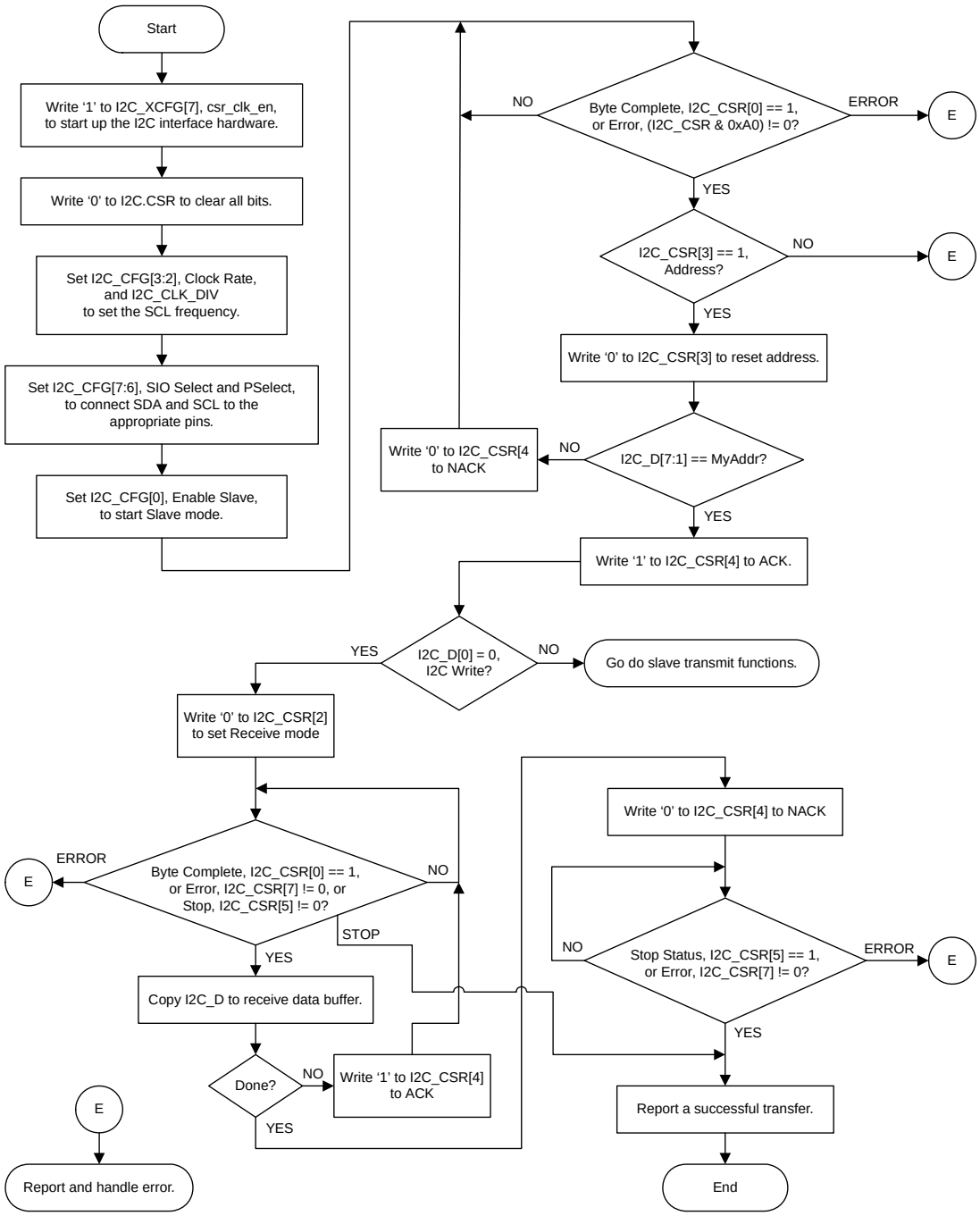
26.7 Slave Mode Transfer Examples

Slave mode receives or transmits data, as described in this section.

26.7.1 Slave Receive

A slave receive operation is accomplished as shown in Figure 26-5.

Figure 26-5. Slave Receive Operation Sequence

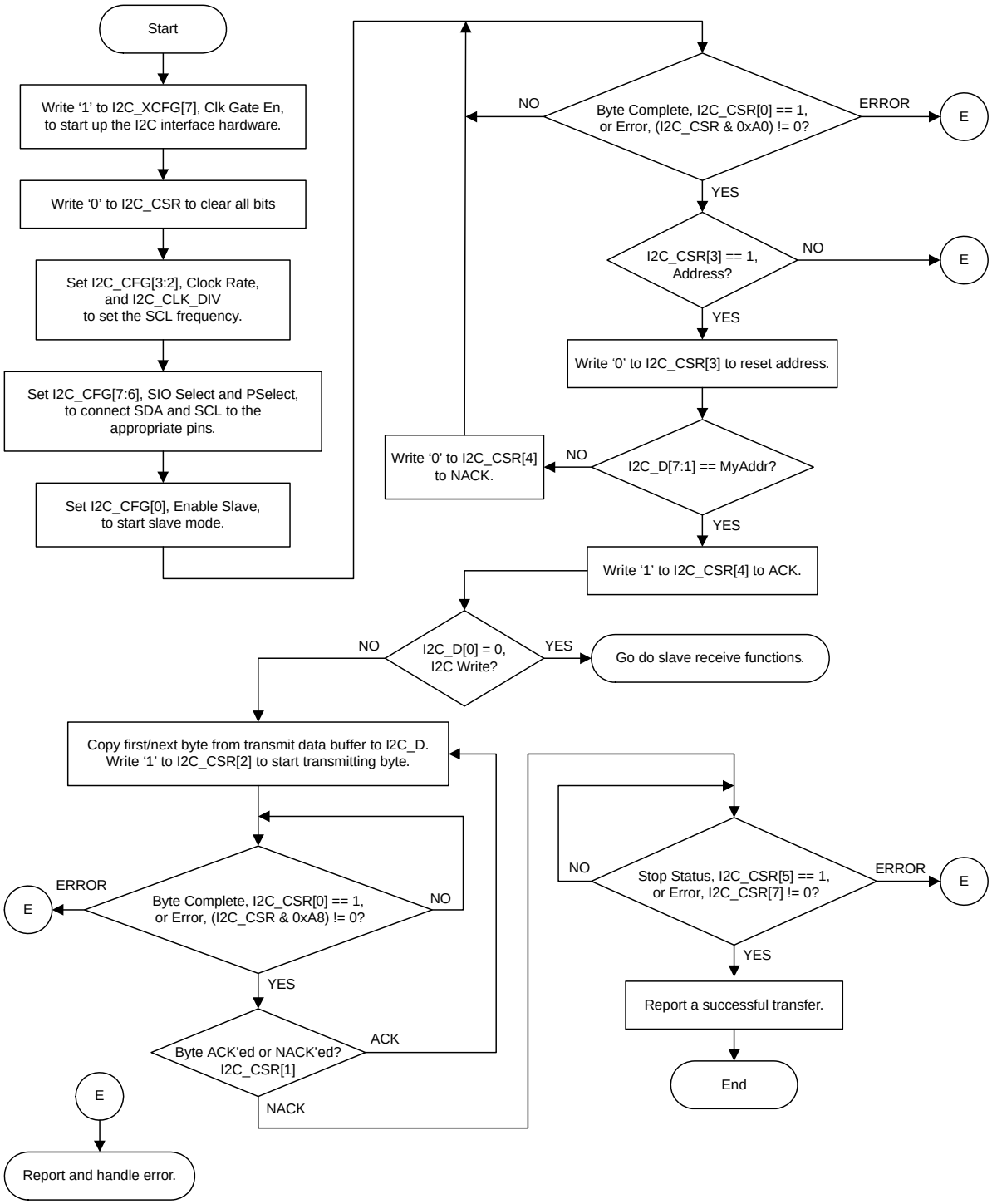


26.7.2 Slave Transmit

A slave transmit operation is accomplished is accomplished as shown in Figure 26-6.

Figure 26-6. 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.

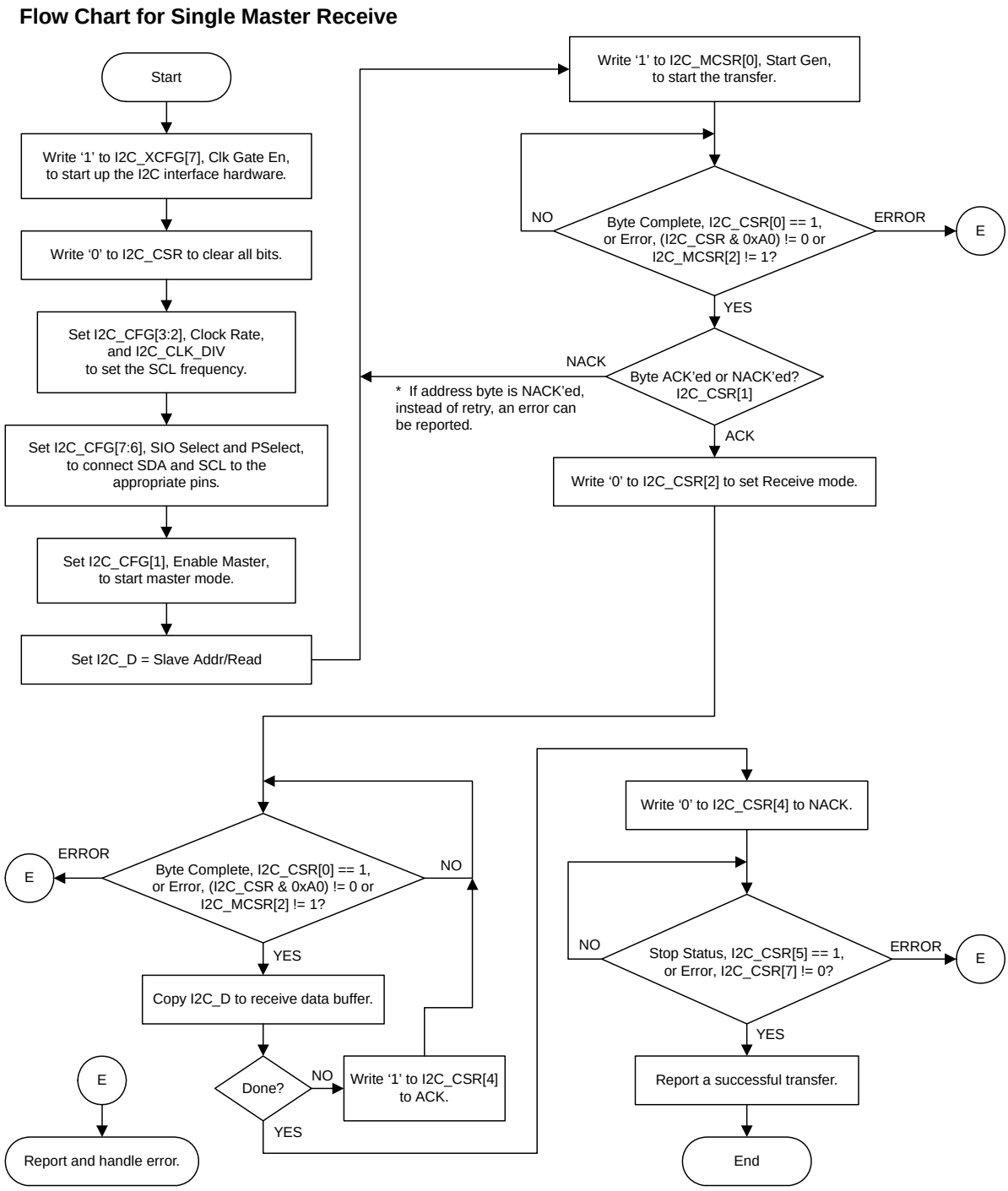
26.8 Master Mode Transfer Examples

Master mode receives or transmits data, as described in this section.

26.8.1 Single Master Receive

A master receive operation in a single-master system is accomplished as shown in Figure 26-7.

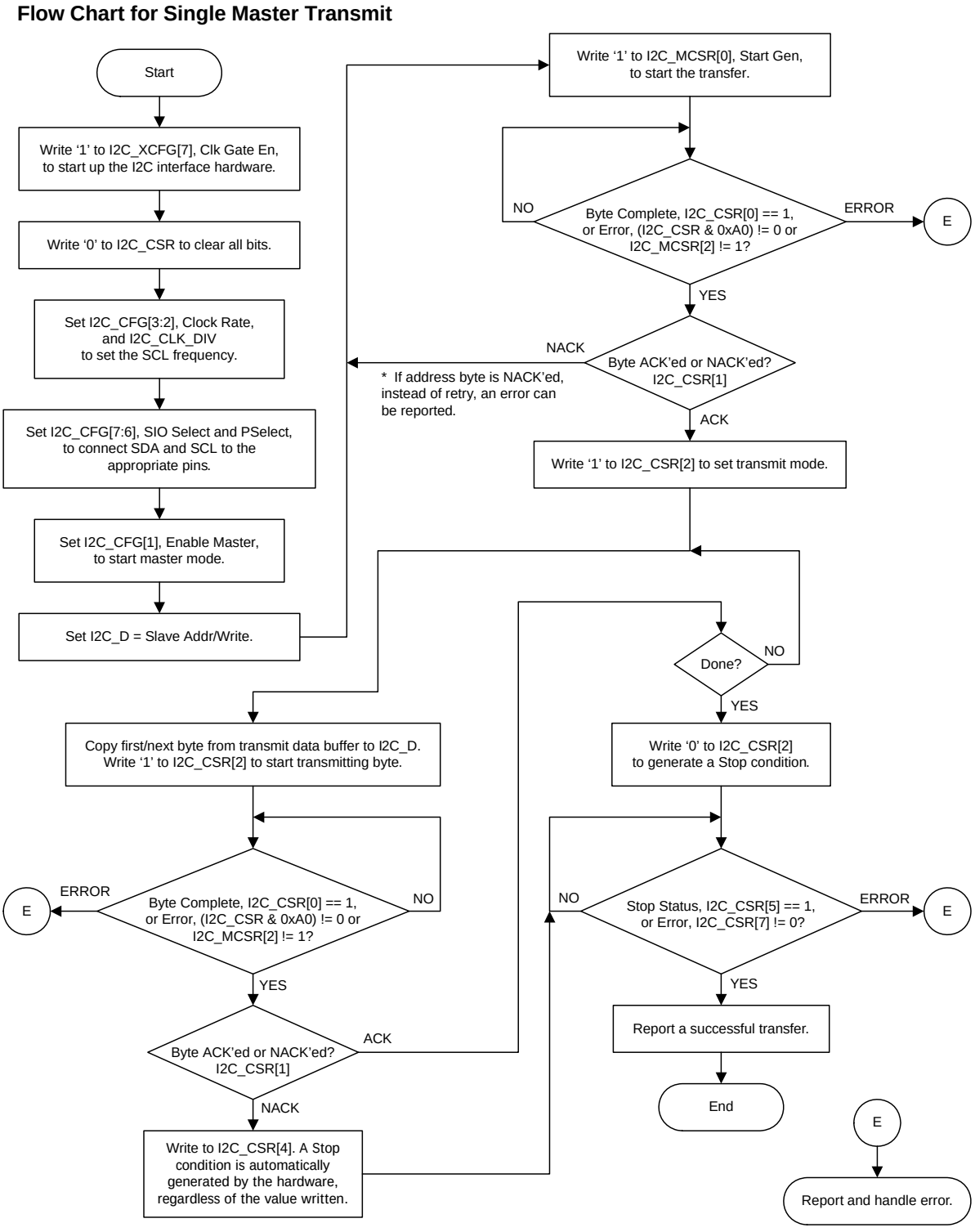
Figure 26-7. Single Master Mode Receive Operation



26.8.2 Single Master Transmit

Figure 26-8 illustrates the process by which you generate a master transmit operation in a single master system.

Figure 26-8. 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.

26.9 Multi-Master Mode Transfer Examples

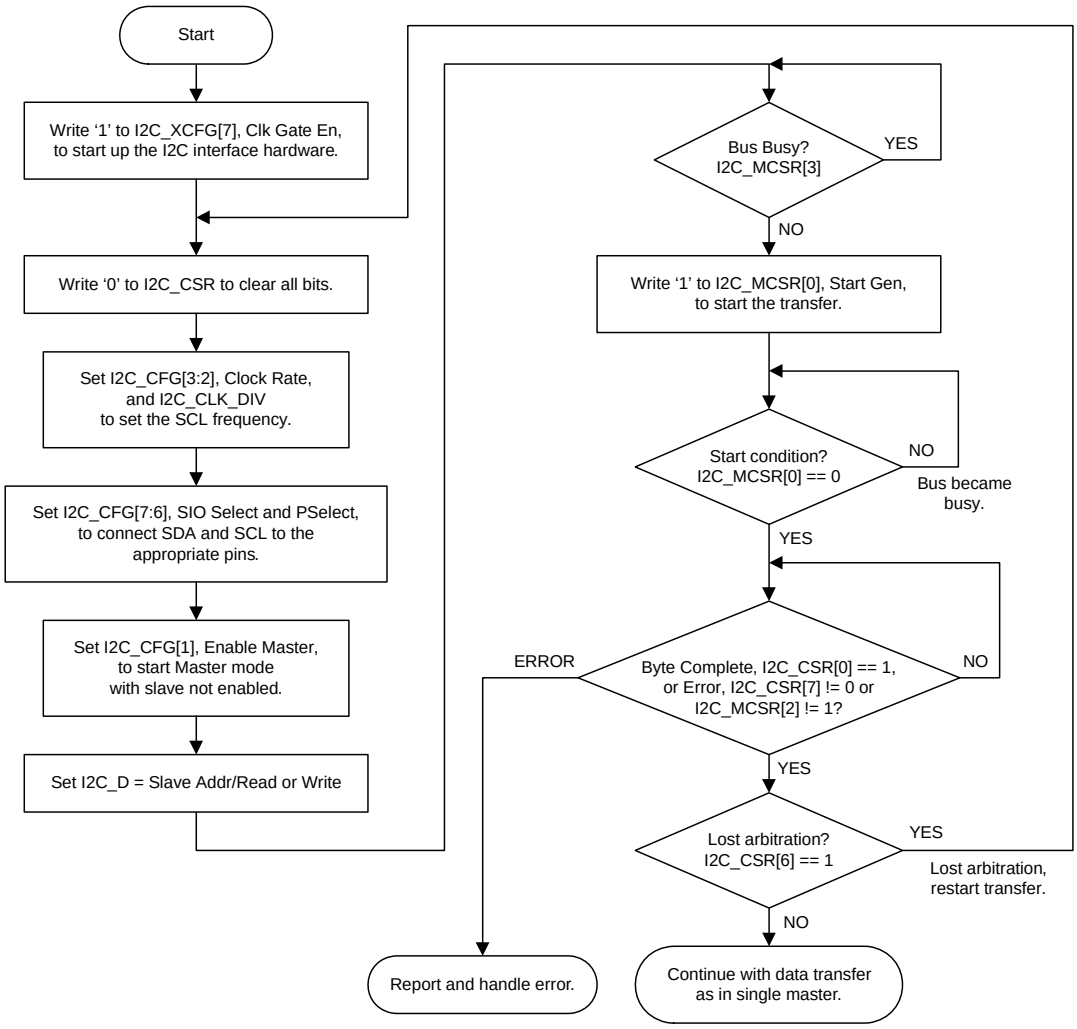
In multi-master mode, data transfer can be achieved with the slave mode not enabled or with the slave mode enabled.

26.9.1 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 26-9.

Figure 26-9. Multi-Master Mode, Slave Not Enabled Sequence

Flow Chart for Multi-Master, Slave Not Enabled

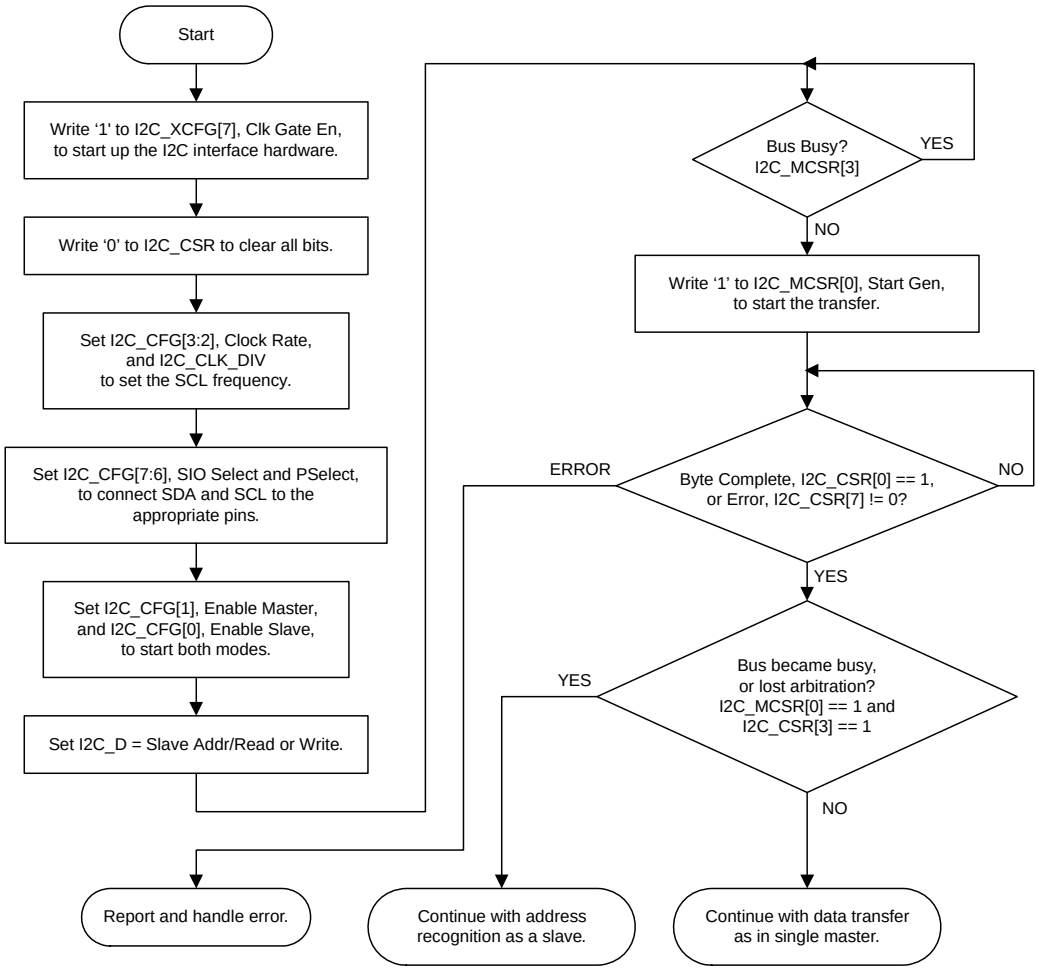


26.9.2 Multi-Master, Slave Enabled

A master data transfer operation in a multi-master system, where the slave mode is enabled is accomplished as shown in Figure 26-10.

Figure 26-10. Multi-Master Mode, Slave Enabled Sequence

Flow Chart for Multi-Master, Slave Enabled



27. 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.

27.1 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

27.2 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 27-1.

- 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 27-1) 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 27-1. Digital Filter Block Diagram

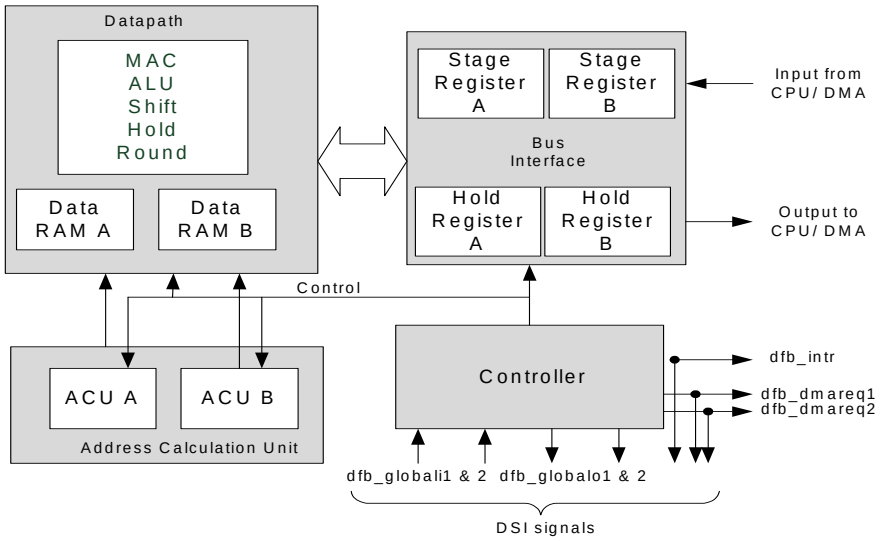


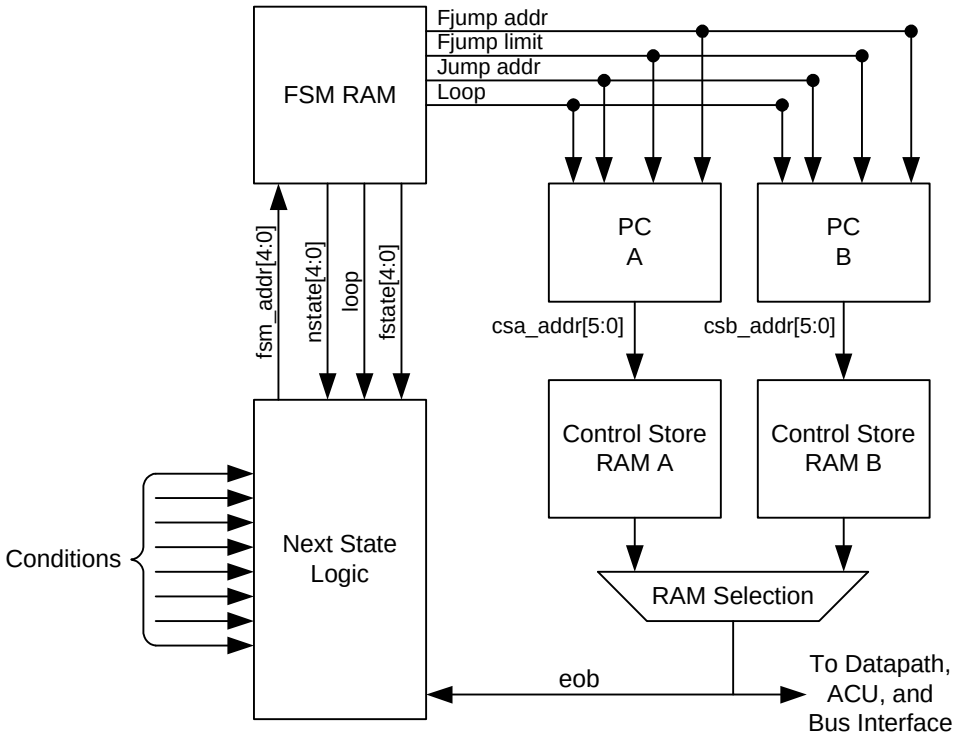
Figure 30-1. Digital Filter Block Diagram

27.3 How It Works

27.3.1 Controller

The controller consists of a RAM-based state machine, a RAM-based control store, program counters, and next state control logic (see Figure 27-2 on page 297). 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 27-2. 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 and the RAM produces next state information as well as 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.

27.3.1.1 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 27-1 shows the bit fields used for the controller by the 32-bit FSM RAM.

Table 27-1. 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.					

27.3.1.2 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

27.3.1.3 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 27-2.

Table 27-2. 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

27.3.1.4 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. See 27.3.2 Datapath on page 299 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 (see the PHUB and DMAC chapter on page 73).

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 27-3.

Table 27-3. 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

27.3.2 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.

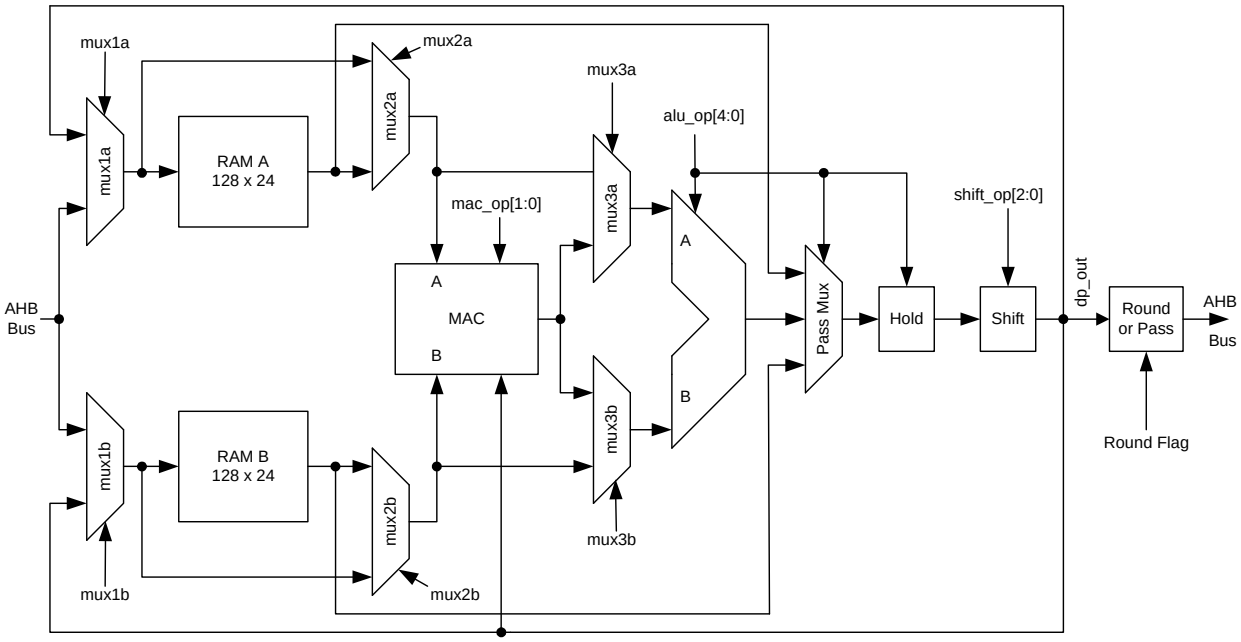
With the exception of the DP RAM addresses, the DP unit is completely controlled by seven control fields totaling 18 bits coming from the DFB Controller as the DP_CTRL control bus (Table 27-2 on page 298). These 18 bits of control are listed in Table 27-4.

Table 27-4. 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 27-4 affect the functioning of the different elements in the datapath.

Figure 27-3. 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].

27.3.2.1 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 27-7 on page 307, Table 27-6 on page 307, and Table 27-8 on page 308.

27.3.2.2 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 27-6 for various instructions supported by ALU.

The ALU commands as well as 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.

27.3.2.3 Shifter and Rounder

The shifter at the ALU output can be used to shift the ALU results as required. See Table 27-8 for various shifter commands. Rounder rounds the results to a 16 bit value when the data path is operating in round mode.

27.3.3 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.

27.3.4 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.

27.3.4.1 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.

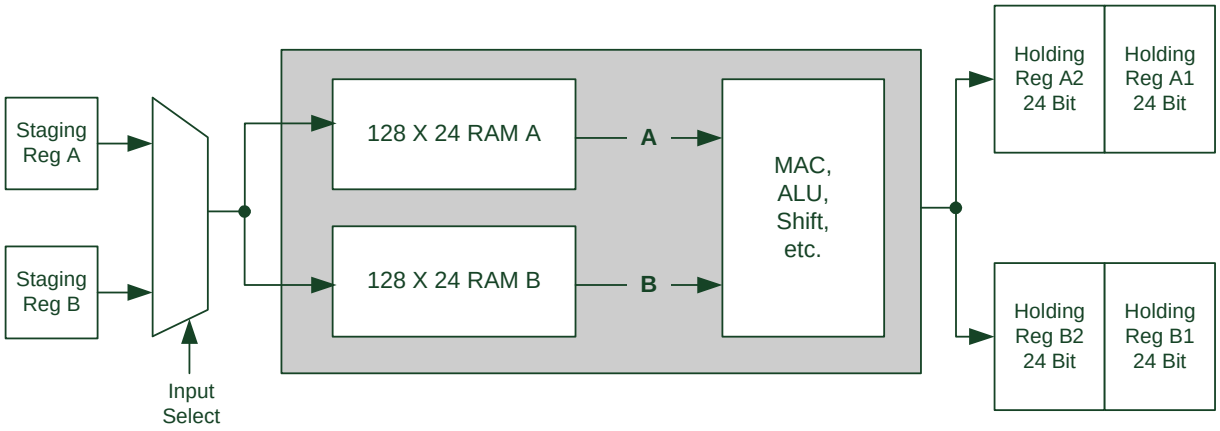
The DFB has:

- Two 24-bit input staging registers
- Two 24-bit output holding registers

These registers can be accessed by both DFB as well as 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 27-4. 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.

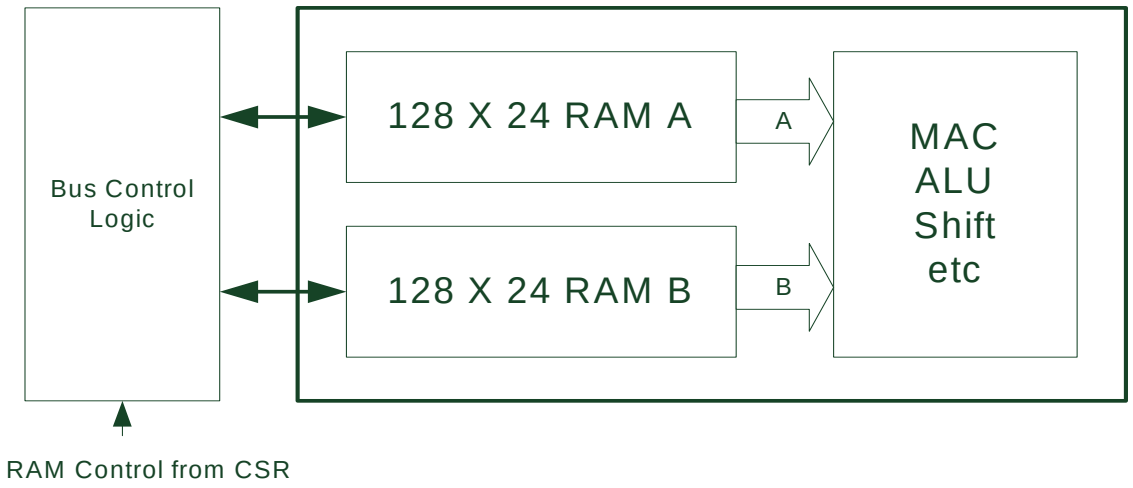
Apart from this, the Staging register also has a key coherence byte setting. This setting is available to reduce errors due to bus access being less wide as compared to the register width. The staging registers are protected on writes, so the underlying hardware does not incorrectly use the field when 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 desired. In the Staging register the new value availability is flagged only when the key coherency byte is written to.

27.3.4.2 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 27-5 on page 303.

Figure 27-5. 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.

27.3.4.3 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.

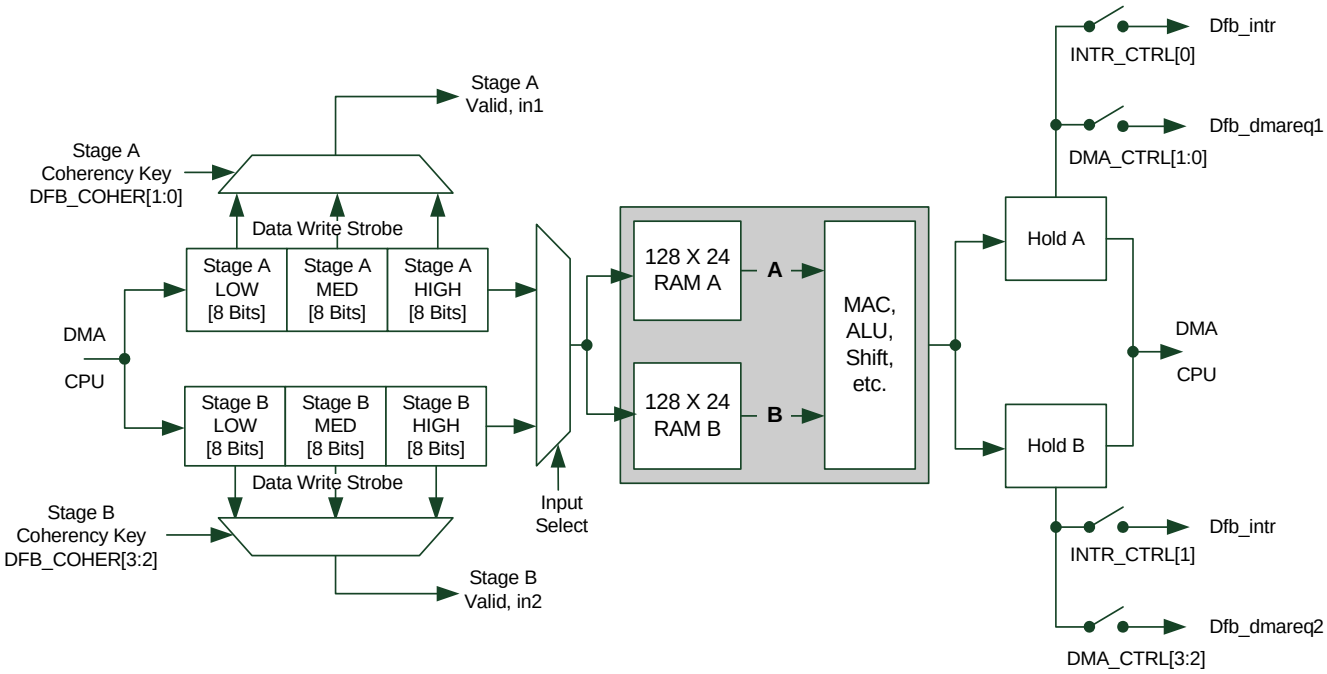
Operations on the Holding registers are protected. The nature of the protection is set by the coherence bits (DFB_COHER). The Holding registers are protected on reads so that the underlying HW does not update it when partially read by the System SW or DMA. The key coherency byte is selected in the Coherency 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 when 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 27-6 explains DFB control signals can be used for data streaming and result handling.

Figure 27-6. Control Signals for Data Streaming and result handling



27.3.4.4 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.

27.3.4.5 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.

27.3.4.6 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 27-5 on page 306. 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 27.3.1.4 Next State Decoder on page 298 for more details.

27.4 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 27-5.

Table 27-5. 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 27-6 and are encoded in 5 bits.

Table 27-6. 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 27-7 and are encoded in 2 bits.

Table 27-7. 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 27-8 and are encoded in 3 bits. If deeper shifts are required, data can be passed through the ALU on multiple cycles.

Table 27-8. 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 27-9 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 27-9. 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 27-9. 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

27.5 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 on 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

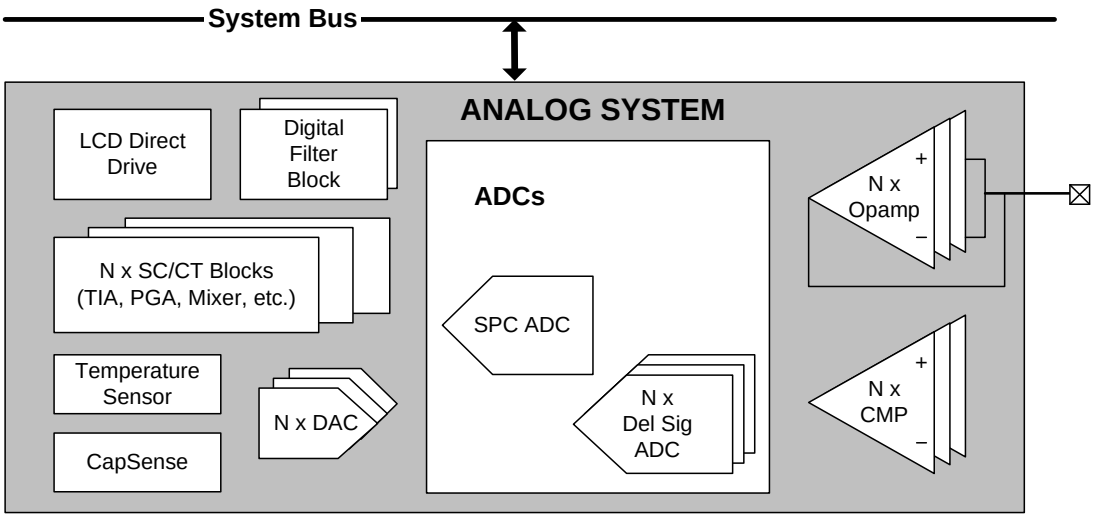
The PSoC® analog subsystem provides the device the second half of its unique configurability. All analog performance is based on a highly accurate absolute voltage reference with less than 0.2% error over temperature and voltage. The configurable analog subsystem includes analog muxes, comparators, mixers, 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 313
- Analog Routing chapter on page 327
- Comparators chapter on page 343
- Opamp chapter on page 347
- LCD Direct Drive chapter on page 351
- CapSense chapter on page 365
- Temperature Sensor chapter on page 371
- Digital-to-Analog Converter chapter on page 375
- Precision Reference chapter on page 379
- Delta Sigma Converter chapter on page 383

Top Level Architecture

Analog System Block Diagram



28. Switched Capacitor/Continuous Time

The PSoC® 3 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.

28.1 Features

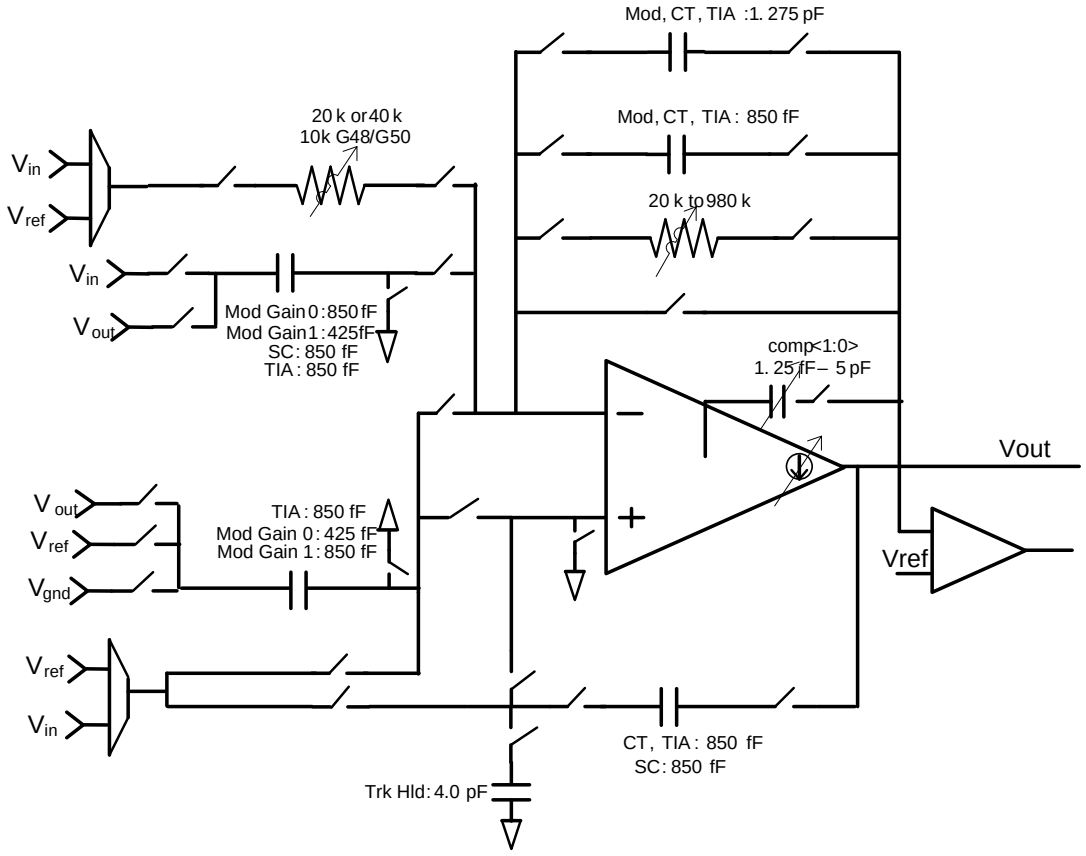
The PSoC SC/CT block has these features:

- 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

28.2 Block Diagram

The overall block diagram of the block is shown in Figure 28-1 on page 314. Individual block diagrams for the possible implementations are shown in separate sections.

Figure 28-1. Switched Capacitor and Continuous Time Block Diagram



28.3 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.

28.3.1 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].

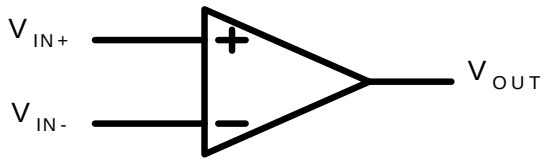
Table 28-1. SC/CT Block Operational Mode Settings

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

28.4 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 28-2. 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.

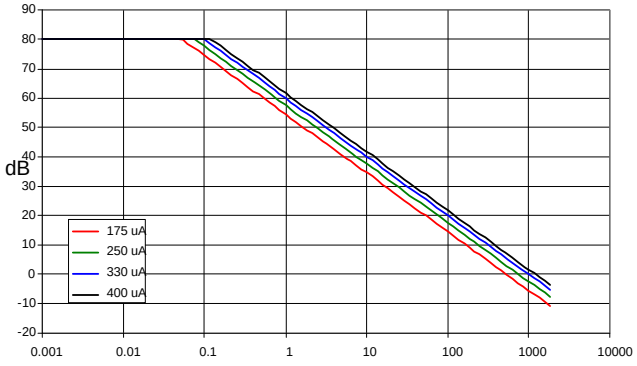
$$I_{load} = C_{load}\frac{\Delta V}{\Delta t} \qquad \textbf{Equation 1}$$

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 28-2. 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 28-3. Naked Opamp Drive Control I_LOAD



28.4.1 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]).

28.4.1.1 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.

28.4.1.2 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 28-3. 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

28.4.1.3 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 28-4. 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 28-5 on page 316. These are the settings 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 is modeled with two 150–Ω switches with an 8-pF load in between, then followed b ya 300–Ω impedance and a 50-pF external load.

The modulator mode is simulated with a 0.5-pF load at the output.

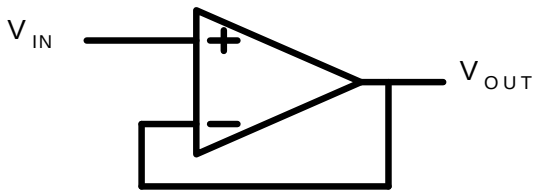
Table 28-5. Recommended Stability Settings by Mode

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 28-7 on page 318		
[111]	Track and Hold Amplifier	1	2	0

28.5 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 28-4. Unity Gain Buffer Configuration

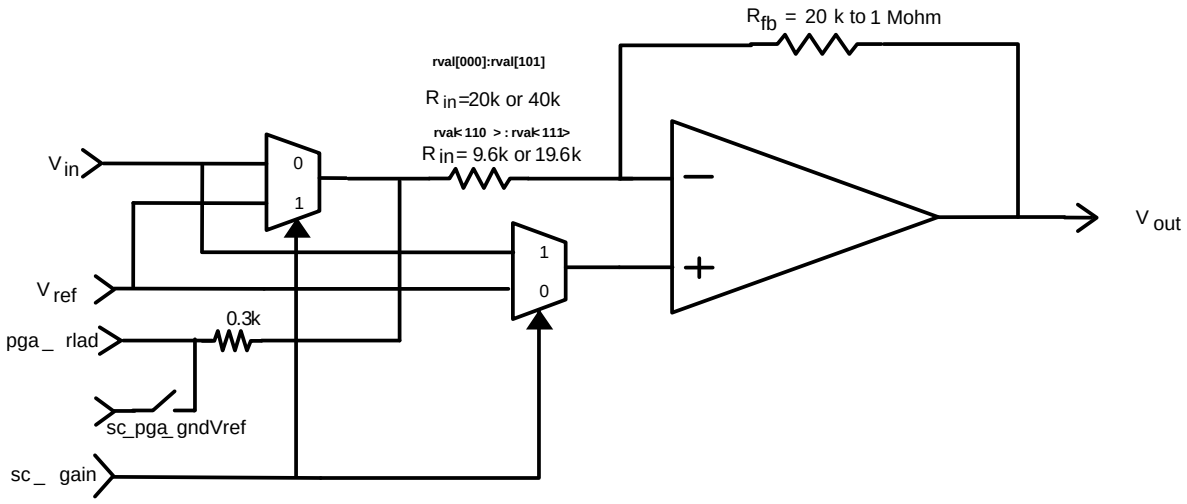


The unity gain buffer is used when an internally generated signal with high output impedance, such as a voltage DAC output, is required to drive a load; or when an external source with a high impedance is required to drive a significant on-chip load, such as the Continuous Time Mixer.

28.6 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 28-5. 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 28-6.

Table 28-6. PGA Gain Configuration

SC_GAIN	Gain
0	Inverting ($-R_{FB}/R_{IN}$)
1	Non-inverting ($1+ R_{FB}/R_{IN}$)

Figure 28-6. PGA Positive Gain (Noninverting) Topology

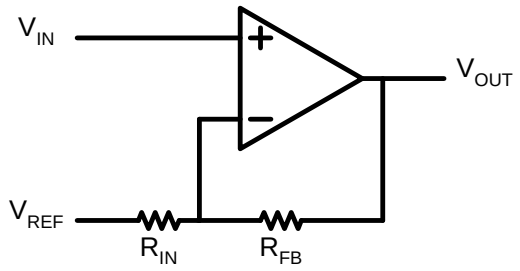


Figure 28-7. PGA Negative Gain (Inverting) Topology

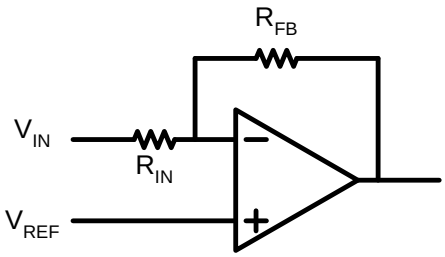
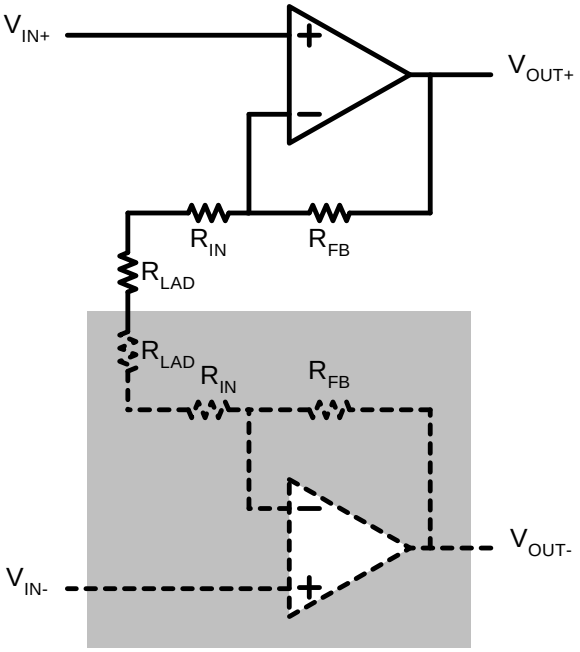


Figure 28-8. PGA Differential Amplifier Topology



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 28-7. PGA Stability Settings by Gain

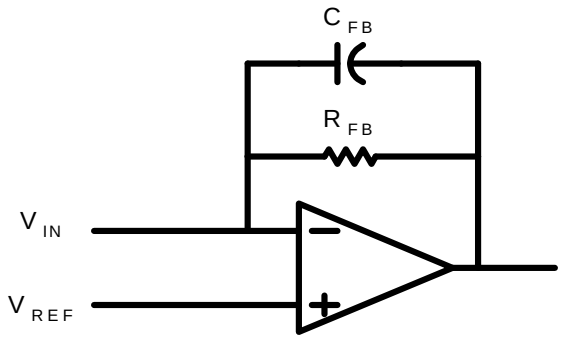
SC_RVAL [2:0]	R20_40B	Non-Inverting Gain (AC)	BIAS_ CONTROL	SC_COMP [1:0]	SC_REDC [1:0]
Bin	Bin	Lin			
0	0	1	1	2	0
0	1	1	1	2	0
1	0	2	1	2	1
1	1	2	1	2	1
10	0	4	1	0	1
10	1	4	1	0	1
11	0	8	1	0	1
11	1	8	1	0	1
100	0	16	1	1	1
100	1	16	1	1	1
101	0	24	1	1	3
101	1	32	1	1	3
110	0	24	1	0	2
110	1	48	1	1	0
111	0	25	1	0	2
111	1	50	1	1	0

The negative gain (inverting) topology is shown in Figure 28-7.

28.7 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 28-9. 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 28-8. 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 28-9. The feedback capacitance is set in bits [3:2] of the SCL[0..3]_CR2 register.

Table 28-9. 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.

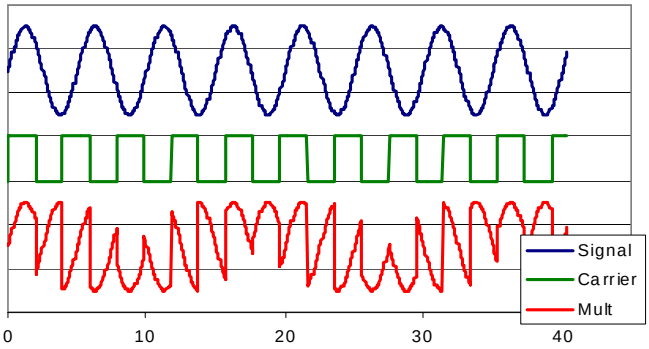
28.8 Continuous Time Mixer

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 is 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 28-10.

Figure 28-10. Continuous Time Mixer Waveforms



The output spectrum of the mixer includes terms at 455 kHz, 55 kHz, at $3 \times f_{CARRIER} \pm f_{SIGNAL}$, $5 \times f_{CARRIER} \pm f_{SIGNAL}$, $7 \times f_{CARRIER} \pm f_{SIGNAL}$, and so on. The up conversion is ultimately achieved by filtering out the desired 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. Figure 28-11 shows the continuous time mixer configuration.

Figure 28-11. Continuous Time Mixer Configuration

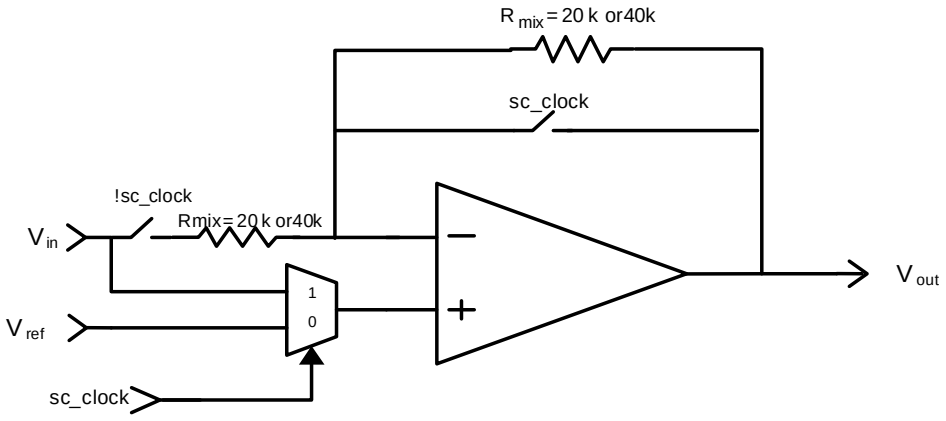


Table 28-10. Sampling Configurations for CT Mixer

SC_DYN_CNTRL	Configuration
0	Inverting Amplifier with Gain of 1
1	Unity Gain Buffer

Table 28-11. Input Resistor Settings for CT Mixer Inverting Mode

R20_40B	R_{MIX}
0	40 kΩ
1	20 kΩ

28.9 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 (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 desired 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. Set the sample clock frequency and ADC sample frequency to be 2 MHz.

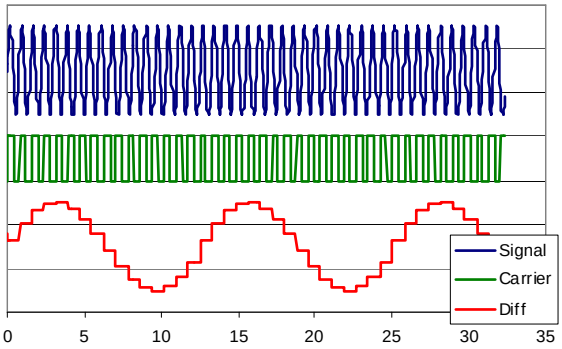
From the down conversion equations above, calculate the IF frequency with N = 7.

$$F_{IF} = 7 \cdot F_{CLK} - F_{IN} = 500kHz \qquad \textbf{Equation 10}$$

This example has 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.

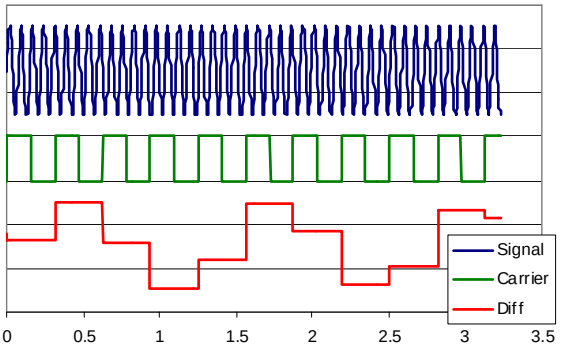
A few examples illustrate the frequency shifting capabilities of the mixer. For a signal frequency at 1.36 MHz, and a carrier at 1.28 MHz, the output frequency is the difference between the two frequencies, as shown in Figure 28-12.

Figure 28-12. 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 28-13.

Figure 28-13. Sampled Mixer N = 3

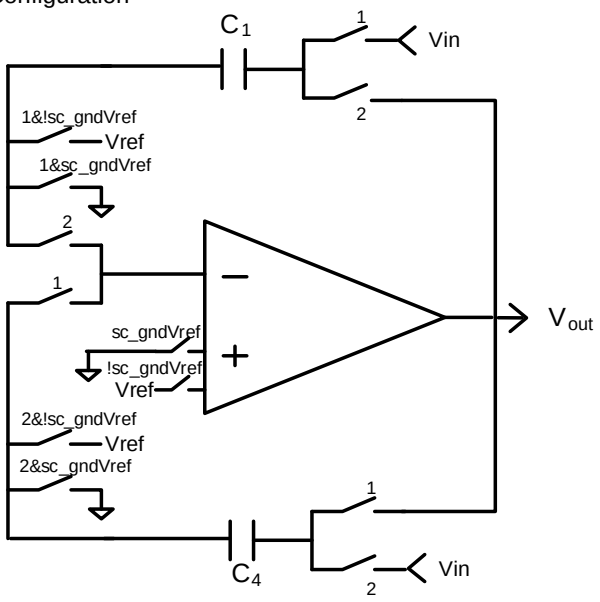


There is no increase in harmonic distortion, only an increase in the level of the sampling aliases. When the mixer output is sampled at the same rate as the carrier frequency, the aliases are suppressed.

The discrete time mixer configuration (NRZ S+H) is shown in Figure 28-14 on page 322. The options specific to this

configuration are the reference option and the clock division option.

Figure 28-14. 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 28-12.

Table 28-12. 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 28-13. Clock Division Option for Sample and Hold Mixer

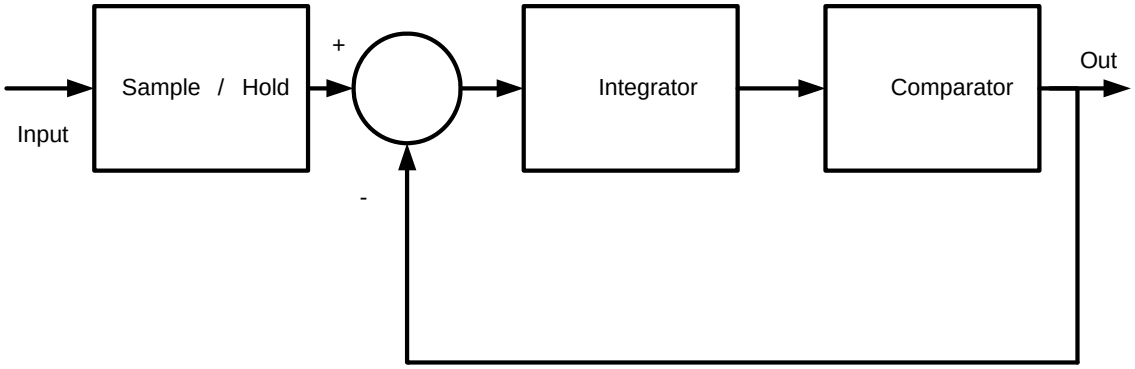
SC_DIV	SC_CLOCK Requirements
0	SC_CLOCK should be set to half the desired sample frequency
1	SC_CLOCK should be set to the desired sample frequency

28.10 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 28-15. 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 28-14 below shows the expected performance from a system simulation.

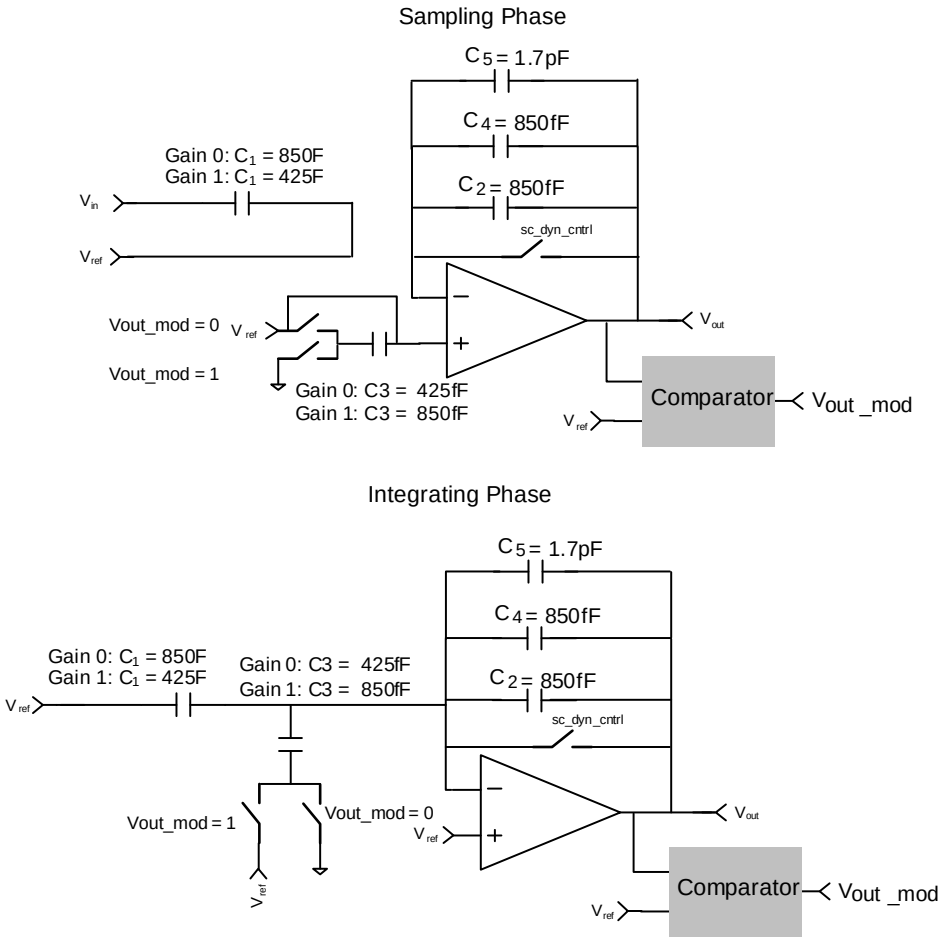
Table 28-14. Incremental Modulator Expected Performance from System Simulation

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 28-16 on page 324. 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 28-16. Switched Capacitor First-Order Modulator Configuration



28.10.1 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 28-15. 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 28-16.

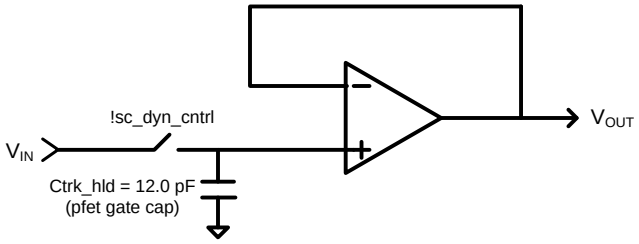
Table 28-16. First-Order Modulator, Input Amplitude

SC_GAIN	Maximum Input Amplitude
0	± half V_{REF}
1	± 2 V_{REF}

28.11 Track and Hold Amplifier

Track and hold amplifier mode is derived using the unity gain buffer amplifier. Implementation is shown in Figure 28-17.

Figure 28-17. Track and Hold Block Diagram



Track and hold mode tracks to 1% 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 28-17.

This feature is enabled by setting the register bit value SC[0..3]_CLK[5].

Table 28-17. Track and Hold Amplifier Control

SC_DYN_CNTRL	Output
0	Track V_{IN}
1	Hold sampled value

29. Analog Routing

PSoC® 3 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 flexible 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, on the fly, by DSI control signals and register settings. Knowing and understanding the architecture enables efficient and optimal utilization of the device analog routing resources.

29.1 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 analog local bus (abus)

29.2 Block Diagram

The PSoC 3 analog system block diagram is shown in Figure 29-1 on page 328. In Figure 29-1, the CapSense® system is limited to the GPIO controls, there are no separate blocks. Figure 29-2 on page 329 shows detailed analog routing architecture. All the figures used to explain analog routing in this chapter are derived from Figure 29-2.

Figure 29-1. Analog System Block Diagram

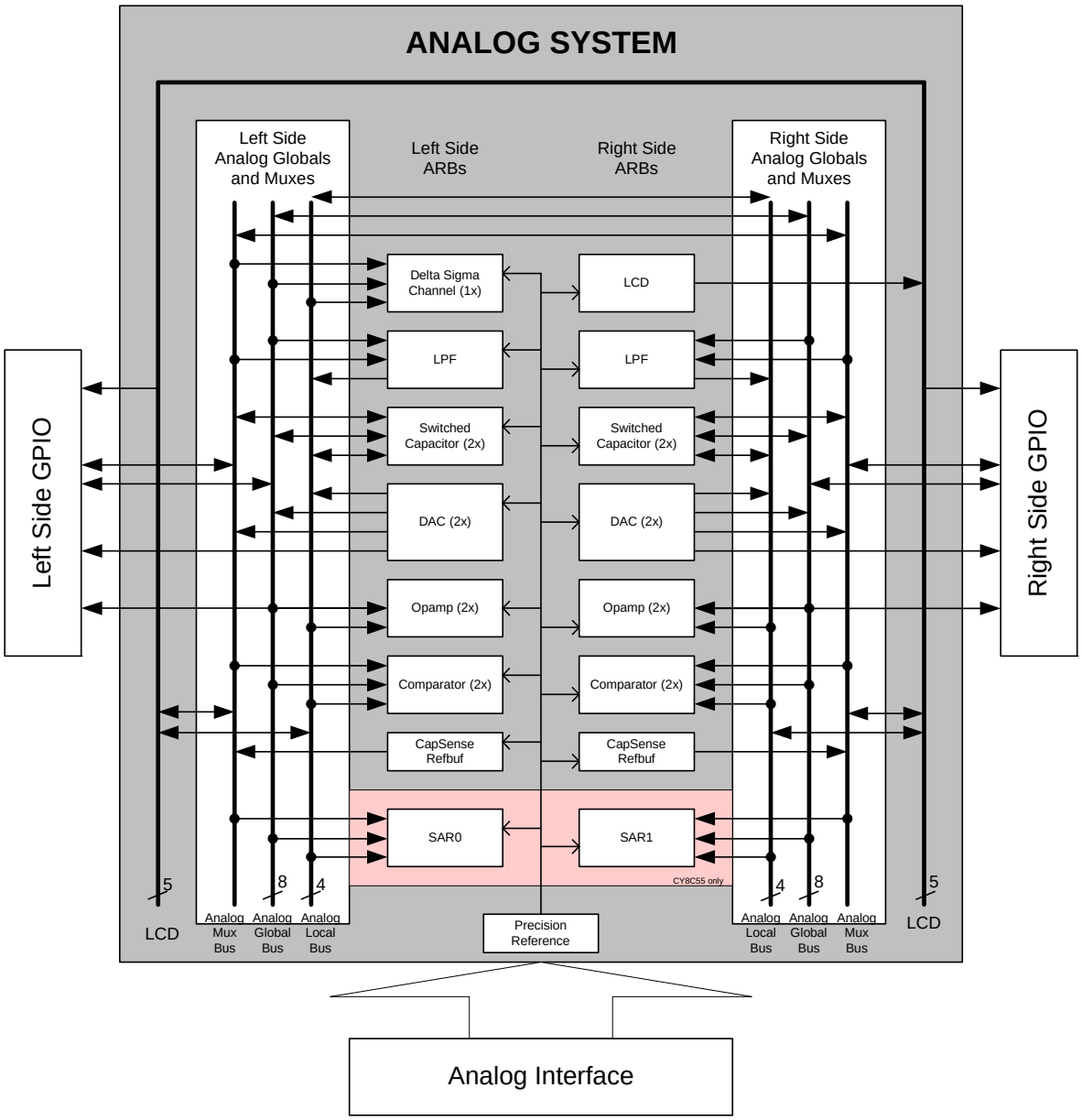
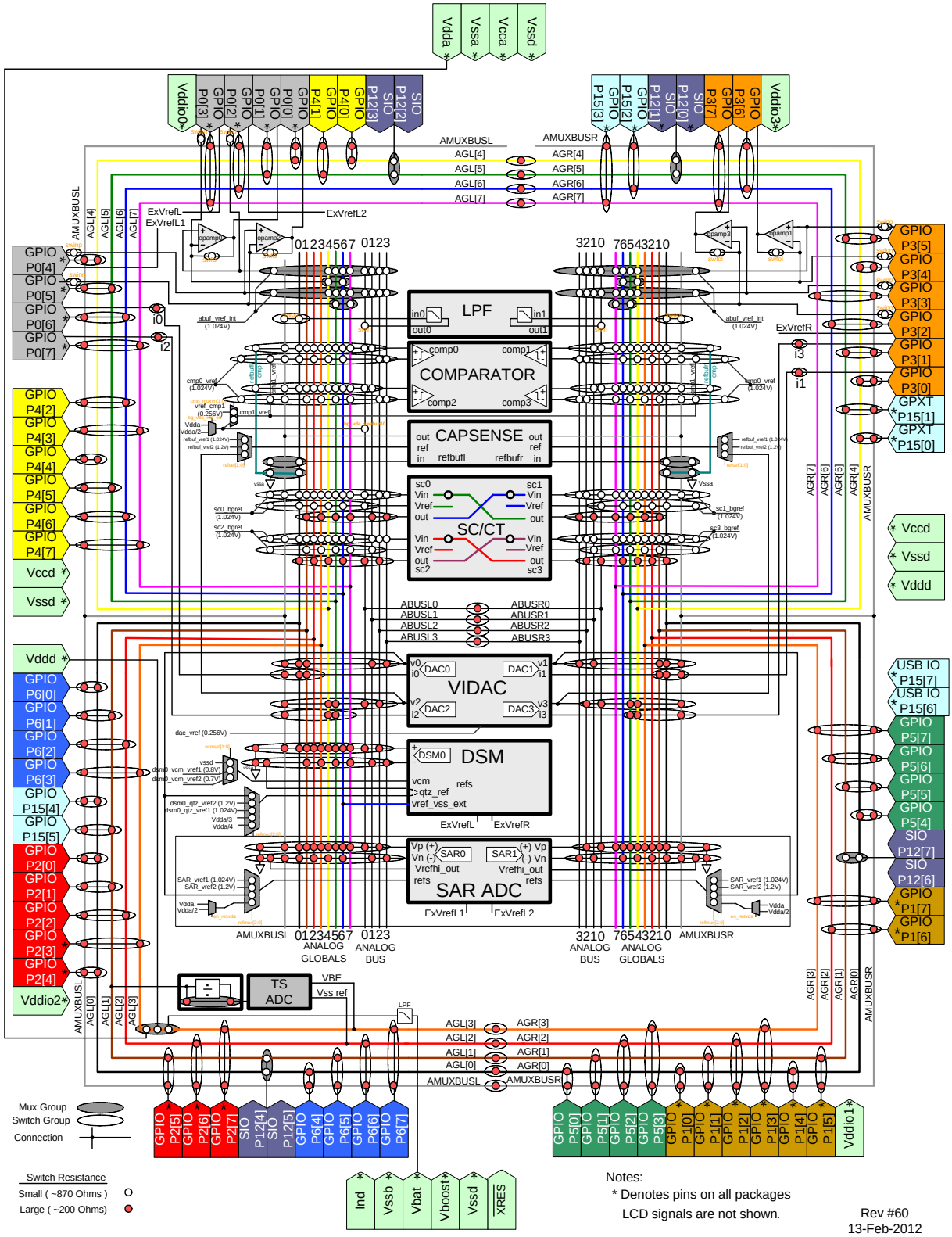


Figure 29-2. Analog Interconnect



29.3 How it Works

Analog routing resources in PSoC 3 devices include analog globals (AGs), analog mux bus (AMUXBUS), liquid crystal display bias bus (LCDBUS), and local analog buses (abus). The analog globals and AMUXBUS go to the GPIOs and provide a way to route signals between the GPIOs and the analog resource blocks (ARBs). The LCDBUS is used for LCD bias signal routing.

Analog resource blocks include the following: DACs, comparators, CapSense, switched capacitors, Delta Sigma ADC, and opamps. The analog local buses (abus) are local buses used for connections between ARBs.

In addition, there is a V_{REF} bus, as shown in Figure 29-2 on page 329. This V_{REF} bus carries the reference voltages for different analog blocks that are generated by the precision reference block. See the Precision Reference chapter on page 379 for details on 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 29-4 on page 332 illustrates the difference between switches and muxes.

29.3.1 Analog Globals (AGs)

The PSoC 3 die is divided into four quadrants, as shown in Figure 29-2 on page 329 and Figure 29-3 on page 331. 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 in the lower half of the die and the upper four globals on each side are routed to the GPIO in 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 may operate independently or they may be joined through the switches that are shown at the top and bottom of Figure 29-3 on page 331.

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, and CapSense (which is a virtual block). These connections are made through switches and muxes. PRT[x]_AG registers are used to configure the analog globals (AGs) for each GPIO port pin. See 29.6 Analog Routing Register Summary on page 341 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 29-2 on page 329. The mux selection is controlled by the {PRT12_AG} register. See the I/O System chapter on page 159 for details about SIO operation.

29.3.2 Analog Mux Bus (AMUXBUS)

There are two AMUXBUS routes in PSoC 3 devices. The 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 has the provision to connect to an AMUXBUS through an analog switch. CapSense applications use the AMUXBUS for their operation. See CapSense chapter on page 365 for details on using this bus for CapSense applications. PRT[x]_AMUX registers are used to configure the AMUXBUS routing for each GPIO port pin. See 29.6 Analog Routing Register Summary on page 341 for register details.

29.3.3 Liquid Crystal Display Bias Bus (LCDBUS)

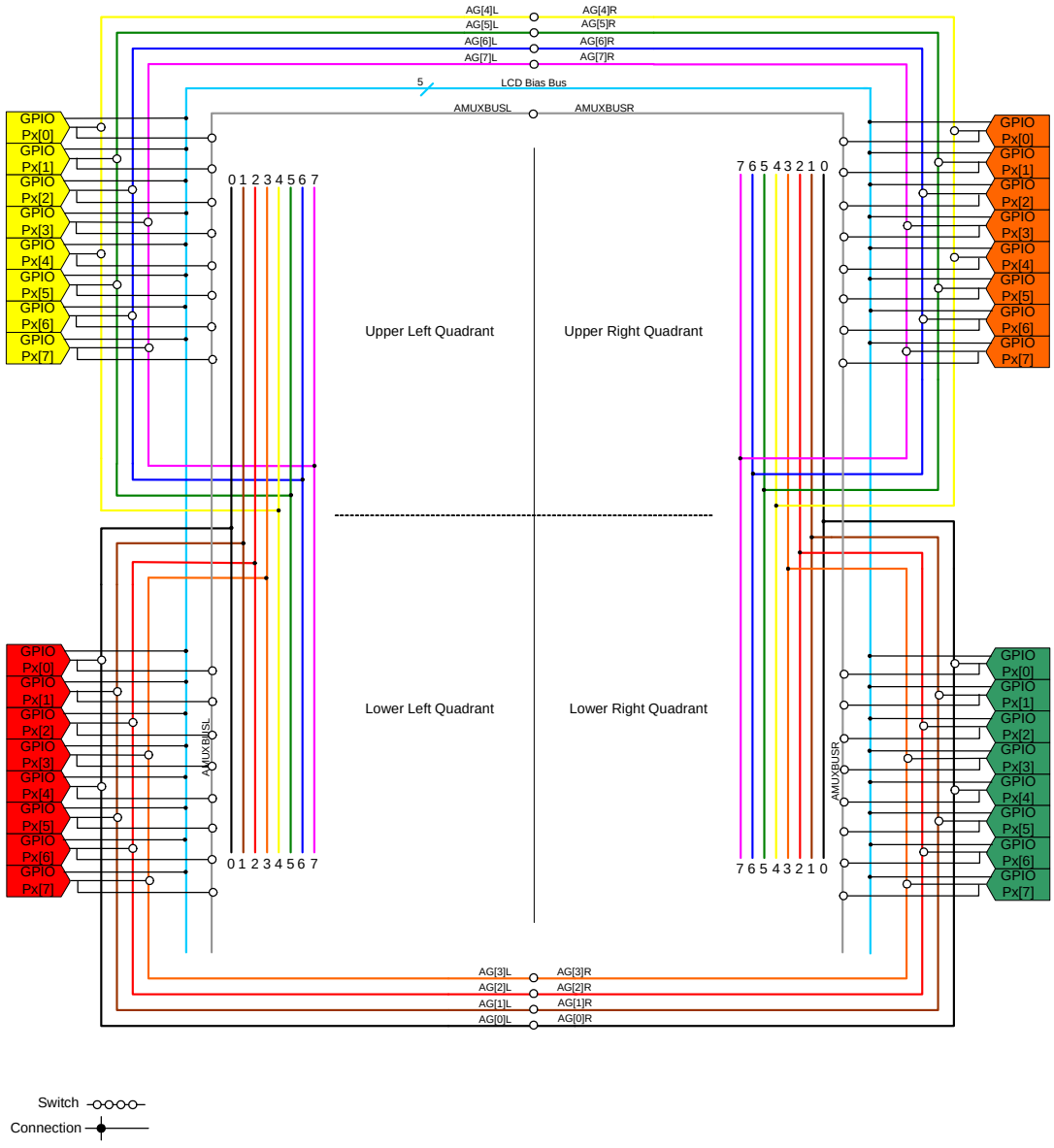
The LCD bias bus contains five routes that connect to every GPIO. These routes are continuous around the device periphery and are not separated by switches at the midline as are the analog globals and AMUXBUS. Each LCD route is individually configurable so that they are driven by the analog local bus or LCD bias voltage to the LCD driver buffer located in the GPIO. Connecting to an analog bus allows low frequency analog signals to drive off-chip through the LCD driver buffers.

The LCDBUS mux selections are given in the following table. See the LCD Direct Drive chapter on page 351 for LCD operation and biasing. See 29.6 Analog Routing Register Summary on page 341 for register details.

Table 29-1. 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 29-3. Analog Globals, AMUXBUS, and LCDBUS Routing



29.3.4 Analog Local Bus (abus)

There are eight analog local bus (abus) routes in PSoC 3 devices, four in the left half (abusl[0:3]) and four in the right half (abusr[0:3]), as shown in Figure 29-2 on page 329. These are local routes located in the analog subsystem and are for interconnecting ARBs, which reduces the usage of AGs. They do not route directly out into GPIOs. It is possible to short the left and right abus' together with four analog switches. ARBs may connect to each other through analog globals (AG) or the analog local bus (abus). For example, in Figure 29-2 on page 329, 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 can be placed on AGR0 and the comparator input switch can also be set to AGR0. Limited number of available analog globals (eight per side) and some block to block connections can be made through analog local bus for direct connections between blocks. For the above example, the DAC output (V1) can be routed directly to the analog local bus (abusr3) that goes to the negative input of the comparator (COMP1). This saves the GPIO routing resource from being used for interconnecting two ARBs.

29.3.5 Switches and Multiplexers

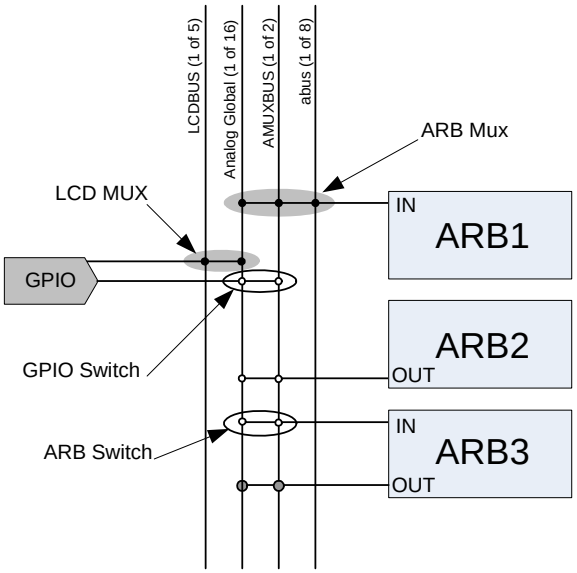
Switches and multiplexers are used to establish connections using different analog routing buses. They are placed on the various buses to direct signals into and out of the GPIOs and ARBs.

In a switch with 'n' inputs and one output, zero through 'n' switches may be on at a time, whereas in a multiplexer (mux) with 'n' inputs and one output, only one 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 29-4 illustrates the difference between switches and muxes, in switch and mux symbols.

For example, in Figure 29-4, 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 29-4. Difference Between Analog Switches and Muxes



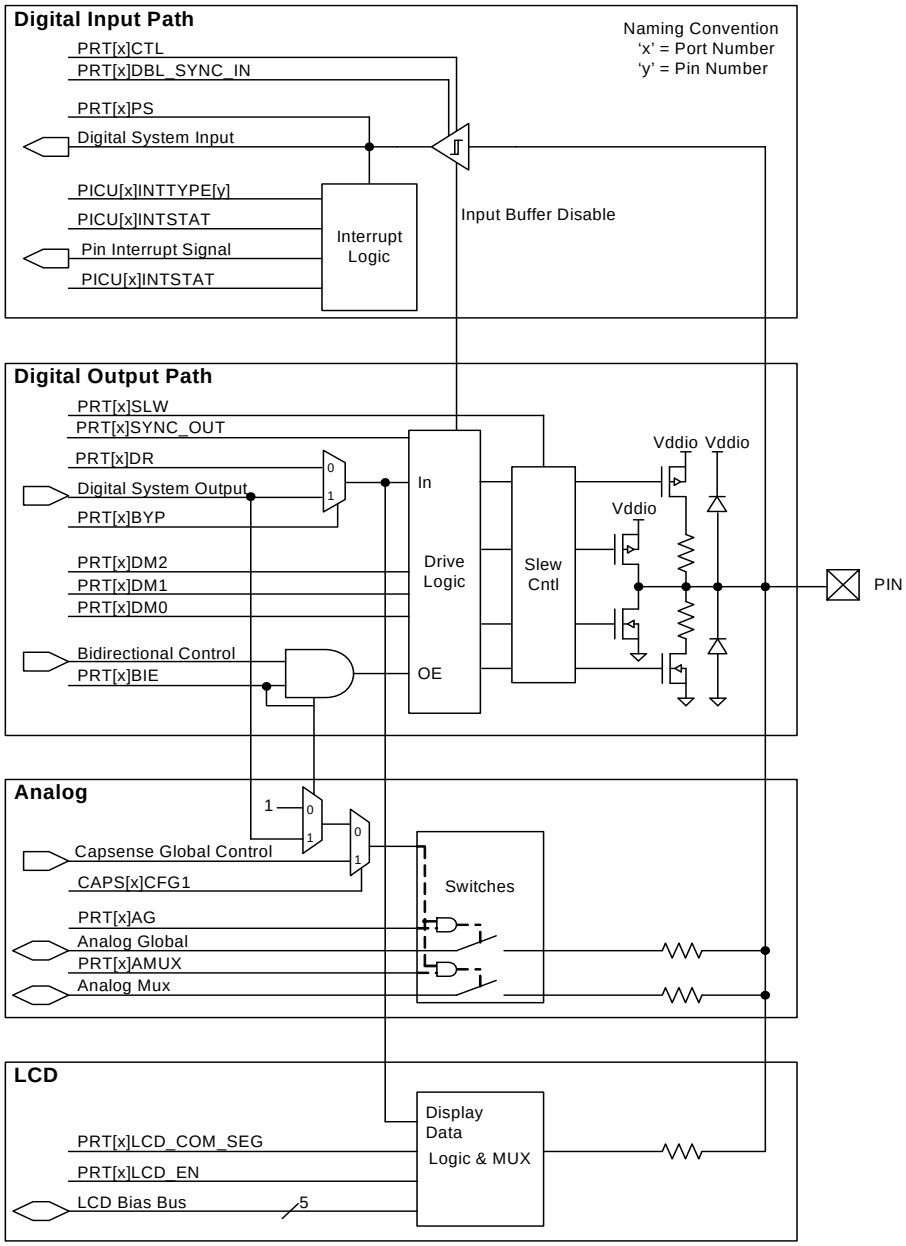
29.3.5.1 Control of Analog Switches

Analog globals (AGs), analog mux bus (AMUXBUS) and the analog local bus (abus) all use analog switches to establish connections. As stated earlier, analog switches can be grouped together to form multiplexers or 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. In addition, there is a provision 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 29-5.

Figure 29-5. GPIO Pin Input/Output Block Diagram



In addition, there are control signals that are dedicated for CapSense applications as shown in Figure 29-5. See the CapSense chapter on page 365 for the usage of these control signals.

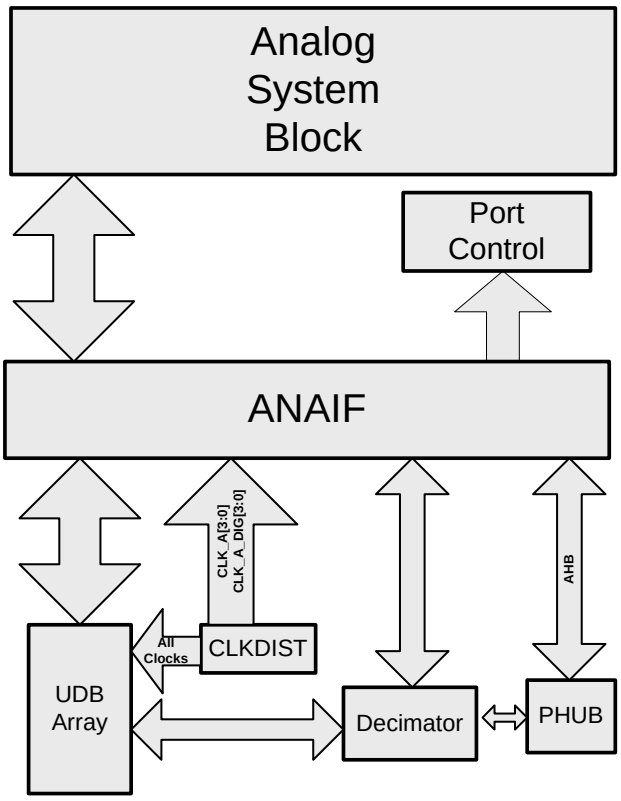
The analog switches corresponding to the analog resource blocks can be controlled only by the register settings of the respective ARBs.

For example, to switch 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 is configured.

29.4 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 output is routed to the digital systems interconnect (DSI) through the analog interface. The modulator output (digital) is routed to the decimator 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 interfaces are provided by the ANAIF are given in the individual chapters in Section F: Analog System on page 311. The following figure shows the top level diagram of the analog interface.

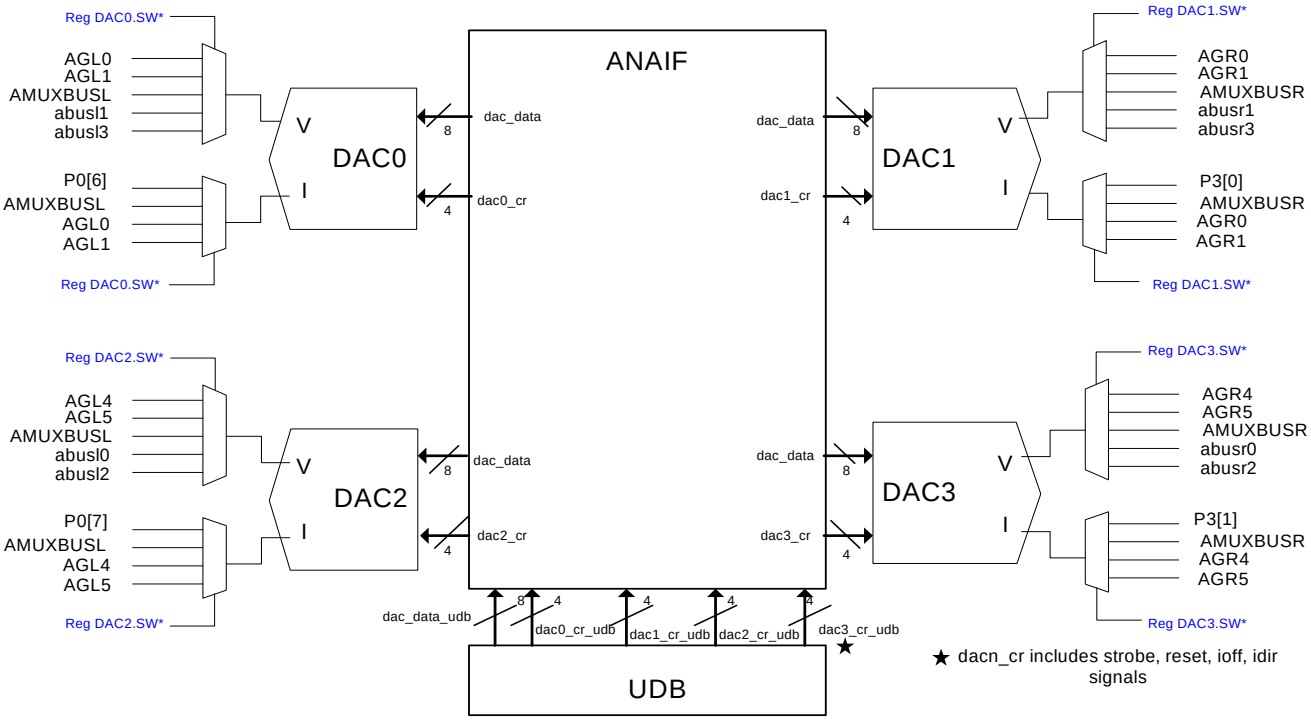
Figure 29-6. Analog Interface System Diagram



29.4.1 Digital-to-Analog Converter (DAC)

The DAC routing options and connections to other PSoC subsystems through the analog interface are shown in Figure 29-7. 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. See the Digital-to-Analog Converter chapter on page 375 to learn more about DAC control and operation.

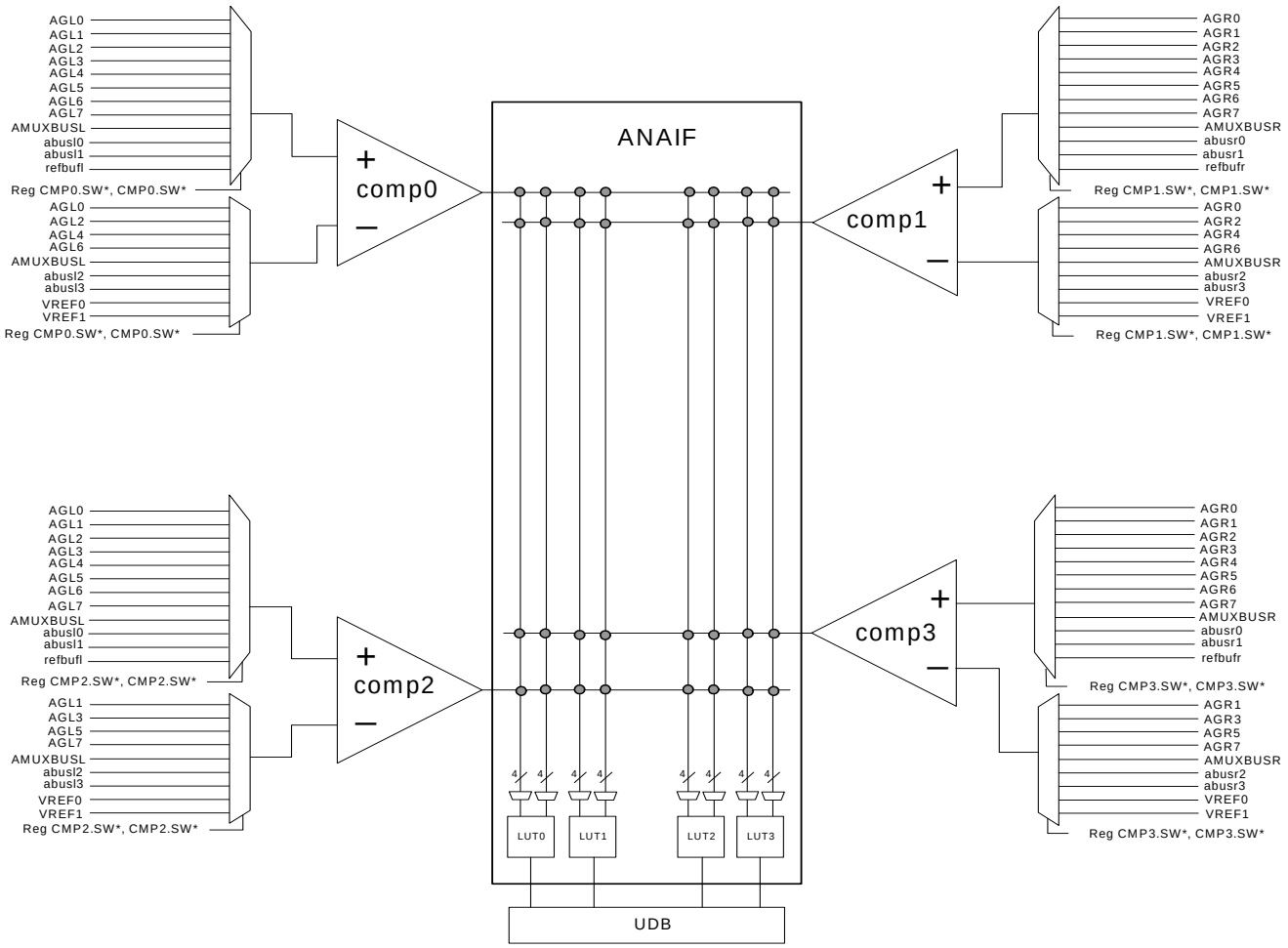
Figure 29-7. DAC Routing, Interface



29.4.2 Comparator

The comparator routing options and connections to other PSoC subsystems through the analog interface are shown in Figure 29-8. 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. 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 DSI. In addition, LUT outputs can generate interrupts (LUT_IRQ) to the device. See the Comparators chapter on page 343 to learn more about comparator control and operation.

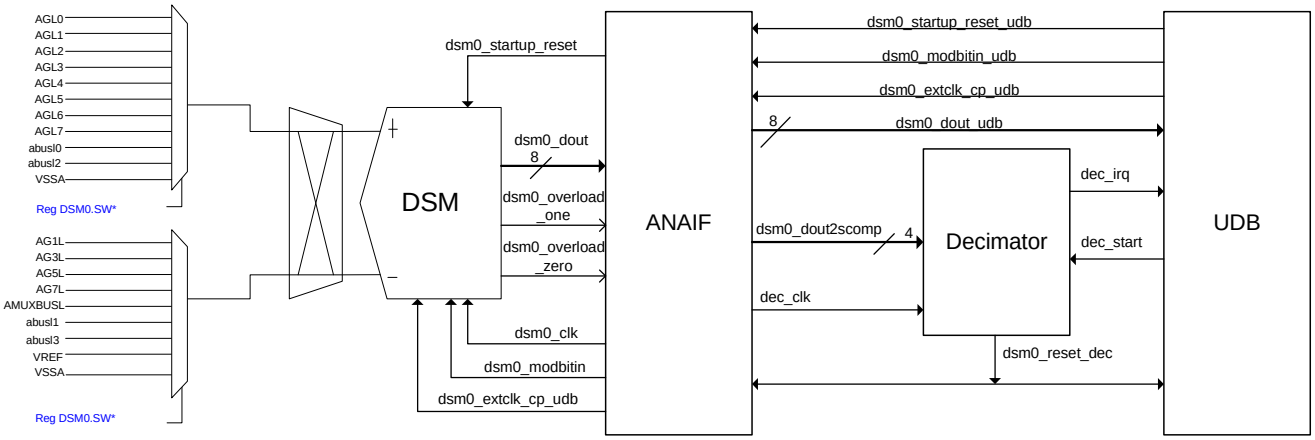
Figure 29-8. Comparator Routing, Interface



29.4.3 Delta Sigma Modulator (DSM)

The Delta Sigma modulator (DSM) is part of the Delta Sigma ADC and consists of various blocks that are mentioned in the Delta Sigma Converter chapter on page 383. 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. See the Delta Sigma Converter chapter on page 383 to learn more about the control and operation of this block.

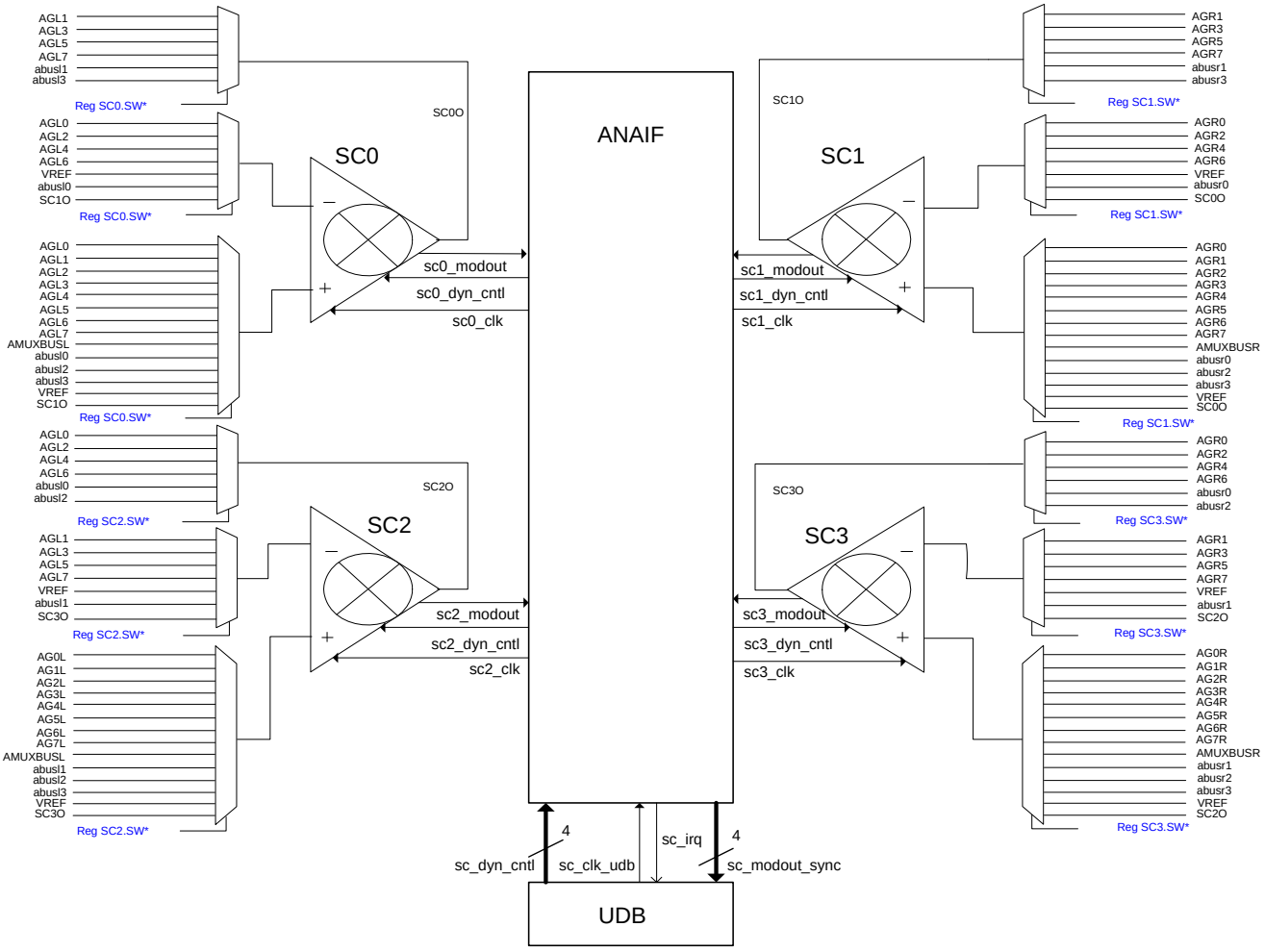
Figure 29-9. DSM Routing, Interface



29.4.4 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 29-10). The four analog clocks and the corresponding digitally aligned clocks, as well as 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. See the Switched Capacitor/Continuous Time chapter on page 313 to learn more about SC/CT.

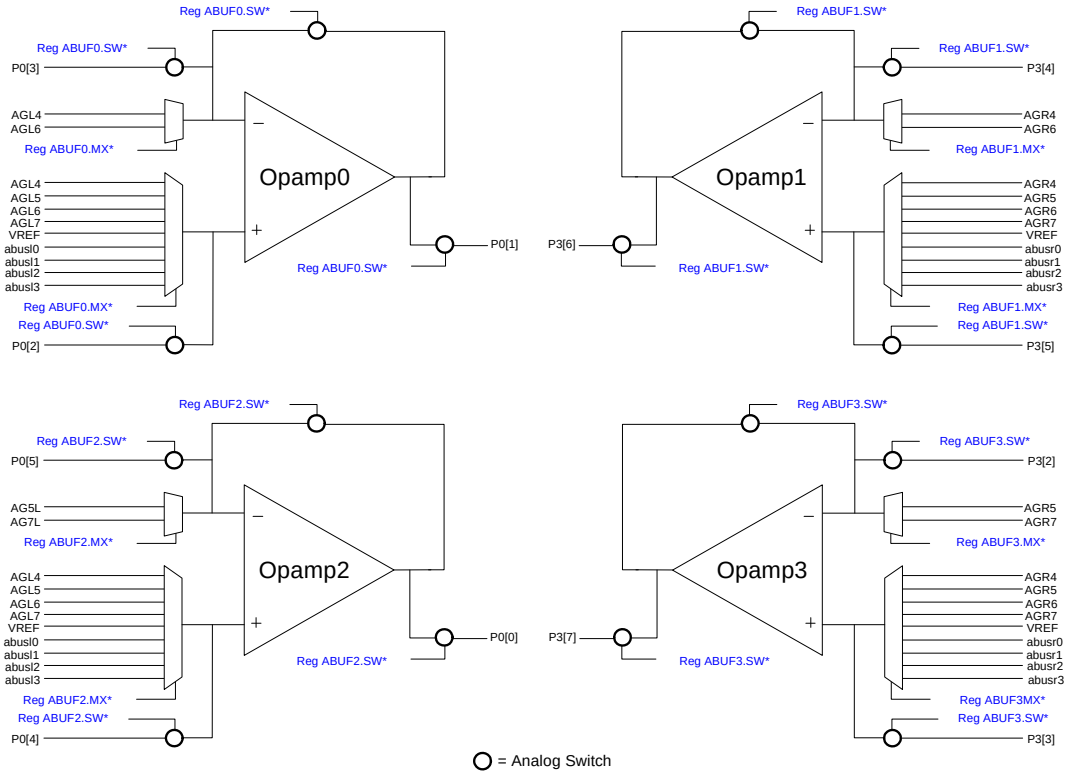
Figure 29-10. Switched Capacitor Routing, Interface



29.4.5 Opamp

The input and output routing options for the output buffer (opamp) are shown in Figure 29-11. See the Opamp chapter on page 347 for details on configuration and operation of this block.

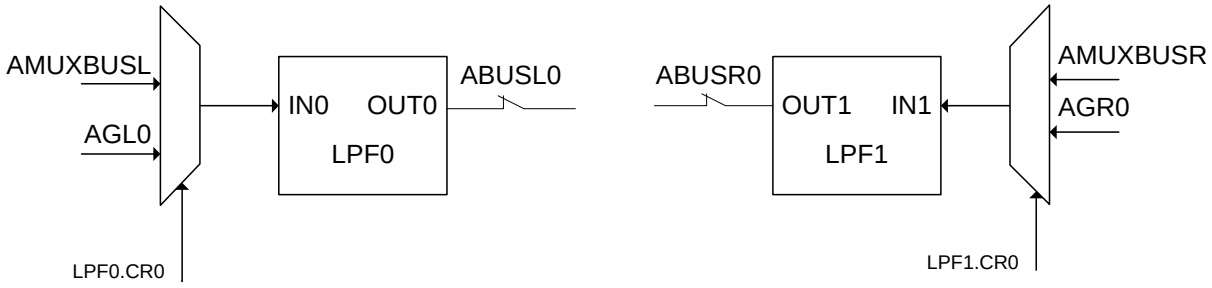
Figure 29-11. Opamp Input/Output Routing



29.4.6 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 29-12. 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 29-12. LPF Routing



29.5 Low-Power Analog Routing Considerations

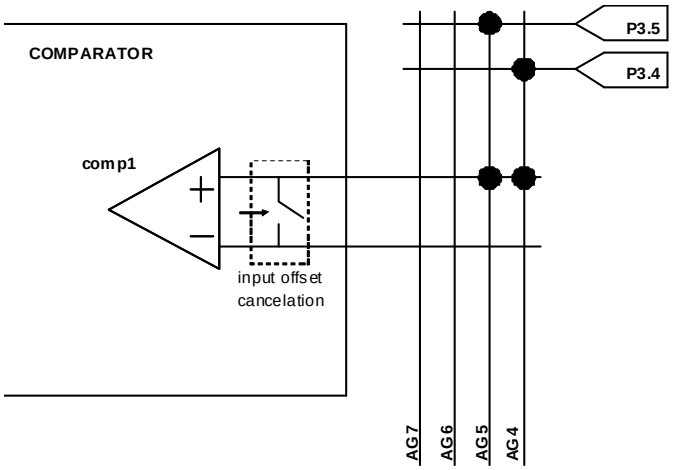
Figure 29-2 illustrates the analog global routing network, overlaid on top of the ARBs. Each ARB has a set of muxes and switches that it uses to connect to the global analog routing. By connecting to one of the analog routing channels virtually any ARB can be connected to any other ARB or pin on the 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 you want to connect P3.5 to P3.4 in a simple pass-through configuration. This is illustrated in Figure 29-13. This illustration is taken from the full chip diagram shown in Figure 29-2. P3.5 enters the chip on analog global AG5. P3.4 enters the chip on analog global AG4. To connect these two pins together, you need to track jump between AG4 and AG5. To do this, you can use the comparator ARBs comp1 positive input switches (assuming the rest of our project isn't using these switches). The switch for AG4 and AG5 on the comparator comp1 positive input is closed, while the rest of the switches remain open. The inputs to the comparator ARB itself are isolated from the switch group via a transmission gate.

After this configuration is programmed into the device, any signal seen on P3.5 will show up on P3.4.

Figure 29-13. Simplified Diagram of Routing P3.5 to P3.4 Using Track Jumping on the Positive Input of Comparator 1



29.5.1 Mitigating Analog Routes with Degraded Low-power Signal Integrity

The analog router in PSoC Creator uses track jumping to connect analog globals together. Track jumping is done on the muxes/switches of unused ARBs. The auto-router in PSoC Creator will always choose routes that ensure signal integrity in all power modes.

If the auto-router is not sufficient and you need to resort to manual routing techniques to realize a design, then take special care. For performance reasons, the SC/CT ARB controls the availability of all of its associated analog switches.

You can modify which analog routes are chosen by using the Manual Analog Routing (MARS) tool to force the routes. If you modify the analog routes chosen, by using the manual routing components in PSoC Creator, make certain to avoid routing that uses the SC/CT block for track jumping purposes if the SC/CT is not enabled in Hibernate/Sleep modes.

If a design uses SC/CT analog switches to realize a design in sleep/hibernate modes, but has this block powered down, significant degradation in signal integrity may be experienced. Explicitly Start() components derived from the SC/CT block and leave on in hibernate/sleep if it is necessary to still use these switches to route the design. Not starting the ST/CT block is equivalent to stopping it. By starting this block its routing resources become available for routing. See the PSoC Creator datasheet associated with Manual Routing for more details on how to use the MARS tool.

29.6 Analog Routing Register Summary

Table 29-2. 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) positive input to analog globals 0-7
{SC[0..3]_SW2}	SC positive input to analog local bus
{SC[0..3]_SW3}	SC positive input to AMIUXBUS and V_{REF} SC negative input to AMUXBUS and V_{REF}
{SC[0..3]_SW4}	SC negative input to analog globals 0-7
{SC[0..3]_SW5}	SC negative input to analog globals 8-15
{SC[0..3]_SW6}	SC negative input to analog local bus
{SC[0..3]_SW7}	SC output to AMUXBUS, other SC negative and positive 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.

Table 29-2. Analog Routing Register Summary *(continued)*

Name	Brief Description
{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

30. Comparators

PSoC® 3 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). The comparator can also be used to wake the device from sleep. An ‘x’ used with a register name denotes the particular comparator number (x = 0 to 3).

30.1 Features

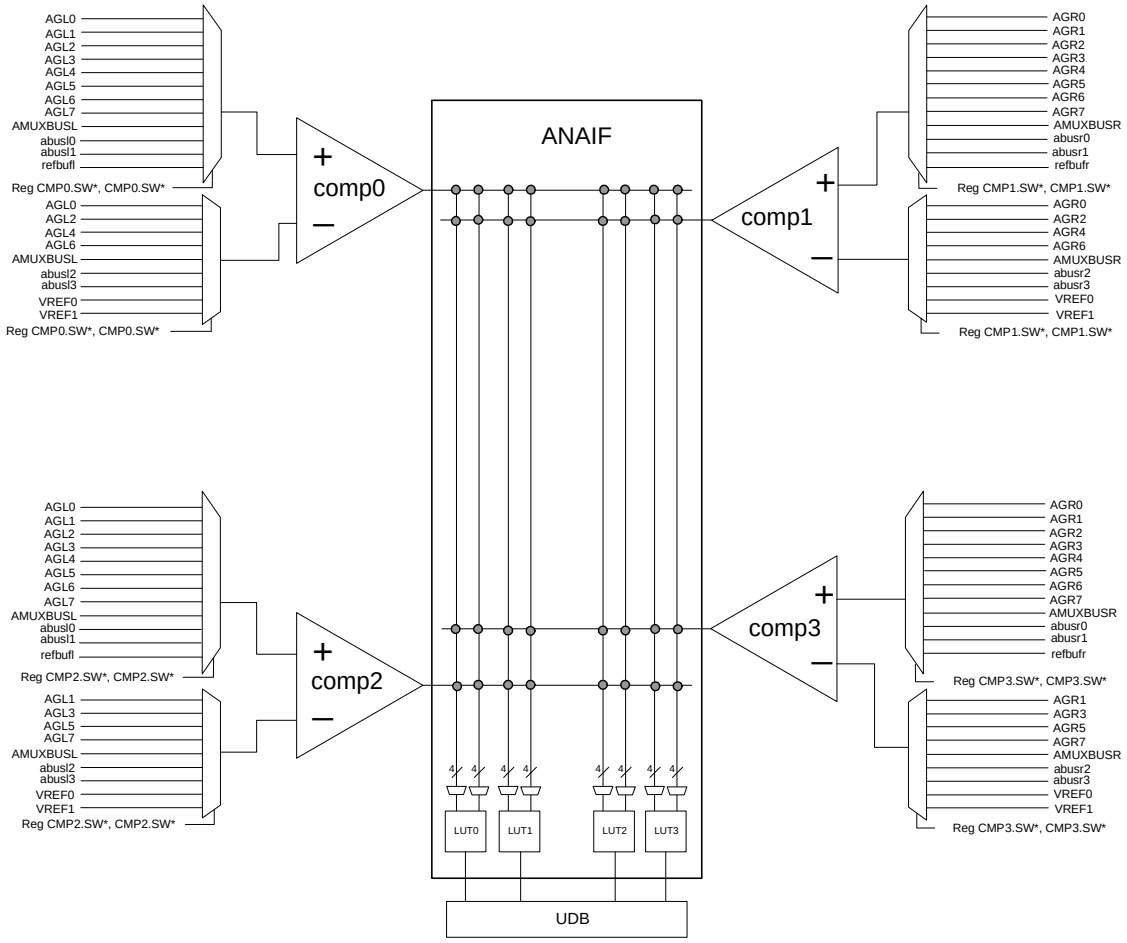
PSoC® comparators have the following features:

- Flexible input selection
- Speed power tradeoff
- Optional 10 mV input hysteresis
- Low-input offset voltage (<1 mV)
- Glitch filter for comparator output
- Sleep wakeup

30.2 Block Diagram

Figure 30-1 on page 344 is a block diagram of PSoC Comparators.

Figure 30-1. Comparator Block Diagram



30.3 How it Works

The following sections describe the operation of PSoC comparators.

30.3.1 Input Configuration

Inputs to the comparators are as follows:

- Positive – from analog globals, analog locals, analog mux bus, and comparator reference buffer. See the CapSense chapter on page 365.
- 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 30-1. Inputs are configured using registers CMPx_SW0, CMPx_SW2, CMP_SW3, CMP_SW4, and CMP_SW6.

30.3.2 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 datasheet.

30.3.3 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 30-1.

Table 30-1. 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.

30.3.4 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.

30.3.5 Wake Up from Sleep

The comparator can run in sleep mode and the output used to wake the device from sleep. Comparator operation in sleep mode is enabled by setting the override (PD_OVERRIDE) bit in the control (CMPx_CR[2]) register.

In low-power modes, the analog global pumps are disabled, increasing the resistance of analog routes for low-voltage applications that rely on the pumps in active mode. For best results, the device should be operated with VDDA at 2.7 V or higher. At lower voltages, the analog global routes will not meet their impedance specifications in low-power modes.

30.3.6 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.

30.3.7 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 you know 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 desired 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 offset polarity. Increase the value of trim2[2:0] until the offset measured is less than 1 mV.

30.3.8 Register Summary

Table 30-2 is a summary listing of applicable registers.

Table 30-2. 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

31. Opamp

PSoC® 3 devices have four operational amplifiers. An ‘x’ used with register name identifies the particular opamp number (x = 0 to 3).

31.1 Features

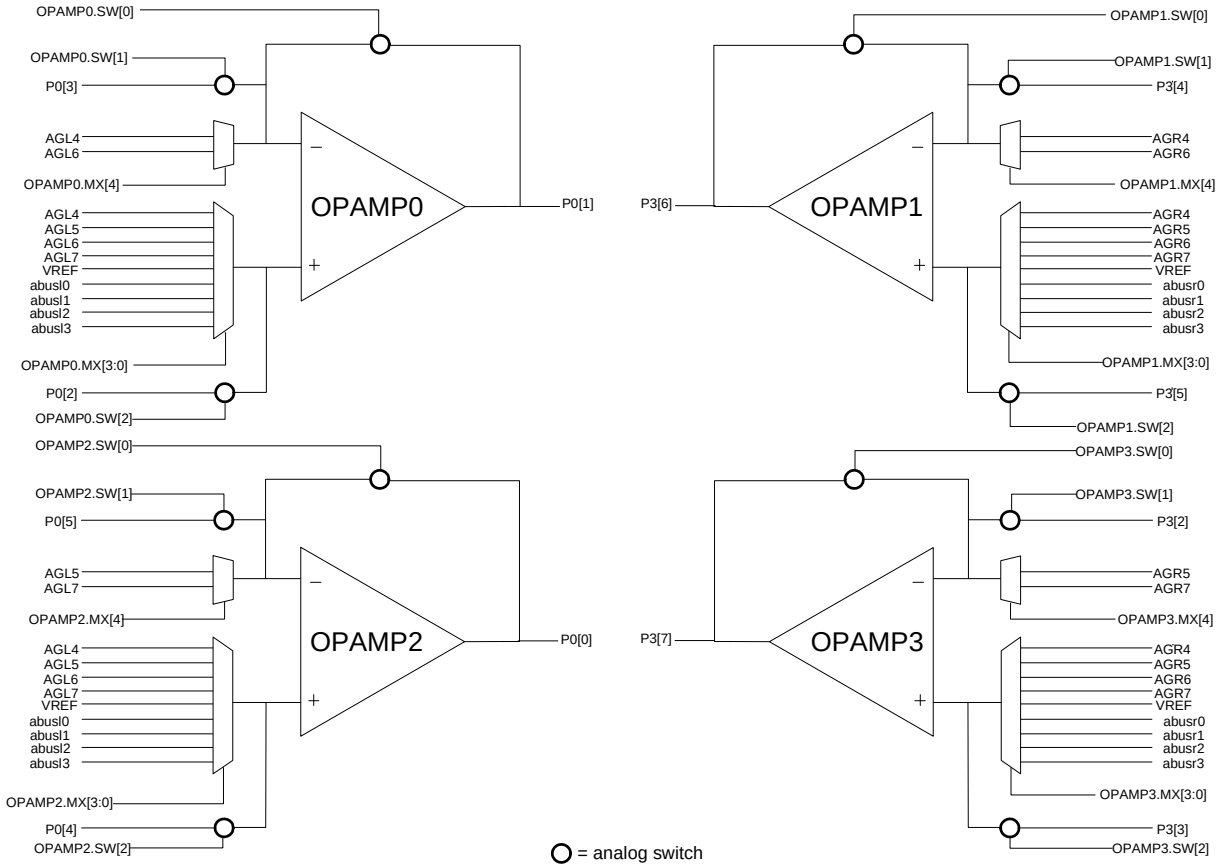
PSoC operational amplifiers have the following features:

- 25 mA current drive capability
- 3-MHz gain bandwidth for 200-pF load
- Offset trimmed to less than 0.5 mV
- Low noise
- Rail-to-rail to within 50 mV of Vss or Vdda for 1-mA load
- Rail-to-rail to within 500 mV of Vss or Vdda for 25-mA load
- Slew rate 3 V/µs for 200-pF load

31.2 Block Diagram

Figure 31-1 is the PSoC operational amplifiers block diagram.

Figure 31-1. Operational Amplifiers Showing Available Connections



31.3 How it Works

PSoC 3 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.

31.3.1 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 31-1. 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.

31.3.2 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. See the device datasheet for gain bandwidth and slew rate specifications in various power modes.

31.3.3 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.

31.3.4 Register Summary

Table 31-1 summarizes applicable registers.

Table 31-1. 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.

32. 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 LCDs. The flexible power settings allow this peripheral to be used in applications where a battery is the power source.

32.1 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 mux)
- 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

32.2 LCD System Operational Modes

PSoC 3 LCD architecture contains two operation modes.

- LCD always active
- LCD low power

LCD always active mode is used when the device is not in low-power mode and when the LCD does not need to be operational in device low-power mode.

LCD low-power mode is used when the LCD needs to be operational while the device is in low-power mode. This uses the same LCD always active system, but with some additional hardware.

The LCD drive system does not work when the chip is placed in hibernate mode.

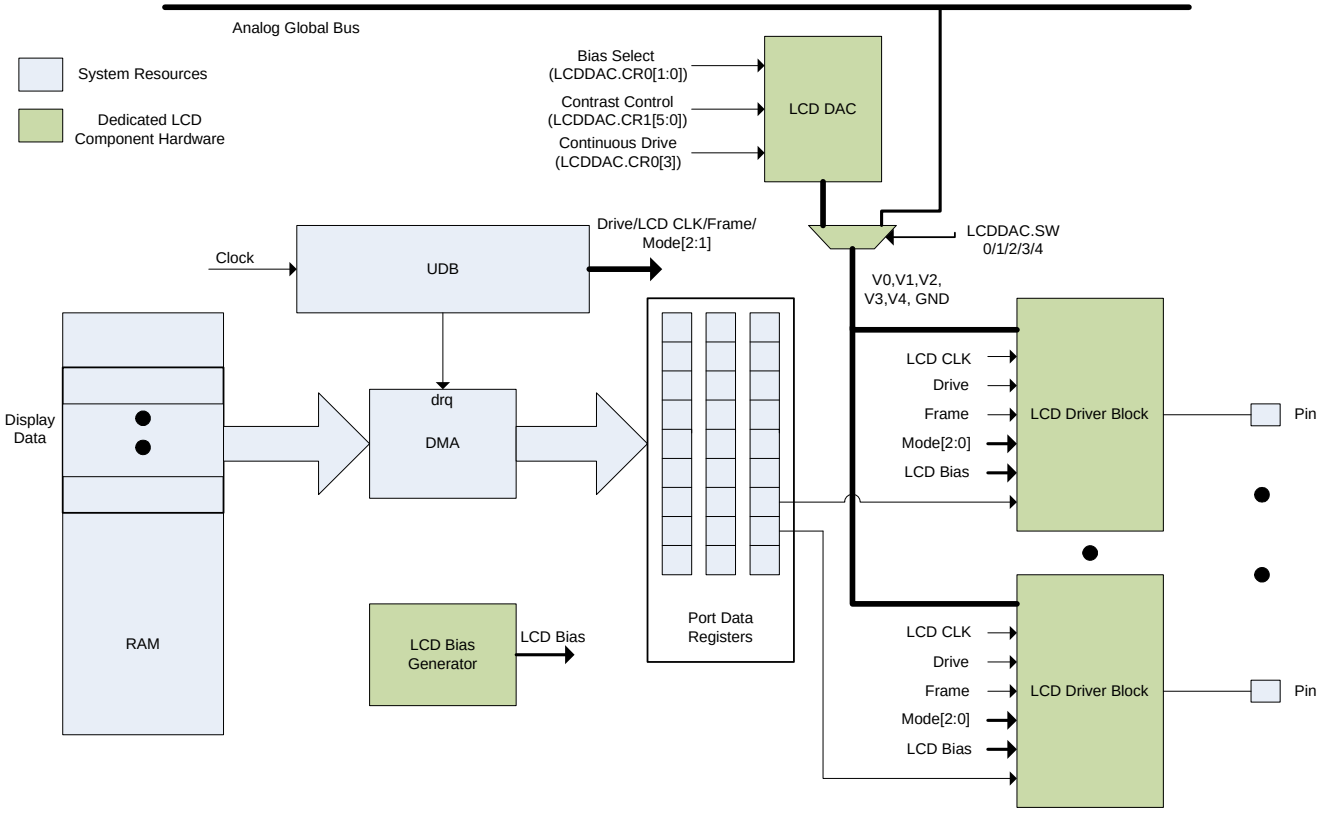
The details of both modes are discussed in the following sections.

32.3 LCD Always Active

A complete functional LCD always active drive system is formed using the following major blocks:

- Dedicated LCD hardware
  - LCD DAC
  - LCD driver
  - LCD bias generator
- System resources
  - DMA
  - Clocks: global
  - RAM
  - Universal digital block (UDB)

Figure 32-1. LCD Always Active System



Any LCD drive system requires the bias generating circuitry and system to interpret the data supplied, to display correctly on the LCD. PSoC 3 contains 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, along with ground. These bias voltages are distributed to all of the drivers of the LCD-capable pins. This DAC also helps to set contrast control.

LCDs have two sets of pins: commons and segments. LCD functionality in PSoC 3 GPIOs can be enabled by setting the appropriate bits of the PRT[0..11]_LCD_EN register. These 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 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. It also drives the pin, depending on the LCD refresh state, whether the pin is configured as common or segment, and the display data.

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 of the control signals required by the rest of the blocks of the LCD system.

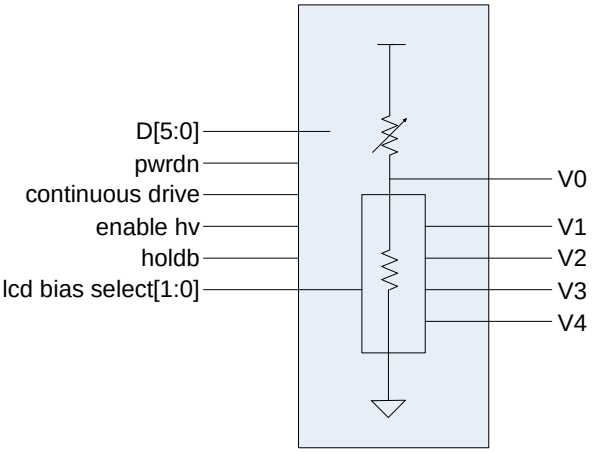
32.3.1 Functional Description

This section provides details of the LCD DAC, LCD driver, UDBs, clocking, DMA, CPU, and RAM, which all contribute to generating and sequencing the driving voltage for the LCD glass.

32.3.1.1 LCD DAC

The LCD DAC is a 6-bit resistor ladder DAC. The LCD DAC is responsible for contrast control and bias voltage generation for the LCD drive system. When the device is put in low-power mode, the LCD can remain operational. During this low-power mode, the DAC can directly drive the LCD pixel, bypassing the driver, thus compensating for the leakage. This is possible in LCD low-power mode, which is explained in section 32.4 LCD Low-Power Mode on page 357.

Figure 32-2. LCD DAC (inputs and outputs)



The LCD DAC generates five voltages that are driven to LCD driver block. Important points regarding LCD DAC are:

- All of the voltages V0 to V4 are generated using an internal resistor divider; V0 is the highest voltage and V4 the lowest voltage. By default, the five bias voltages, V0 to V4, from the LCD DAC are driven to each of the 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 the power by switching off the internal DAC. In this mode, there is no software contrast control available.
- The LCD DAC can directly drive the LCD pixel, bypassing the LCD driver block. This is useful for driving the LCD even when the chip is put to sleep. You can do this by setting the LCDDAC.CR0[3] bit, which enables the continuous drive of the LCD DAC.

32.3.1.1.1 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[5:0]), which sets the 6-bit DAC input (D[5:0], as shown in Figure 32-2). Thus, it provides 2 ^ 6 = 64 levels of contrast. Table 32-1 shows the V0 range and step size for 3.0-V and 5.5-V supply voltage.

Table 32-1. LCD DAC V0 Range and Step Size

	3.0 V Supply	5.5 V Supply
V0 Range	2 V to 3 V	2 V to 5.5 V
Step Size	27.3 mV	50 mV

32.3.1.1.2 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 32-2 on page 354.

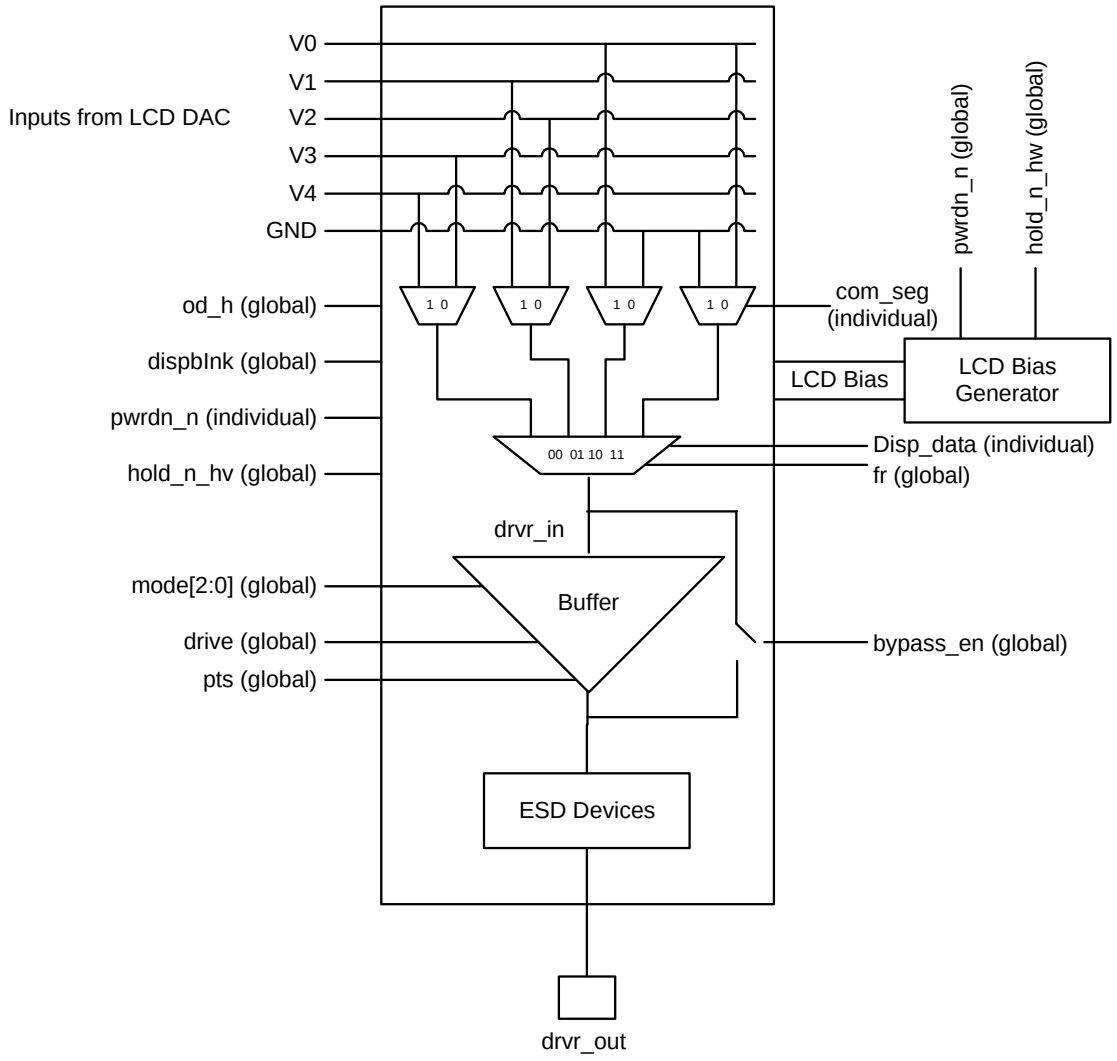
Table 32-2. LCD DAC Bias Select

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	8:1	1/4	2.0 V to supply	0.750 × V0	0.500 × V0	0.500 × V0	0.250 × V0
00	4:1	1/3	2.0 V to supply	0.666 × V0	0.333 × V0	0.666 × V0	0.333 × V0
00	3:1	1/3	2.0 V to supply	0.666 × V0	0.333 × V0	0.666 × V0	0.333 × V0
00	2:1	1/3	2.0 V to supply	0.666 × V0	0.333 × V0	0.666 × V0	0.333 × V0

32.3.1.2 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. Figure 32-3 shows the architecture of the LCD driver block.

Figure 32-3. 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 32-3 on page 354, 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 32-3, 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 32-3 shows the 4:1 multiplexer output and driver input for different combinations of COM_SEG, DISP_DATA and the frame signal.

Table 32-3. LCD DAC Output Selection

com_seg	disp_data	fr	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

Note For proper functionality of the LCD driver, the port reset drive mode (PRTxRDM[1:0]) NVL bits of any port used for LCD drive must be configured as a high-Z input (00b). By default, these bits are set to the proper configuration. Ensure that the bits are not altered in an LCD application; otherwise, the drivers will fail to produce proper LCD drive signals. For more details on these registers and their configuration, see the Nonvolatile Latch chapter on page 99.

32.3.1.2.1 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 eight modes of operation, selectable from the Mode[2:0] bits. Mode[0] comes from LCDDRV_CR[1]; the remaining two bits are driven from the UDB through the digital system interconnect (DSI).

Each mode has a different power drive capability. Depending on the LCD, the appropriate one can be used to eliminate AC coupling between segment and common lines. Note that these buffer power modes are different than the I/O drive modes.

Table 32-4. LCD Drive Modes

Control Bits			Mode	Drive Strength
Mode[2]	Mode[1]	Mode[0]		
0	0	0	High Drive	Seg = 1x, com = 1x
0	0	1	High Drive	Seg = 1x, com = 2x
0	1	0	High Drive	Seg = 1x, com = 4x
0	1	1	High Drive	Seg = 2x, com = 2x
1	0	0	High Drive	Seg = 2x, com = 4x
1	0	1	High Drive	Seg = 4x, com = 4x
1	1	0	Low Drive	Seg = 0.1x, com = 0.1x
1	1	1	Low Drive	Seg = 0.2x, com = 0.2x

The LCD display size and capacitance and the application power budget are two criteria for selecting buffer modes. The buffer is enabled only when the drive signal is high. Drive signal high time can be configured according to the application requirements. The drive current provided by the High Drive mode of the buffer (the mode that is normally used) is high, so it charges the pixel capacitance quickly. The disadvantage of this is higher power consumption. The time for which the buffer is kept on depends on the power budget and the LCD waveform's rise time requirements. The Low Drive mode of the buffer and the DAC are other options. It is possible to dynamically select the Low Drive mode by two mode control signals generated by the UDB. You do this in the case of extremely leaky glasses, when it is preferable to use the buffer to drive the LCD continuously throughout the refresh period. This is more effective than using the DAC, whose current drive ability is lower than that of the buffer Low Drive mode. Use the DAC when you have normal glasses and the charge leakage is small. If the leakage is small enough for the offset to be negligible, then the pin can be tristated by clearing the bypass_en bit, after charging the pixel using the High Drive buffer mode.

In normal operation, the buffer in High Drive mode drives the LCD for a while, then a low-power source (either the DAC or the buffer in Low Drive mode) takes over and drives the LCD for the remaining time.

- When using High Drive and DAC:

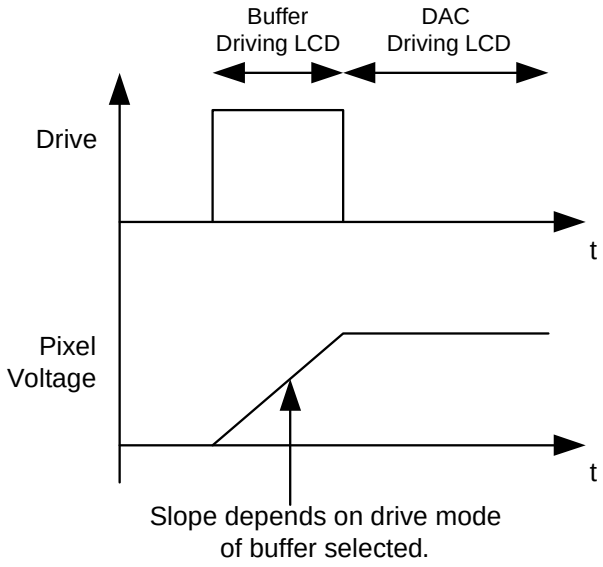
  Initially, for some period of time, the buffer quickly charges the LCD pixel capacitance near to the desired value. Later, when the drive signal goes low, the DAC directly drives the LCD for the remaining period (if the bypass_en bit is set) to sustain the voltage at the LCD

pin. If the bypass_en bit is not set to 1, the pin is tristated and no source drives the LCD. This can lead to charge leakage from the pixel capacitance.

- When using High Drive and Low Drive:

  The drive signal always remains high. This means that the buffer is always enabled. The UDB controls the time for which the buffer remains in High Drive and Low Drive modes.

Figure 32-4. The DAC Charging an LCD Segment Pin in Two Different States

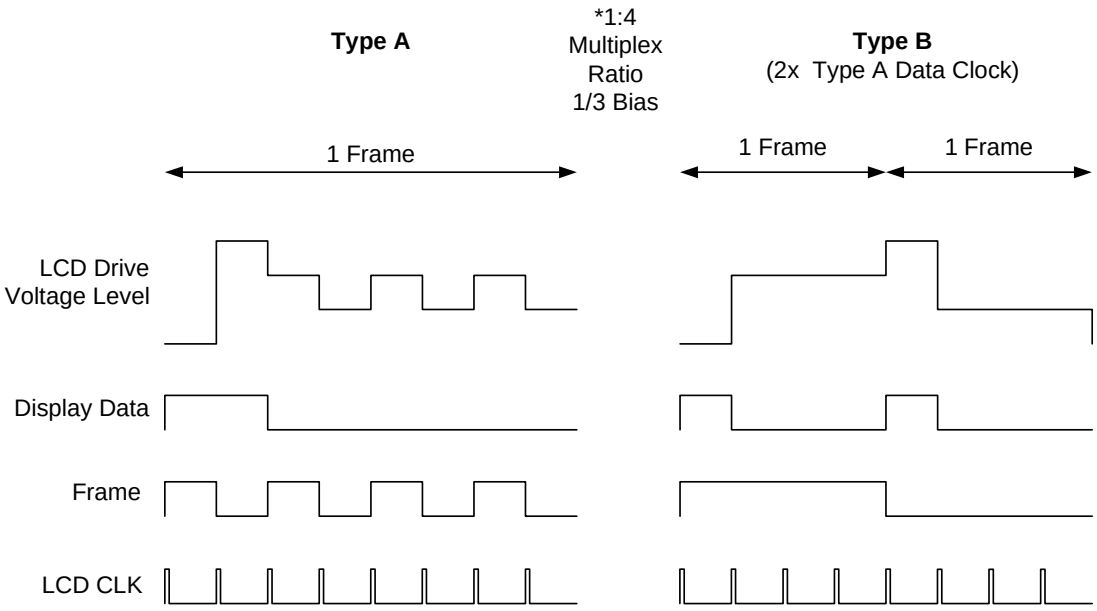


32.3.1.2.2 LCD Driver Bias Generator

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.

Figure 32-5 shows various control signals to the LCD driver block.

Figure 32-5. Control Signals of LCD Driver Block



32.3.1.3 UDB

The UDB performs the following actions in the LCD system:

- Triggers the DMA periodically to bring the display data from SRAM to the port data registers
- Generates various control signals for the functioning of the LCD system hardware
  - The drive signal, which is used to enable the driver buffer
  - Two mode control signals for the buffer
  - A synchronous LCD CLK, which is used to latch the port data register value for a particular pin
  - The frame signal

The clock for the UDB is derived from the IMO. The clock value changes with the refresh rate and the number of commons of LCD.

32.3.1.4 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.

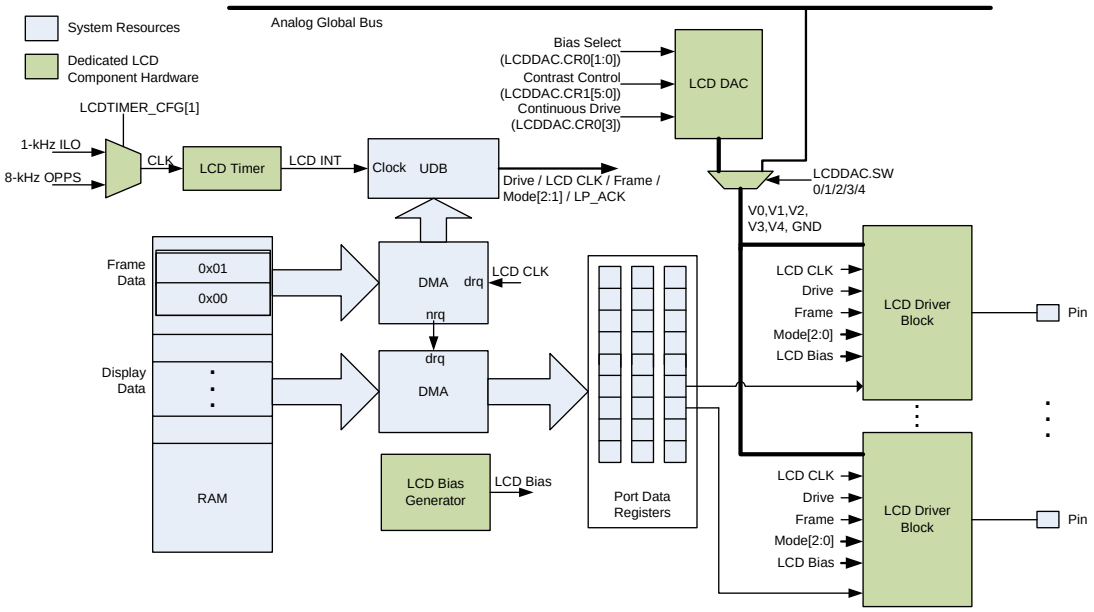
An additional set of registers (per port), the PRTx_BIT_MASK registers, mask off the write capability to the PRTx_DR_ALIAS registers on a bit level. This is an advantage if all of the pins on a given port are not being used for LCD; the unused pins can be masked off and used for other purposes. The port data register (PRTx_DR) can still be used to address pins masked off in the aliased data registers.

32.4 LCD Low-Power Mode

This mode is useful when LCD is required to be functional while the device is in low-power mode. This requires special hardware and firmware logic to wake the system up at regular intervals, refresh the LCD, and put the device back to sleep. Periodic refresh should happen at the specified rate, even if there are other interrupts in the system.

LCD low-power mode uses all the of components that are used for LCD always active mode. In addition to this, it also uses a programmable wakeup source and small dedicated digital logic to allow bug-free transitions to and from the low-power mode. Figure 32-6 on page 357 shows the block diagram for the LCD low-power mode.

Figure 32-6. LCD Low-Power Mode



A complete functional LCD low-power system is formed using these major blocks:

- Dedicated LCD hardware
  - LCD timer
  - LCD DAC
  - LCD driver
  - LCD bias generator
- System resources
  - Clocks: 1-kHz ILO and 8-kHz one pulse per second (OPPS)
  - UDB implementation for sleep acknowledgement
  - DMA for frame data transfer
  - UDB implementation for control signal generation (frame, drive, LCD mode, LCD CLK)
  - DMA for display data transfer

The blocks in bold are unique to the LCD low-power system. The other blocks are same as the LCD always active system.

What makes the LCD low-power system different from the LCD always active system?

- It can wake the system
- It can continuously drive the LCD even when the chip is put in low-power mode

PSoC 3 contains several clock sources that operate during device low-power mode. ILO and OPPS timer are examples. These clock sources are used to trigger periodic interrupts to the device to wake the system up. As shown in Figure 32-6, these two clock sources are selectable using a mux. The selected clock is fed to the 6-bit LCD timer. It is a continuously running timer; that is, when the timer overflows, the original period is reloaded in the timer register. The terminal count pulse from this timer triggers the interrupt to the chip. This restores the main clocks of the chip. When this happens, the interrupt signal from the LCD timer is intercepted by the UDB-implemented pulse generator. In response, the block generates a synchronous clock that causes several operations. See 32.4.1.2 UDB on page 358 for more details. Overall, the UDB's role is to provide control signals to various functional blocks of the LCD low-power system.

At this time, the system must be put back to sleep after the LCD refresh. In an LCD low-power system, the CPU issues a chip low-power (LP) mode command to the power management (PM) controller. (For this, firmware needs to be structured in a specific way, as explained in a later section.) Consent is given by the LCD hardware.

This is because the LCD refresh happens in hardware and CPU does not know when it is completed. So, a control signal (LP_ACK signal shown in Figure 32-6) is generated from the UDB, which keeps the LP command from the CPU on hold until the LCD refresh is completed. This control signal is driven to the power management controller of the device.

There are two DMAs used in this architecture. One DMA is used for the transfer of display data to the port data register, which is the same as in an LCD always active system. the other DMA is used to update the frame information into the control register of the UDB each time the chip wakes up.

32.4.1 Functional Description

This section gives details of the blocks and features used specifically in LCD low-power mode.

32.4.1.1 LCD Timer

The LCD timer is a 6-bit timer dedicated only for the LCD drive application. Its period is set based on the required refresh rate of the LCD. The period of this timer can be configured by setting the period field of the LCDTMR_CFG register. There are two options for the LCD timer clock source:

- 1-kHz ILO
- 8-kHz OPPS. This requires that an external 32-kHz crystal be connected to the system.

The source can be selected by setting the clk_sel field of the LCDTIMER_CFG register.

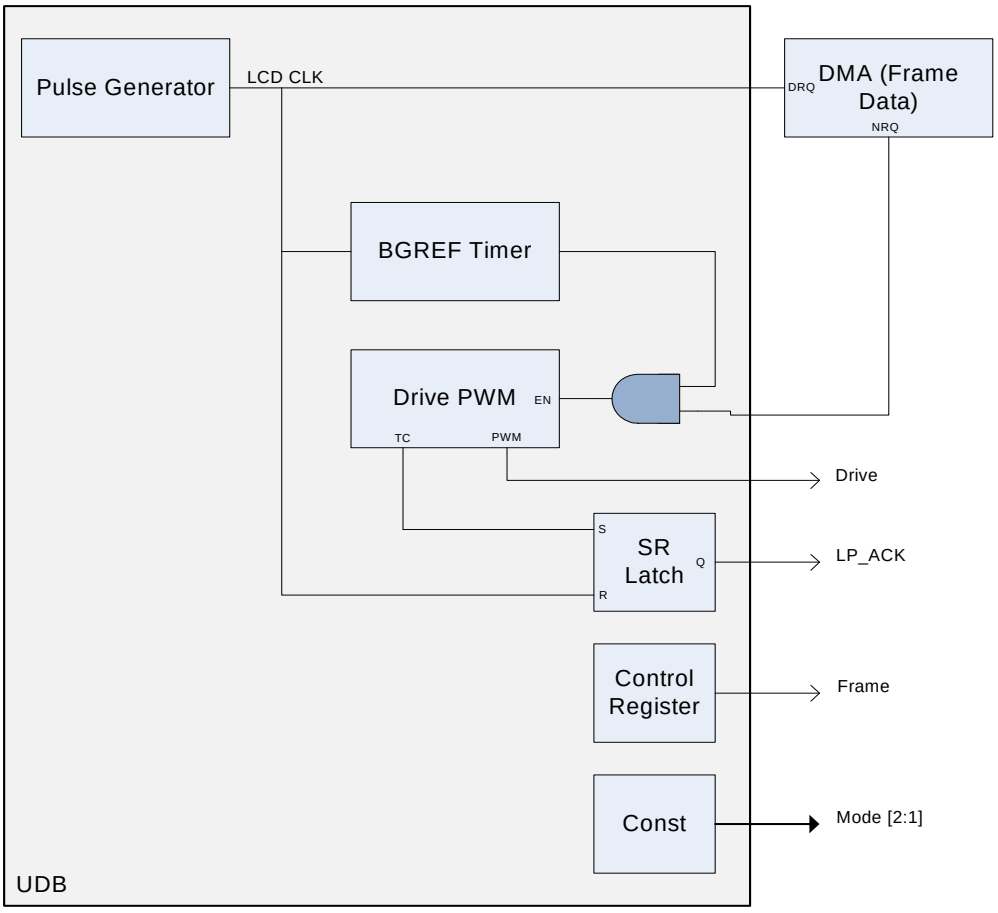
The clock timer provides periodic interrupts to the system PM controller. The interrupt signal is also driven to the UDB to generate the LCD CLK signal.

32.4.1.2 UDB

LCD low-power mode uses the UDB to generate various signals that control the functioning of the LCD system. These control signals are generated using the functional blocks listed below:

- Pulse generator
- BGREF timer
- Drive pulse-width modulator (PWM)
- Control register for frame data
- Mode control signals to the LCD driver

Figure 32-7. LCD UDB Logic



The pulse generator samples the interrupt signal from the LCD timer; in response, it generates one synchronous clock pulse (LCD CLK), which is routed to the BGREF timer and DMA (for frame data). This synchronous clock triggers these operations:

- Puts the sleep command issued by CPU, if any, on hold (using signal LP_ACK) until LCD refresh operation is completed.
- Enables the BGREF timer. The BGREF timer is used to provide a 2.5-µs delay, which is necessary to stabilize the bandgap reference circuit.
- Triggers the DMA to transfer the frame data into the UDB control register. Frame is a square wave signal that is used for proper sequencing of LCD refresh action. Each cycle of the frame signal represents one common update state.

After the DMA transfer for frame data and the BGREF timeout are completed, Drive PWM is enabled. The Drive PWM output “Drive” signal is routed to all the LCD driver blocks associated with the GPIO. It enables the LCD buffer to drive the LCD glass. The UDB also provides the two signals that set the drive mode of the LCD buffer.

32.4.1.3 DMA

Two DMA channels are used by the LCD component for:

- Transferring the frame information into the control register of the UDB from the system memory (RAM)
- Transferring the display information from system memory (RAM) into the port register

32.4.1.4 LCD DAC and Driver: Low Power Feature

The LCD DAC and driver have some features that are useful for LCD low-power mode functioning and help to achieve the lowest power consumption when the LCD system is shut down.

The LCD DAC can remain active when the chip is put in sleep mode. In this mode, the DAC can continue to drive the inputs of LCD drivers. To enable this mode, set the continuous_drive bit in the LCDDAC.CR0 register to 1. The

LCD DAC receives a pwrdn signal, which shuts the DAC off when it is HIGH.

The LCD driver receives a display blank signal, dispbInk, controlled by the LCDDRV.CR register. This signal sets the output to be either tristated or grounded when the chip is in low-power mode. This function works when the power down signal (pwrdn_n) signal is low. The pwrdn_n signal is used when the LCD system needs to be shut down.

The buffer present in the LCD driver can be bypassed by setting the bypass_en bit of the LCDDRV.CR register to 1.

Thus, for operation in sleep mode, for an LCD low-power system, continuous_drive, bypass_en, and pwrdn bit must be set to 1, and pwrdn_n must be set to 0. This causes the DAC to directly drive the LCD, bypassing the LCD driver section, which is shut down in chip low-power mode.

The various operating modes of the LCD DAC and LCD driver are summarized in Table 32-5 and Table 32-6 on page 360.

Table 32-5. LCD DAC Operating Modes

Chip Mode	Block Mode	pwrdn_n	continuous_drive	Description
Active	Active	1	X	LCD DAC is active. It can drive I/Os or LCD drivers depending on the LCD driver mode.
Sleep	Sleep with bypass drive	0	1	LCD DAC is active and driving I/Os even though the chip is in sleep. LCD drivers are bypassed.
Active/ Sleep/ Hibernate	OFF	0	0	LCD DAC is powered down

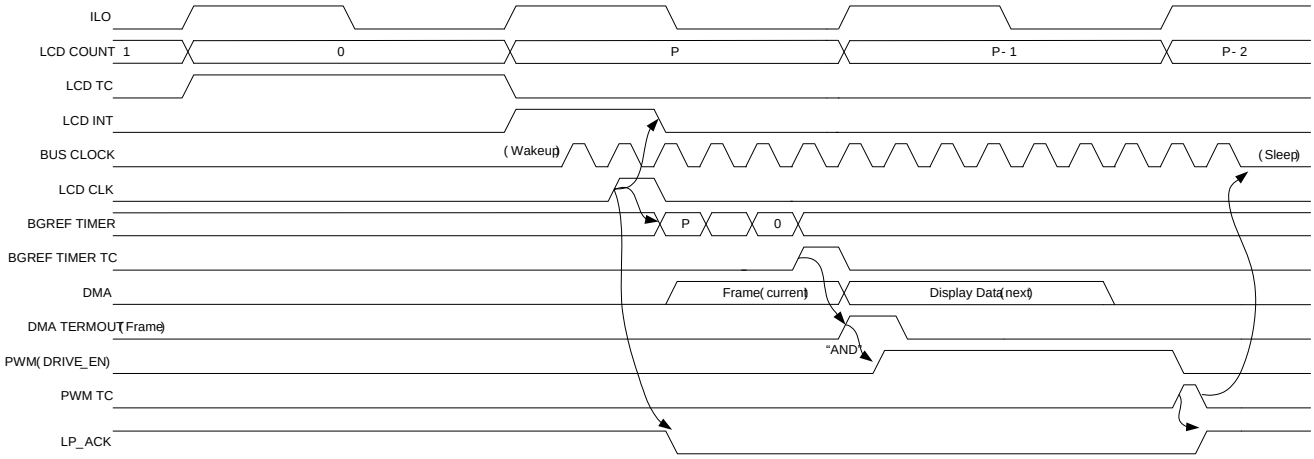
Table 32-6. LCD Driver Operating Modes

Chip Mode	Block Mode	pwrdn_n	dispbInk	Drive	bypass_en	Description
Active	Active drive	1	X	1	X	LCD driver is driving the pin in one of the High Drive or Low Drive modes.
Active	Active with bypass drive	1	X	0	1	LCD driver is bypassed. LCD DAC is driving the I/O.
Active	Active with tristate drive	1	X	0	0	LCD driver is active but the I/O is tristated.
Sleep	Off with bypass drive	0	0	X	1	LCD driver is powered down. LCD DAC is in sleep with bypass drive mode and driving the I/O.
Active/ Sleep/ Hibernate	Off with ground drive	0	1	X	X	LCD driver is powered down. Output is grounded. This is the power down mode for LCD applications. LCD DAC is off.
Active/ Sleep/ Hibernate	Off with tristate drive	0	0	X	0	LCD driver is powered down. Output is tri-stated. This is the power down mode for nonLCD applications. LCD DAC is off.

32.4.2 Timing Diagram for LCD Low-Power Mode

Figure 32-8 shows the timing in low-power mode.

Figure 32-8. LCD Low-Power Mode Timing Diagram



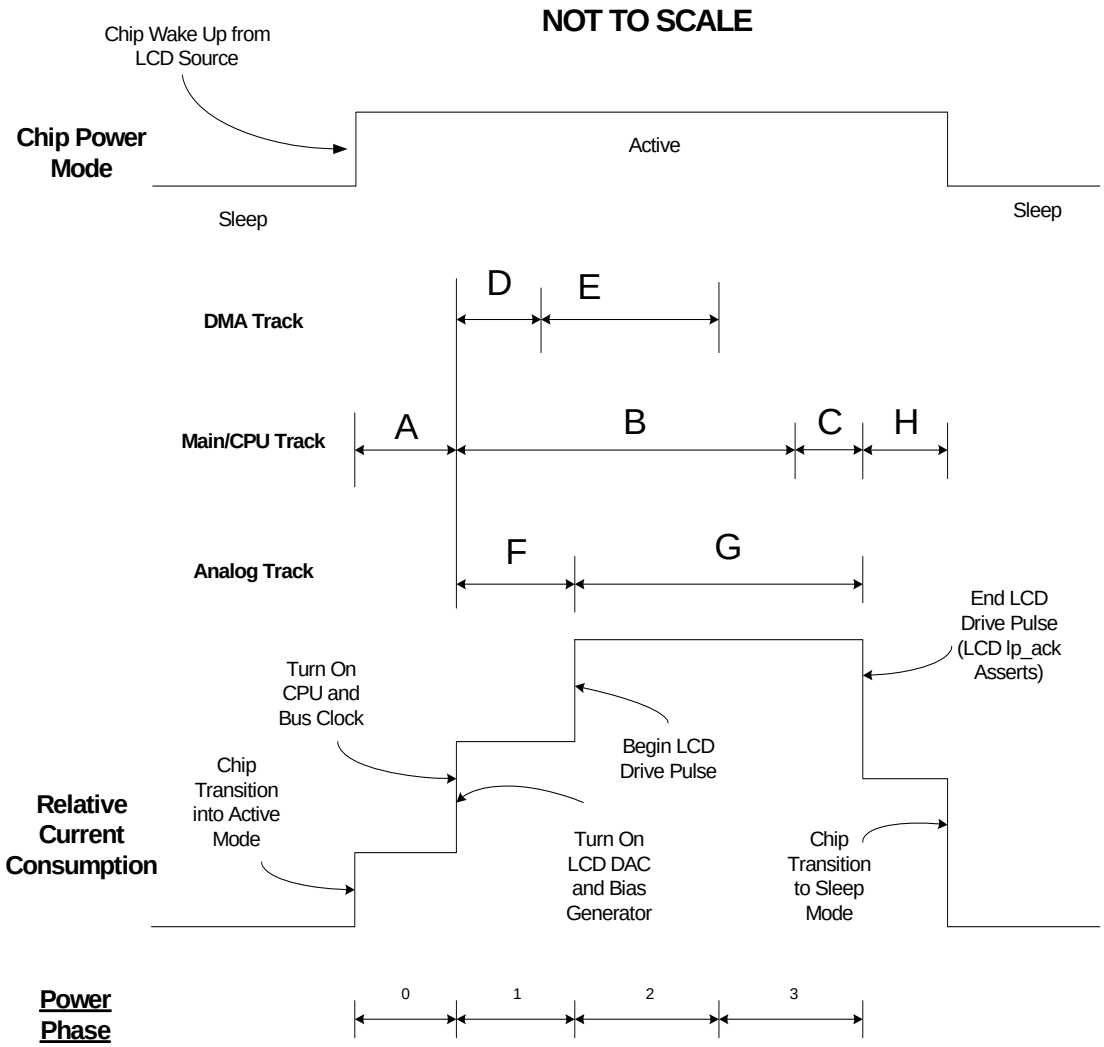
A refresh timer overflow triggers an interrupt to the PM system and also drives the UDB pulse generator logic. After a few microseconds, system clocks are restored. This puts all of the resources on the chip in operation. The UDB-implemented pulse generator outputs an LCD CLK pulse, which:

- Triggers the DMA to transfer frame information into the control register of the UDB
- Enables the BGREF timer (implemented using UDB)
- Copies the display data from the port data register into the driver for the present LCD state
- Clears the refresh rate timer interrupt
- Puts the sleep command from the CPU on hold

After the frame information transfer, another DMA is triggered to transfer the display data into the port data register for the next LCD state. When the frame data transfer is completed and the BGREF timer overflows, the LCD drive buffer is enabled using the drive signal from the Drive PWM. This is when LCD glass refresh begins. The drive mode of the LCD drive buffer determines the current drive. After the drive time is set, the drive line goes low, disabling the buffer. This also releases the sleep command hold set by the LCD CLK. This causes PM to execute the sleep command issued by the CPU. During the rest of the period, the LCD is driven continuously from the LCD DAC, bypassing the driver buffer.

Figure 32-9 on page 362 shows the sequence of operations and relative current consumption for low-power mode.

Figure 32-9. LCD Sequence of Operation



Main/CPU Track
A) Chip wake up process
B) 'Main' execution: Check for interrupts, request sleep
C) Power Manager (PM) asserts low power request (lp_req) to all subsystems and waits for all acknowledge signals (lp_ack) to assert
H) PM Completes Active -> Sleep Mode transition

DMA Track
D) DMA TD: Update FR value
(must complete before starting 'G')
E) DMA TD: Setup Display Data for next LCD refresh cycle

Analog Track
F) LCD DAC and LCD bias generator power up (2.5 µs)
(must complete before starting 'G')
G) "Drive" pulse

32.5 LCD Usage Models

The LCD can be used in these cases:

- The chip is always maintained in active mode. The LCD driver buffer will drive in high drive mode for the specified time; later on, it will switch back to low drive mode. This mode can be used when the system is always on and a power saving feature is not needed. This uses LCD always active mode.
- The chip enters low-power mode and the LCD does not need to function. Disable the entire LCD system before putting the device to low-power mode. This also uses LCD always active mode.
- The chip enters low-power mode and the LCD must be functional. In this situation, the background LCD refresh timer allows the chip to be put to sleep and awakened at regular intervals to refresh the LCD glass. This system uses LCD low-power mode. There are restrictions in refresh rates due to the low frequency clock used for the LCD timer.

  Table 32-7 shows the allowed refresh rate values for this case:

Table 32-7. Refresh Rate Limits

Commons	ILO		ECO	
	Max	Min	Max	Min
2	125	21	128	32
4	125	21	128	32
8	63	21	128	20
16	31	-	128	20

33. CapSense

PSoC® 3 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.

33.1 Features

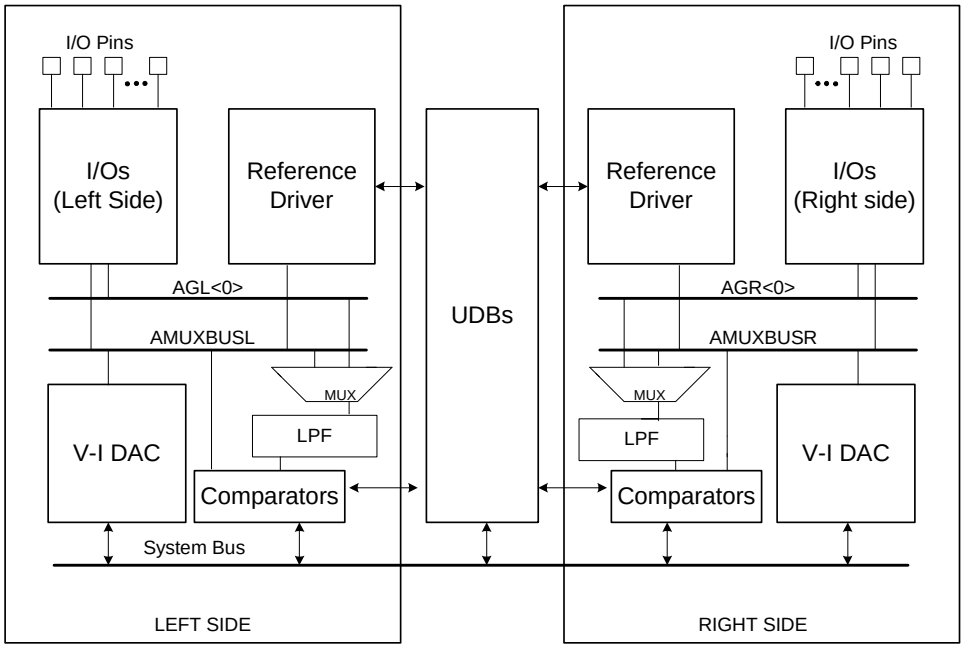
Features of CapSense include:

- 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

33.2 Block Diagram

Figure 33-1 shows a block diagram of the overall capacitive sensing architecture.

Figure 33-1. CapSense Module Block Diagram



33.3 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.

33.3.1 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.

33.3.2 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.

33.3.3 Analog Mux Bus

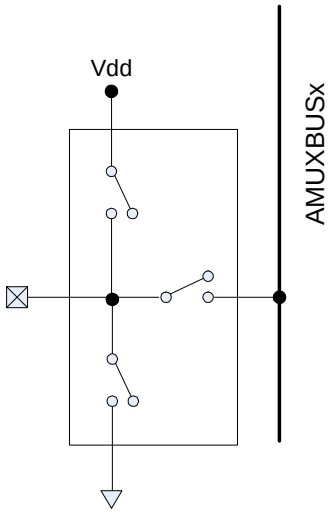
All GPIO pins support CapSense operations except SIO and USB pins. The primary analog mux bus for CapSense is the AMUXBUS, which has two nets (AMUXBUSL and AMUXBUSR) for two simultaneous sensing operations. These can also be shorted to form a single net that connects to all GPIO. See the device datasheet for details about GPIOs available in each package and the Analog Routing chapter on page 327 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.

33.3.4 GPIO Configuration for CapSense

The GPIO switching structure supporting CapSense is shown in Figure 33-2.

Figure 33-2. 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 33-3 on page 367 and Figure 33-4 on page 367, 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 33-3. Charging MUXBUS Through Sense Resistor

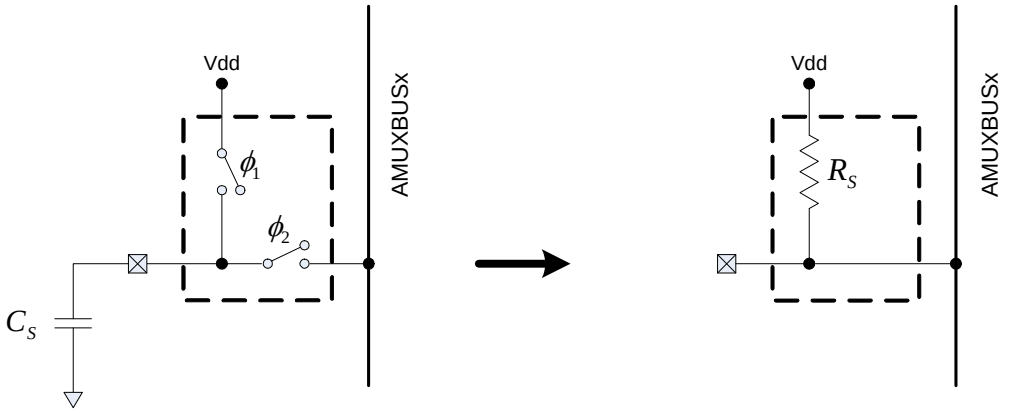
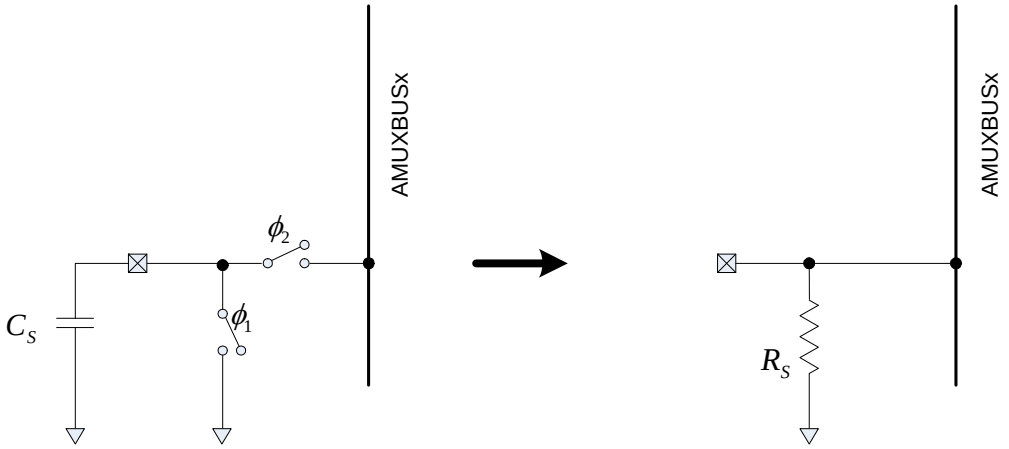


Figure 33-4. Discharging the MUXBUS Through Sense Resistor



The CapSense clock is used for switching. Two alternatives are available to generate the CapSense clock (see also Figure 19-1 on page 160).

- 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.

33.3.5 Other Resources

CSD CapSense techniques use many resources in PSoC 3 devices. These include UDBs, Comparators, and V-I DAC. See the Universal Digital Blocks (UDBs) chapter on page 183, Comparators chapter on page 343, and Digital-to-Analog Converter chapter on page 375 for more details.

33.4 CapSense Delta Sigma Algorithm

The CapSense Delta Sigma (CSD) algorithm shown in Figure 33-5 and Figure 33-6 on page 369 measures capacitance with the hardware configured like 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 Vdd 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 33-5. CSD Hardware Configuration

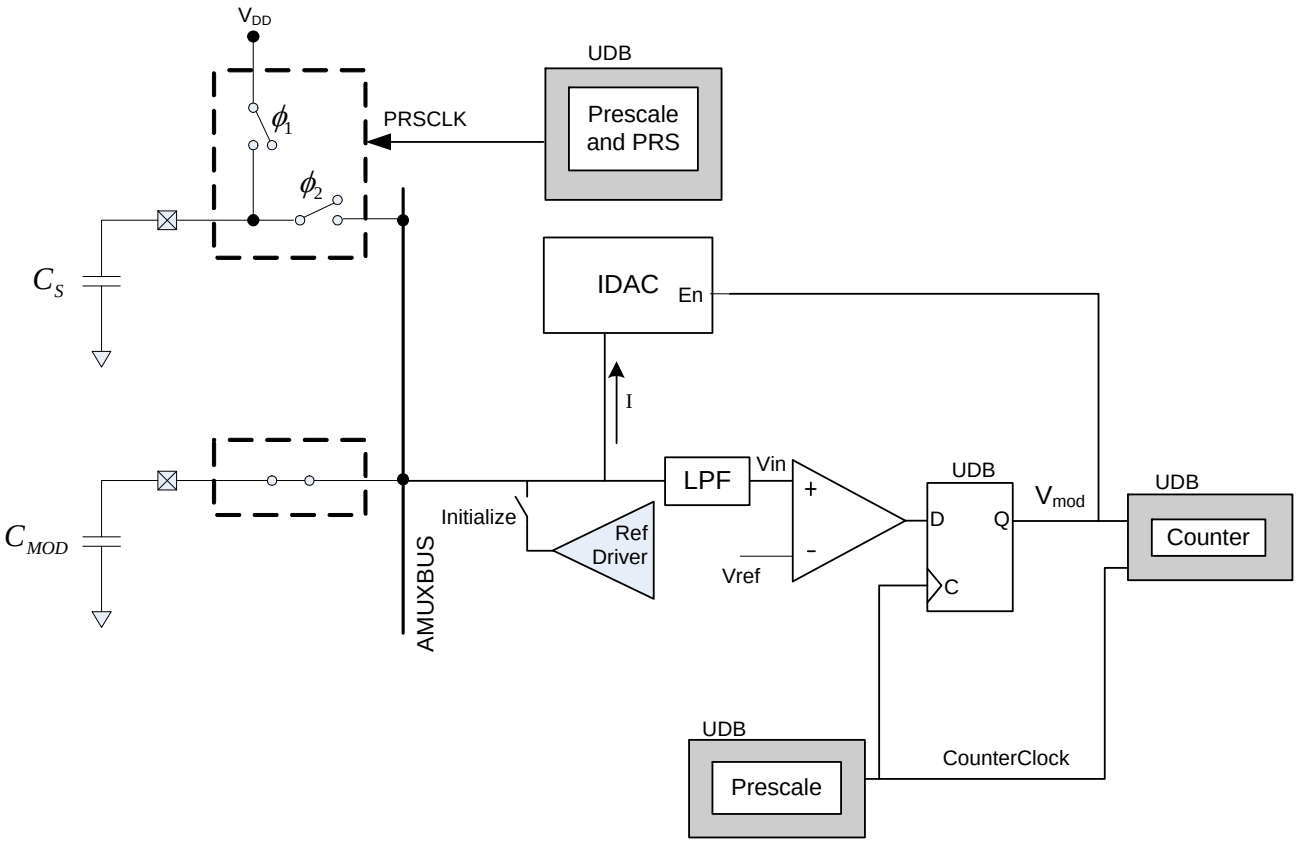
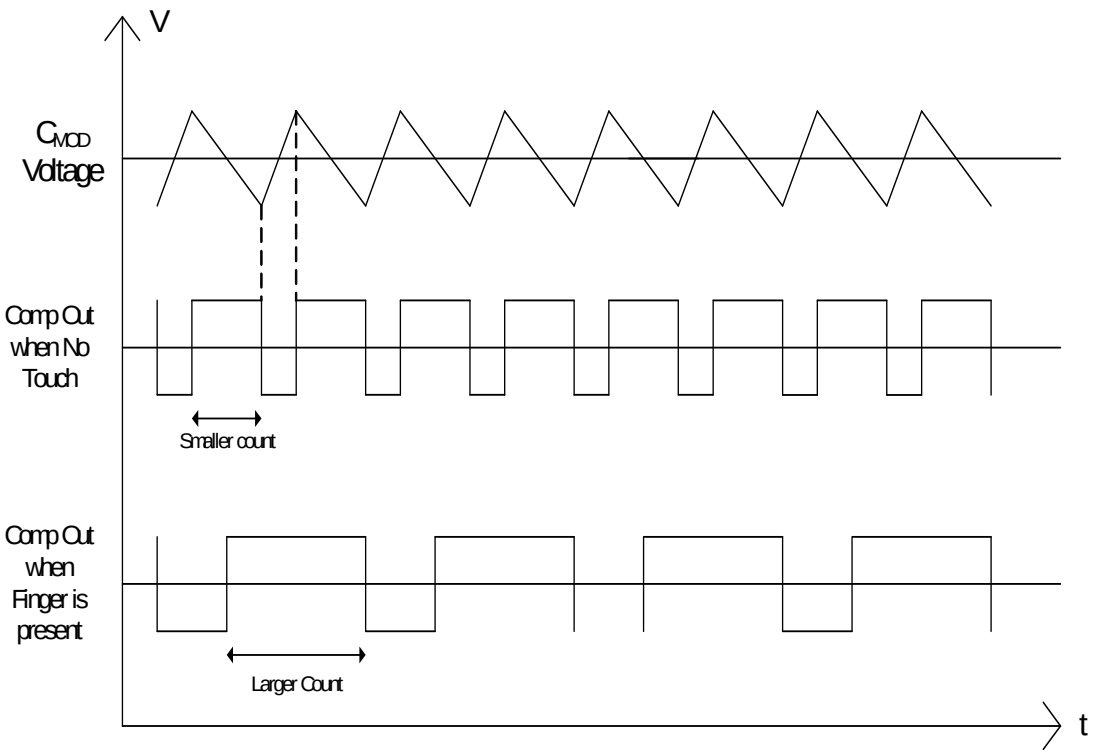


Figure 33-6. CSD Waveform



The PSoC device also supports other variants of the CSD algorithm as follows:

- Switched capacitor resistor (see Figure 33-3 on page 367) 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 33-4 on page 367) discharges the integration capacitor toward ground, based on comparator output.

34. Temperature Sensor

The PSoC® 3 devices have an on-chip temperature sensor that is used to measure the internal die temperature. The temperature sensor uses the Delta Vbe method for digital temperature measurement.

The temperature sensor block has an auxiliary analog-to-digital converter (ADC) to measure the internal die temperature. The auxiliary ADC is a 10-bit accurate ADC in the system performance controller (SPC) primarily designed for measuring temperature sensor output. It is also possible to route the analog output of diode in temperature sensor block to analog globals to measure temperature using the higher resolution Delta-Sigma ADC in PSoC 3.

34.1 Features

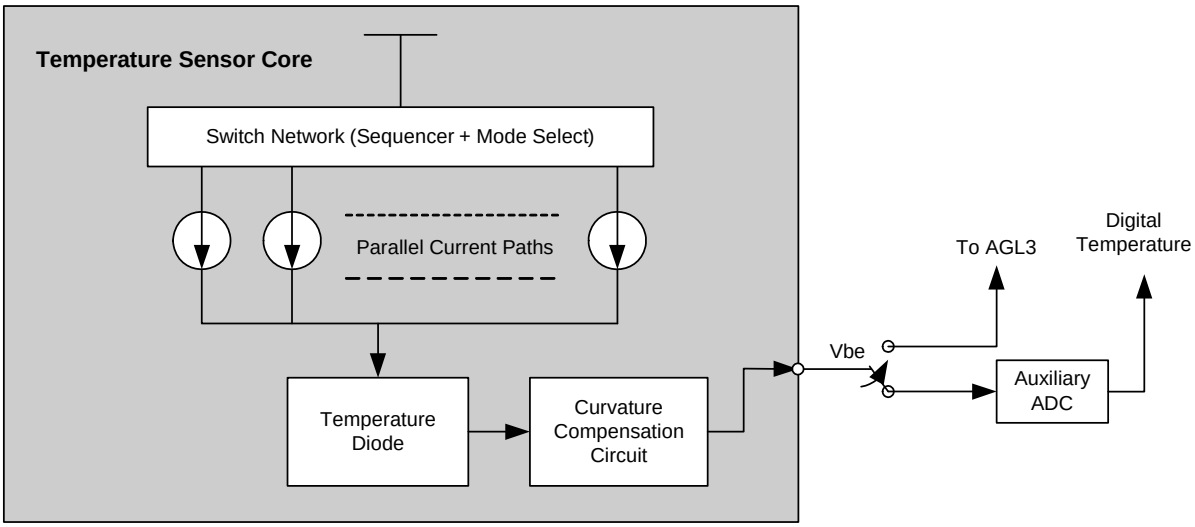
The temperature sensor offers the following features:

- ± 5 degrees Celsius accuracy over commercial temperature range (–50ºC to +150ºC)
- Ability to route temperature sensor output to analog global line, AGL3.

34.2 Block Diagram

The block diagram for the temperature sensor is illustrated in Figure 34-1.

Figure 34-1. Temperature Sensor



34.3 How It Works

The base-to-emitter voltage of a Bipolar Junction Transistor (BJT) device has a strong dependence on temperature at a constant collector current and zero collector-base voltage. The temperature sensor output (Vbe) is measured with two different drive currents: first with low bias current and second with high bias current. A current ratio of 1:29 is maintained between the conversions.

By making the ratio between the two drive currents high, the voltage difference between the Vbe values is linearly proportional to temperature. The output voltage of the temperature sensor is either driven to the Delta Sigma ADC or other on-chip resources using analog global line (AGL3). To increase accuracy, the PSoC 3 temperature sensors use the following techniques:

- Dynamic Element Matching technique is implemented using a sequencer that cyclically selects among the eight current mirror paths during conversion (low current mode and high current mode).
- Curvature compensation circuit to increase linearity when the temperature sensor output is routed to an external resource with a High Z buffer such as the on-chip Delta Sigma ADC.
- A two-point linear fit calibration routine for accurate temperature measurements using the auxiliary ADC.

34.4 Command and Status Interface

The commands associated with the temperature sensor are executed through the simple command/status register interface. “Get Temp,” “Setup Temperature Sensor,” and “Disable Temperature Sensor” are commands associated with the temperature sensor. The command is 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 status register, SPC_SR, indicates whether a new command can be accepted, when data is available for the most recent command, and success/failure response (status code) for the most recent command.

Table 34-1. Command Registers

Register	Size (Bits)	Description
SPC_CPU_DATA	8	Data to or from CPU
SPC_DMA_DATA	8	Data to or from DMAC
SPC_SR	8	Status – ready, data available, status code

The command sequence consists of a 2-byte key, followed by command code and the parameters associated with the command.

- Key byte #1 – always 0xB6
- Key byte #2 – 0xD3 plus the command code (ignore overflow)
- Command code byte
- Command parameter bytes
- Command data bytes

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 then 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.

34.4.1 Status Codes

If the value of the 2-byte key is wrong or if any of the parameters passed are invalid, the command is ignored and the error condition is indicated by the status code in the Status register (SPC_SR). The Status_Code bits (7:2 in the Status register) are used to determine if the command operation is executed successfully or any error occurred. Table 34-2 lists the status code bit values.

Table 34-2. Status Code Bit Values

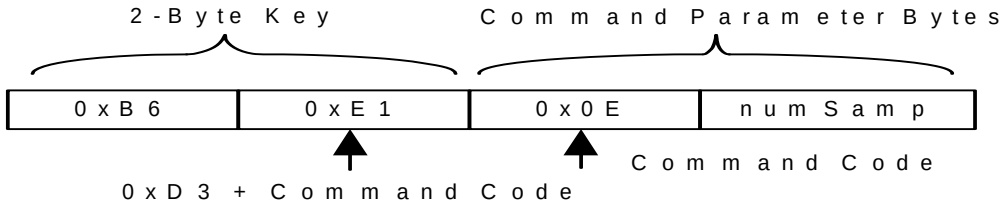
Status_Code Bit Values (Bits[7:2] in SPC_SR register)	Description
0x00	Command successfully executed
0x02	Invalid key
0x0B	Invalid command code
0x0D	Invalid parameter
0x0E	Temperature Sensor Vbe is currently driven to an external device

34.4.2 Temperature Sensor Commands

34.4.2.1 Get Temperature

“Get Temperature” (command code: 0x0E). This command uses auxiliary ADC to measure the die temperature and the ADC output. It returns 2 bytes corresponding to a temperature value. The first byte is the sign of the temperature (0 = negative, 1 = positive). The second byte is the magnitude. These values are read from the SPC Data register. The command sequence is shown in Figure 34-2.

Figure 34-2. Get Temperature Command Sequence



Command Parameters

numSamp

This parameter specifies the number of samples taken. The number of samples is equal to 2^numSamp. Valid values for this parameter are 0, 1, 2, 3, 4, or 5, thereby resulting in 1, 2, 4, 8, 16, or 32 samples, respectively. The ADC output is read after the averaging is done over all the samples as specified by this parameter. The averaging routine can be bypassed by selecting the numSamp value as 0.

Reading Temperature Output

After the command and its parameters are sent, the Temperature Sensor/ADC block is configured and starts the conversion.

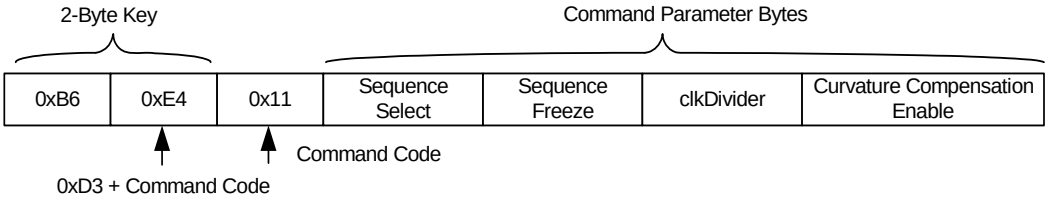
When the conversion is complete, the DATA READY bit in the Status register (SPC_SR) is set. The CPU must poll this bit to check if the ADC output is ready. When the bit is high, the first byte (Sign byte) of output is read from the Data register (SPC_CPU_DATA). The DATA READY bit is reset when a read operation is done. When the second byte (Magnitude byte) is ready to read, the DATA READY bit becomes high again and the second byte is read from the Data register (SPC_CPU_DATA).

34.4.2.2 Setup Temperature Sensor

“Setup Temperature Sensor” (command code: 0x11). The purpose of this command is to connect the raw temperature sensor analog output onto AGL3 for measurement by the High Z buffer/Delta Sigma ADC(DSM) or other external resources. The auxiliary ADC cannot be operated at the same time when the sensor output is routed to AGL3. This command disables the functionality of the auxiliary ADC such that it does not load the sensor when the sensor output voltage is being driven into the DSM or other external ADCs. The “Setup Temperature Sensor” and “Disable Temperature Sensor” are the commands associated with this purpose and drive the temperature sensor output to AGL3. When temperature sensor output is routed to an analog global line, auxiliary ADC cannot be used to measure the temperature.

Note that AGL3 should not be used by analog blocks other than the temperature sensor output when this command is executed. Even though PSoC Creator takes care of routing, ensure that there are no resource conflicts in using AGL3. The command sequence is shown in Figure 34-3.

Figure 34-3. Setup Temperature Command Sequence



Command Parameters

Sequence Select. The temperature sensor output (Vbe) voltage is measured with low bias current and then with high bias current. A current ratio of 1:29 is established between the low bias and high bias current. This ratio is fixed and not configurable.The difference between the two output voltages is linearly proportional to temperature.

- 0 – Low bias current.
  The temperature sensor is driven with low bias current.
- 1 – High bias current.
  The temperature sensor is driven with high bias current.

Sequence Freeze. In low bias and high bias current modes, Dynamic Element Matching (DEM) is implemented by a sequencer that cyclically selects among the eight current mirror paths.

- 0 – Sequencer is enabled.
- 1 – Sequencer is disabled.
  No cycling of the current paths occurs.

clkDivider. This parameter sets the divider value for clock generation from the SPC clock (spcCLK, which is 36 MHz). This clock is used by the sequencer to cycle through the current mirrors. The clock frequency is equal to:

$$\frac{spcCLK}{\langle clkDivider + 1 \rangle}$$ **Equation 1**

The clock divider value (clkDivider) is of 8 bits allowing clock to have 256 different frequencies ranging from spcCLK down to spcCLK/256 (spcCLK is 36 MHz). In general, the slower the clock, the better the linearity that will be achieved.

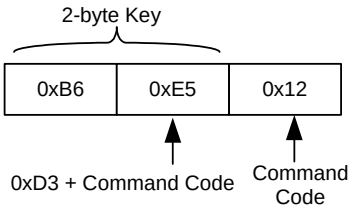
Curvature Compensation Enable. The temperature sensor has a feature to correct for a curvature in its behavior and align it to a more linear path, thus giving it more accuracy when its output is routed to an external resource with a High Z buffer, such as the on-chip Delta Sigma ADC. A High Z buffer is required because the curvature compensation circuit needs to be buffered before driving an external ADC front end.

- 0 – No curvature compensation is used.
- 1 – Curvature compensation is enabled.

34.4.2.3 Disable Temperature Sensor

"Disable Temperature Sensor" (Command code: 0x12). This command is used to disable the temperature sensor from driving its output voltage to the analog global line (AGL3). After calling this command, the "Get Temp" command can be executed, as well as commands using the erase portion of the Smart Write algorithm. This command has no parameters and does not return any value. The command sequence is shown in Figure 34-4.

Figure 34-4. Disable Temperature Command Sequence



35. Digital-to-Analog Converter

The 8-bit digital-to-analog converter (DAC) is configured to output either a voltage or a current. The 8-bit DAC supports CapSense®, power supply regulation, and waveform generation.

35.1 Features

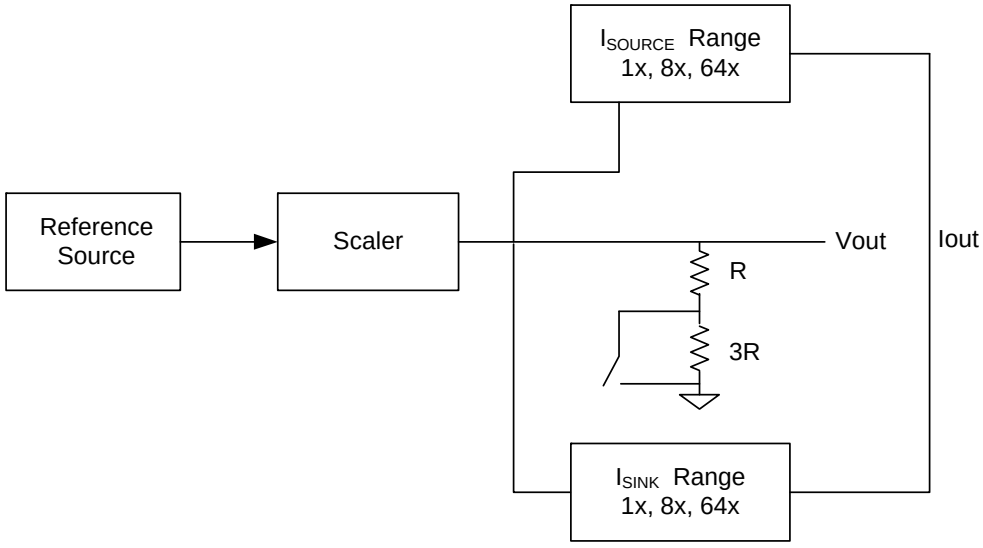
The DAC has the following features:

- 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: 8 Msps
- Output rate for VDAC voltage output: 1 Msps
- Monotonic in nature

35.2 Block Diagram

A block diagram of the DAC is shown in Figure 35-1.

Figure 35-1. DAC Block Diagram



35.3 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, analog clock strobe to any UDB signal strobe. This selection is done on the basis of setting in DACx_STROBE[2:0].

- **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.

35.3.1 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. You also have 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.

35.3.2 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.

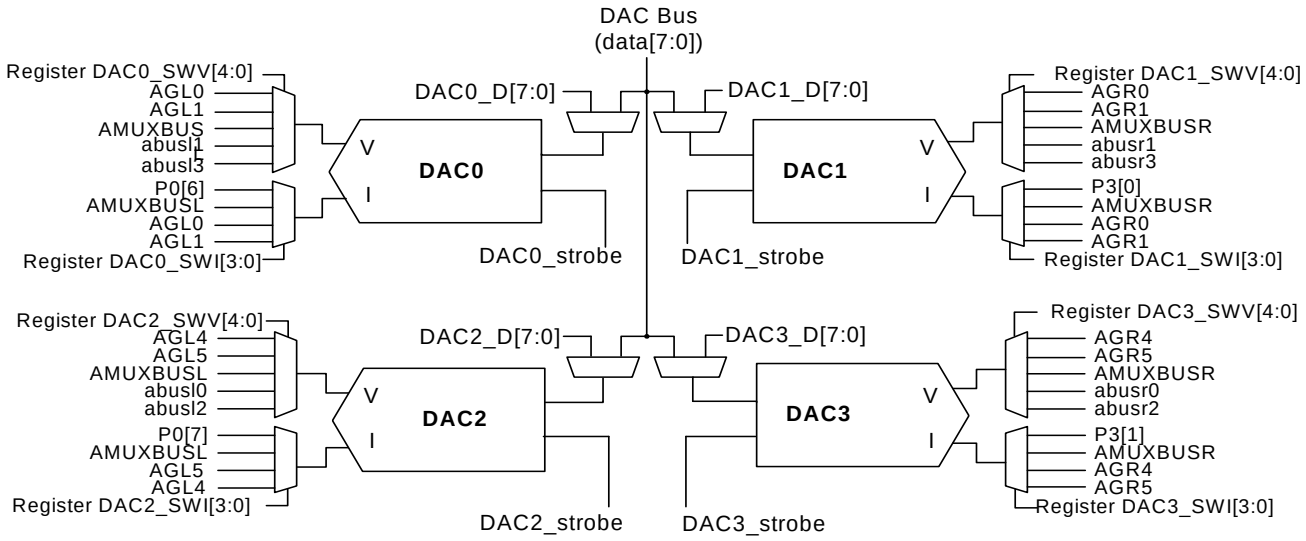
The VDAC is implemented by driving the output of the current DAC through resistors and obtaining a voltage output. Because no buffer is used, any DC current drawn from the DAC affects the output level. Therefore, in this mode any load connected to the output should be capacitive.

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 Vssa is four times larger. In 4 V mode, the VDAC is capable of converting up to 250 ksps.

35.3.3 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 35-2 on page 377.

Figure 35-2. DAC Interconnect



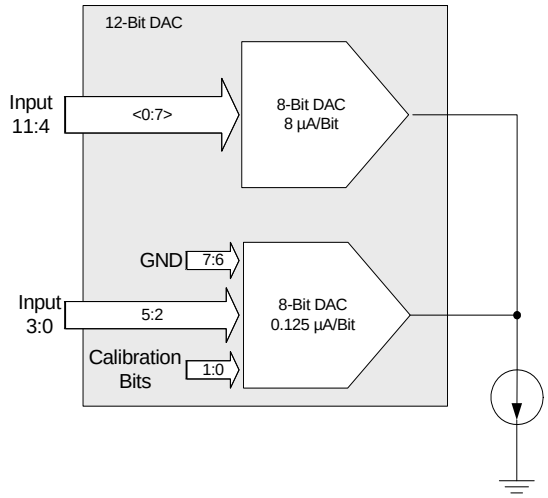
You can route output as follows:

- **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

35.3.4 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 35-3. Higher Resolution DAC Example

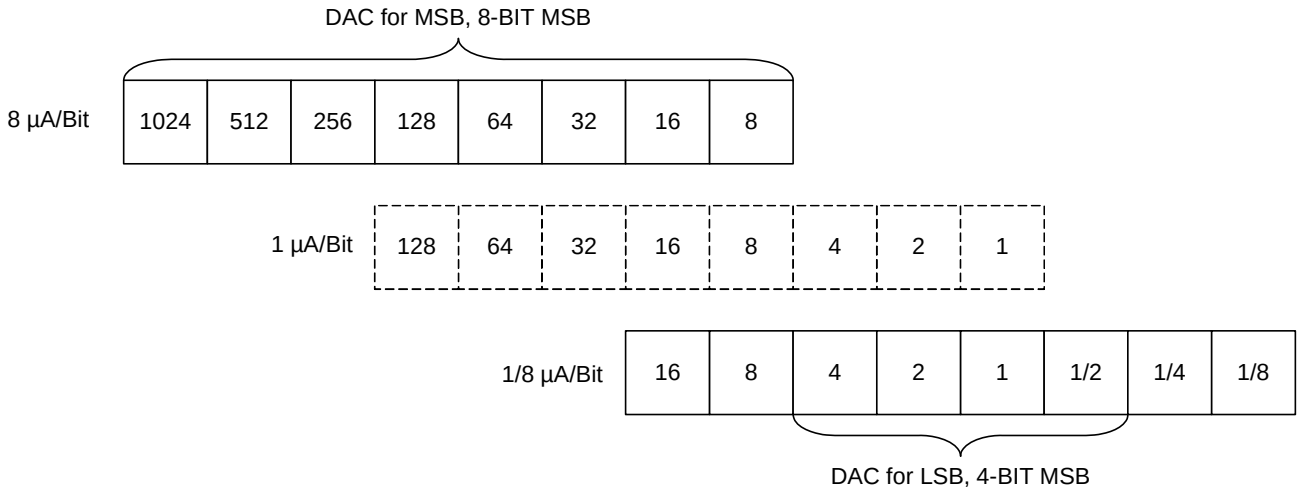


For example, the implementation of a 12-bit DAC using two 8-bit DACs require:

- 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 four bits of the lowest range DAC are used as inputs to the lower four bits. See Figure 35-4 on page 378.

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 35-4. 12-Bit DAC Using Two 8-Bit DACs Example



35.4 Register List

Table 35-1. 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

36. 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 ADC, a reference voltage is required to quantify an input, while in V/I DAC, voltage/current reference is required to define the output full-scale range.

36.1 Block Diagram

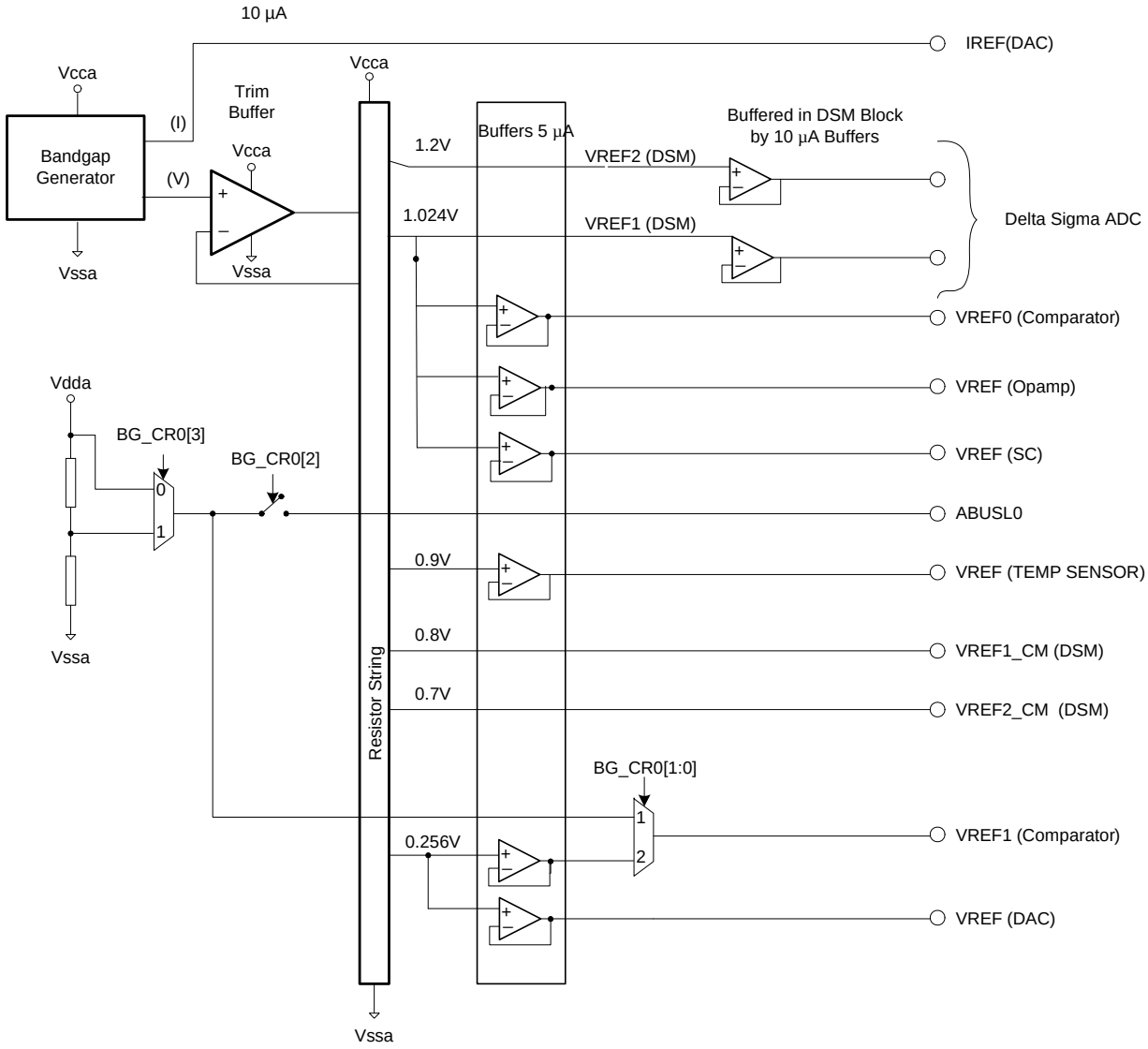
The PSoC® 3 devices have a curvature compensated voltage bandgap along with a trim buffer to get absolute value accuracy. The trim buffer is a multiple reference generator. It takes the bandgap reference voltage as input and produces outputs ranging from 0.256 V to 1.2 V. The reference voltage is buffered by low-power 5 µA, high accuracy buffers, and sent to multiple destinations. There is also a temperature corrected (to flat) current reference that is mirrored and sent to current DAC.

The voltage reference block diagram is illustrated in Figure 36-1 on page 380.

36.2 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 PTAT (proportional-to-absolute-temperature) circuit (which includes the other group of transistors), a fixed DC voltage that does not change with temperature is generated.

Figure 36-1. 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.

Table 36-1. 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. Delta Sigma Converter

The PSoC® 3 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.

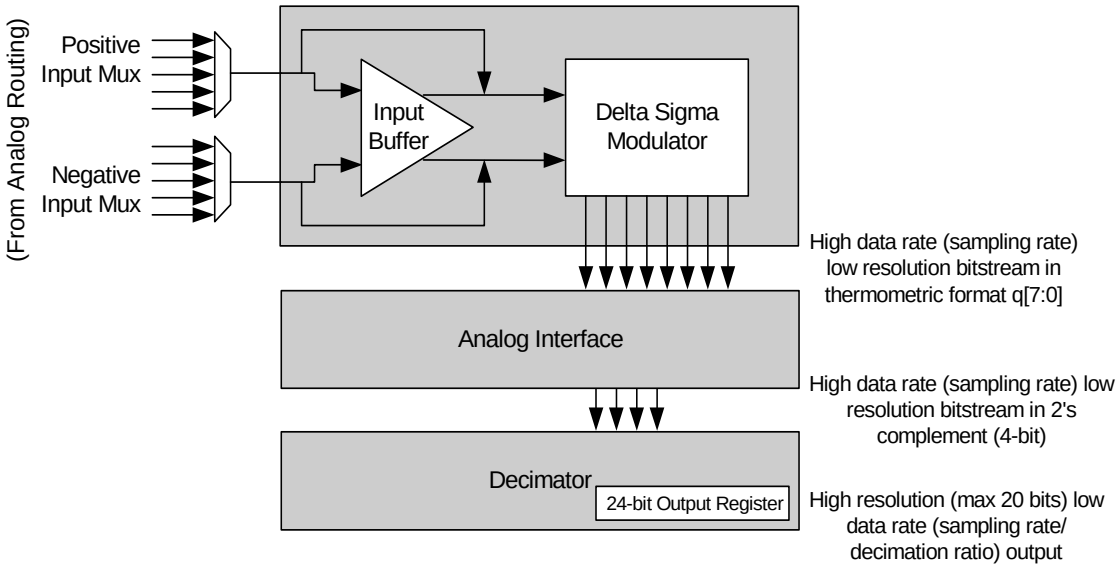
37.1 Features

- 8 - to 20-bit resolution
- Configurable gain from 0.25 to 256
- 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

37.2 Block Diagram

Figure 37-1 is the converter block diagram.

Figure 37-1. Delta Sigma Block Diagram



37.3 How It Works

The PSoC 3 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 37.3.2.6 Quantizer on page 392).
- 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.

37.3.1 Input Buffer

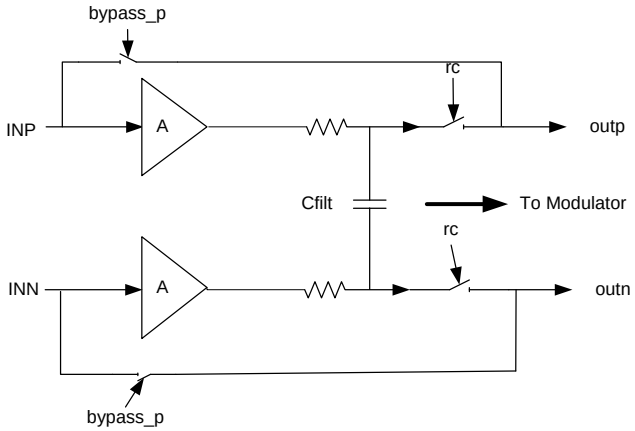
The input impedance of the modulator is very low and not suitable for many applications. For applications that require a higher input impedance, two buffers (one for each differential input) are provided. Figure 37-2 shows the buffer and the RC filter that follows it.

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 0 – vdda- 600 mV.
- **Rail-to-Rail** – This is used when input is rail-to-rail. The operating voltage range is vssa+200 mV to vdda-200 mV.

The input structure is illustrated in Figure 37-2.

Figure 37-2. 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 37-1 for a proper operation. The values in Table 37-1 are for a 1.024 V reference. For other reference values, scale the maximum input amplitude accordingly.

Table 37-1. 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

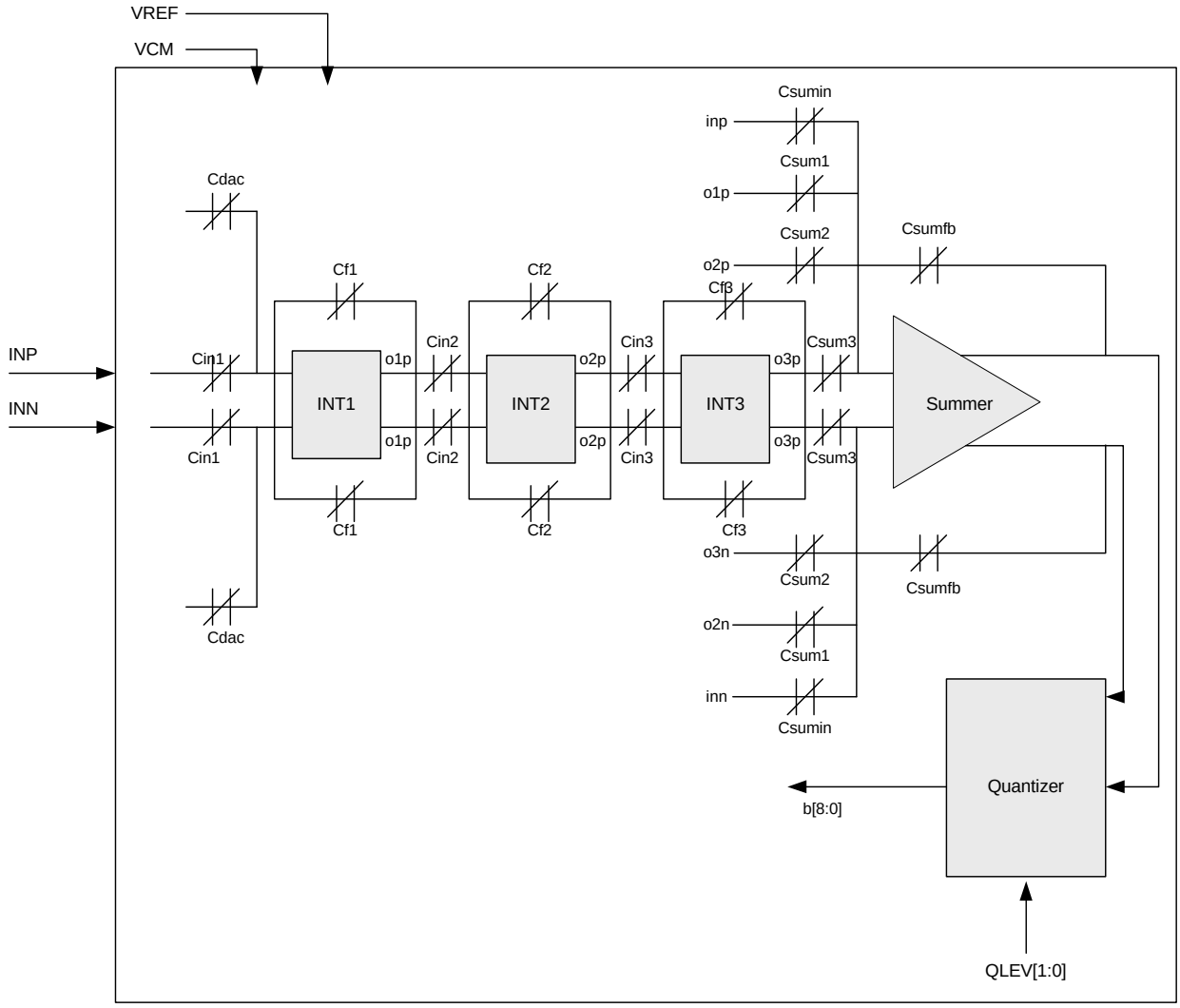
37.3.2 Delta Sigma Modulator

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 3 Delta Sigma modulator implementation is shown in Figure 37-3.

Figure 37-3. 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.

37.3.2.1 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 = fs/n, n = 2,3,4..., fs is the PHUB clock.

37.3.2.2 Capacitance Configuration

All of the capacitors shown in Figure 37-3 on page 385 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 37-2.

Table 37-2. 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	

Table 37-2. Capacitance Values *(continued)*

Register Bit	Description	Value	Default	Typical Value
IPCAP3[2:0]	Third stage input cap	C_{unit} = 50 fF	101	0.25 pF
IPCAP3EN	Enable for LSB Cap of IPCAP3	50-400 fF in 50 fF steps	0	
SUMCAPIN[4:0]	Summer cap for input path	C_{unit} = 50 fF	00101	0.25 pF
SUMCAPINEN	Enable for LSB Cap of SUMCAPIN	50-1.6 pF in 50 fF steps	0	
SUMCAPFB[3:0]	Summer cap for feedback path	C_{unit} = 50 fF	1010	0.5 pF
SUMCAPFBEN	Enable for LSB Cap of SUMCAPFB	50-800 fF in 50 fF steps	0	
SUMCAP1[2:0]	Summer cap for first stage output	C_{unit} = 50 fF	101	0.25 pF
SUMCAP1EN	Enable for LSB Cap of SUMCAP1	50-400 fF in 50 fF steps	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	

37.3.2.3 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 37-3, and those for 16, 0.25, and 0.5 are shown in Table 37-4.

Table 37-3. 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
SUMCAP1EN	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 37-4. 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	
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	

37.3.2.4 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 37-5.

Table 37-5. 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 37-5 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 tables below show a set of operational modes that indicate how to configure power based upon the other parameters or vice versa.

Table 37-6. 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 37-7. 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 37-8. 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

37.3.2.5 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 151.

37.3.2.6 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 37-9. 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

37.3.2.7 Reference Options

The Delta Sigma channel has selectable analog reference input (REFBUF0) options, as shown in Figure 37-4 on page 393. 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 37-5 on page 393)
- Internal Reference that is buffered and filtered with an external capacitor tied between P0[3] and ground or P3[2] and ground (Figure 37-6 on page 394)
- External Reference source driving reference into the DSM (Figure 37-7 on page 394)

Figure 37-4. Delta Sigma Channel Analog Reference Selection

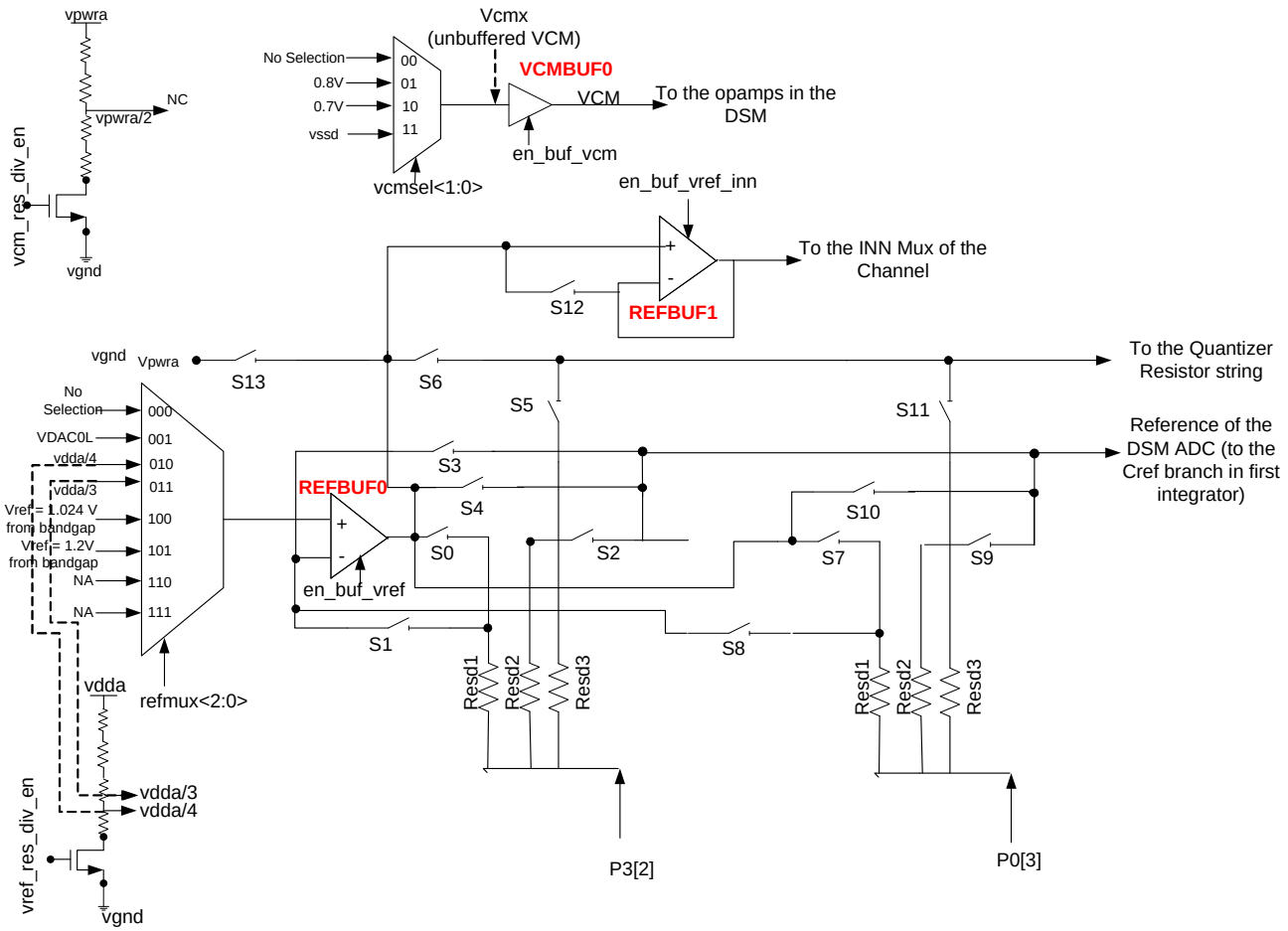


Figure 37-5. Connection Scenario: Internal Reference with No RC Filtering (using P0[3])

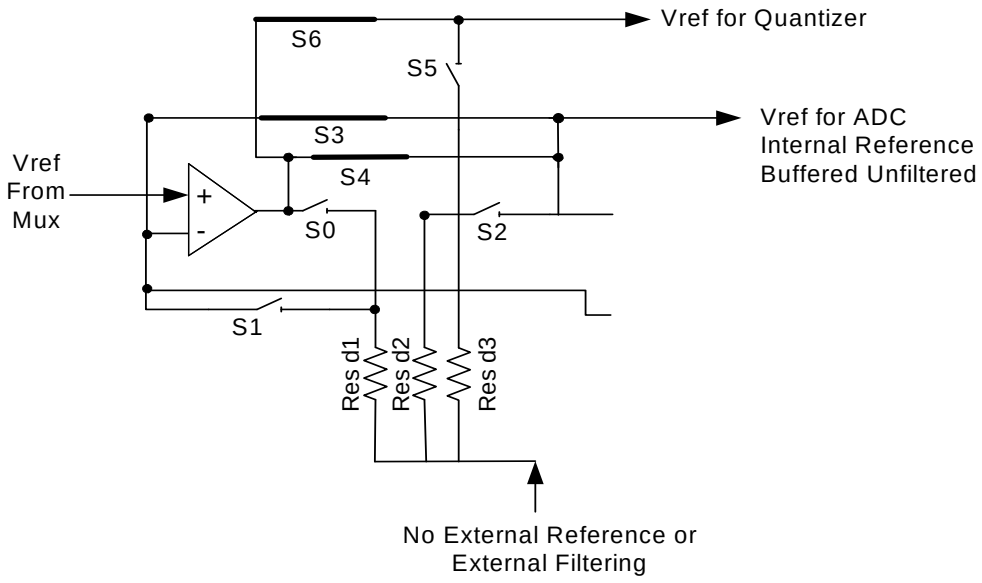


Figure 37-6. Connection Scenario: Internal Reference with RC Filtering

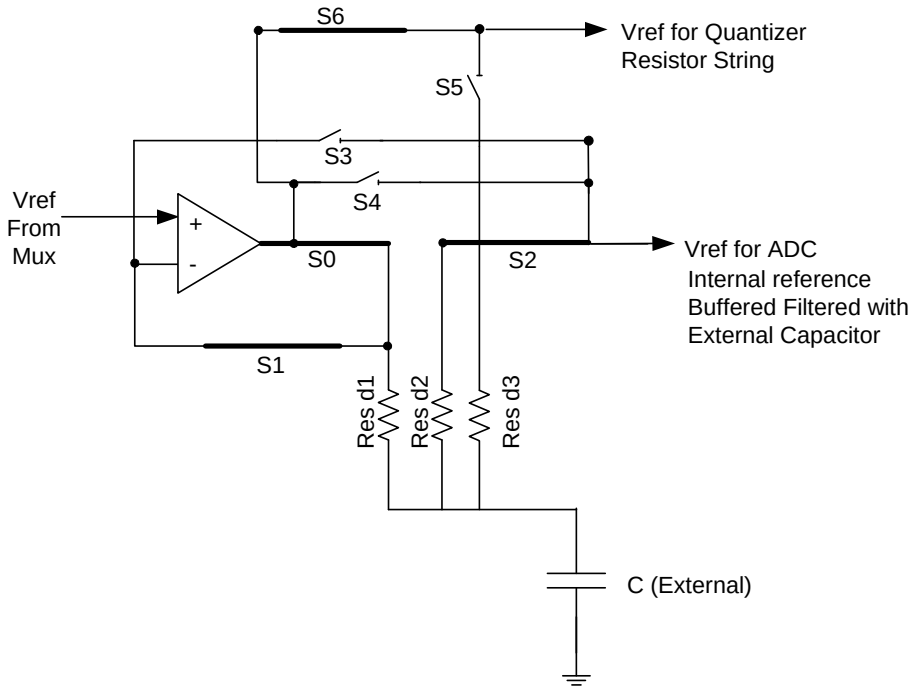
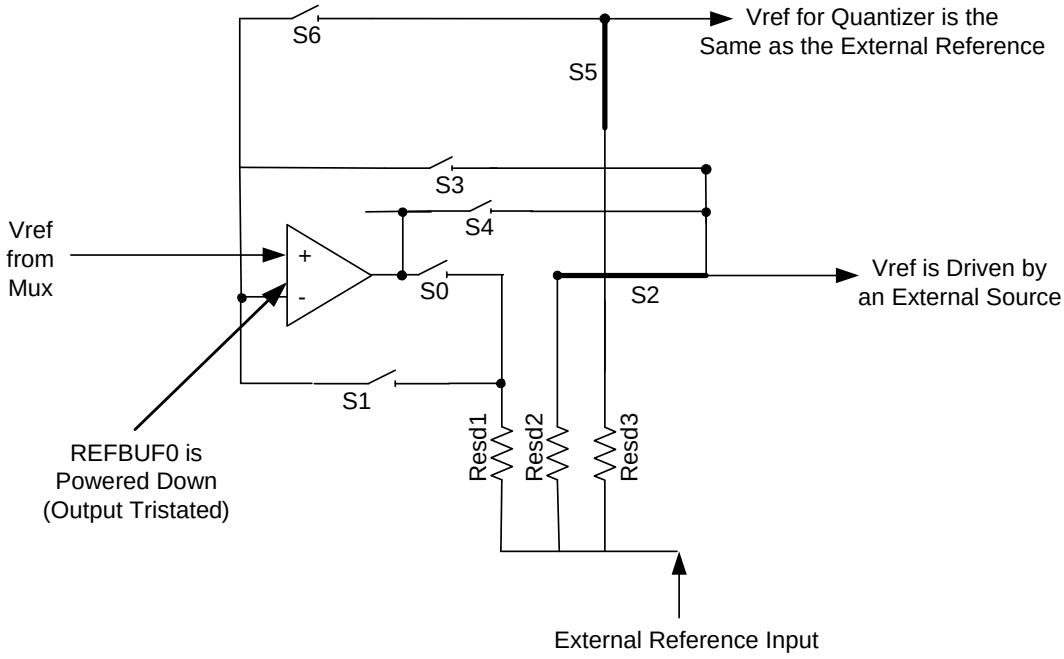


Figure 37-7. Connection Scenario: External Reference Only



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 37-3 on page 385) 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.

37.3.2.8 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 37-10).

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 37-4 on page 393 and Figure 37-7 on page 394. 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 3 devices. The switches in Table 37-10 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 37-10. 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

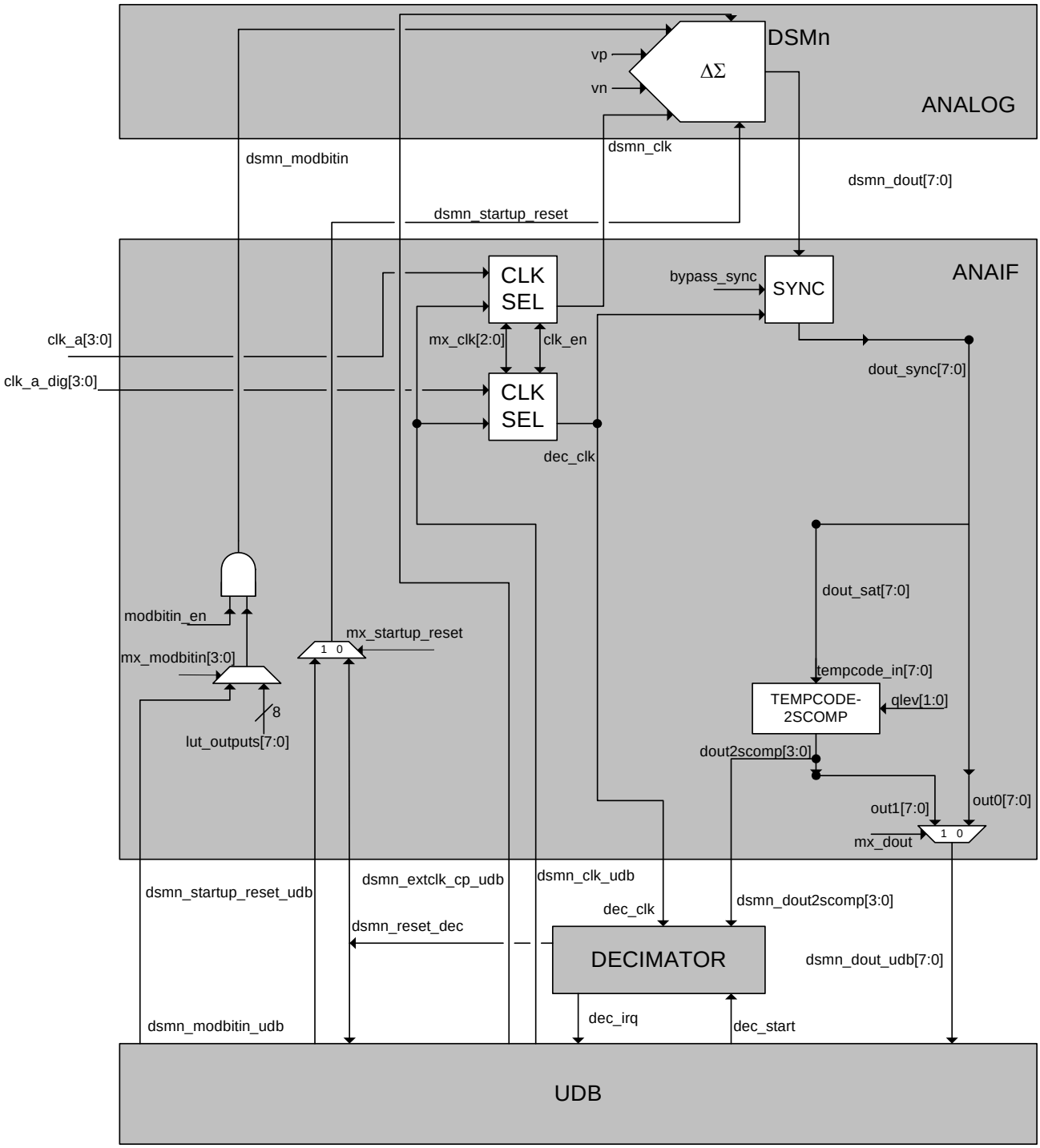
Table 37-11. Guidelines for External Reference Bypass Capacitors

Resolution	Bypass Capacitor Value
less than 16-bit	0.1 µF
16 bit or more	0.1 µF to 10 µF

37.3.3 Analog Interface

The analog interface connects the modulator to the other blocks including the decimator and the UDB. As shown in Figure 37-8 on page 396, the analog interface converts thermometric code sent by the modulator to two's complement and allows for selection of modulation input, selecting and synchronizing clocks.

Figure 37-8. Analog Interface



37.3.3.1 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.

Table 37-12. 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

37.3.3.2 Modulation Input

As discussed in 37.3.2.5 Other Configuration Options on page 392, 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.

37.3.3.3 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 37.3.2 Delta Sigma Modulator on page 385, clock selection is done by DSM_CLK[2:0] register bits. Clock synchronization is enabled by clearing the DSM_CLK[4] register bit.

37.3.4 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 3 devices is a fourth order Cascaded Integrator Comb (CIC) filter. The decimator structure is shown in Figure 37-9 on page 398.

37.3.4.1 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 37-13 and Table 37-14 on page 398.

Table 37-13. 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	4095 to -4096	12	Left shift 20
8	9	16383 to -16384	14	Left shift 18
16	2,3	65535 to -65536	16	Left shift 16
16	9	262143 to -262144	18	Left shift 14
32	2, 3	1048575 to -1048576	20	Left shift 12
32	9	4194303 to -4194304	22	Left shift 10
64	2, 3	16777215 to -16777216	24	Left shift 8
64	9	67108863 to -67108864	26	Left shift 6
128	2, 3	268435455 to -268435456	28	Left shift 4
128	9	173741823 to -1073741824	30	Left shift 2

Table 37-14. 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

37.3.4.2 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 37-15.

Table 37-15. 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 37-16.

Table 37-16. 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)

37.3.4.3 Post Processing Filter

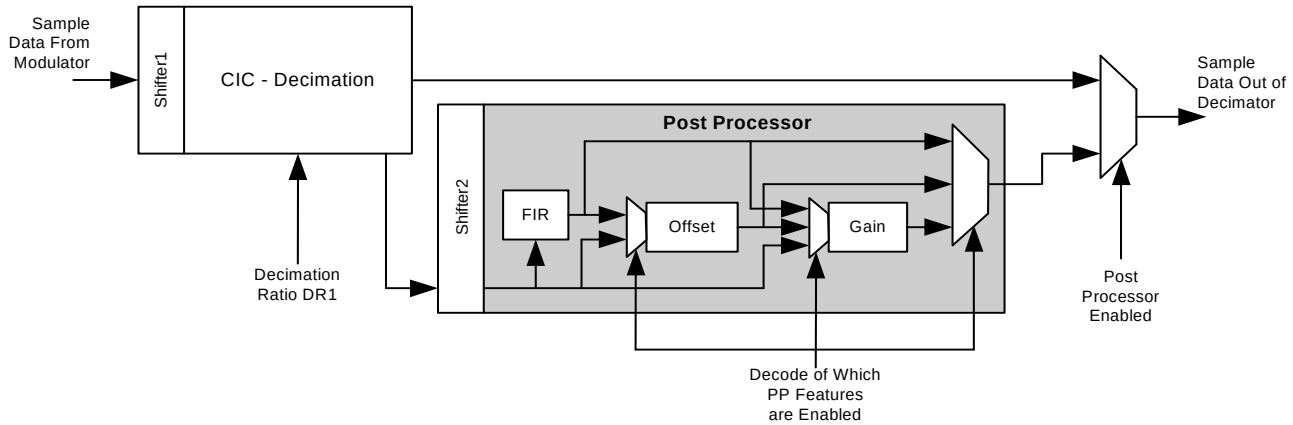
The Post Processor receives 28-bit data from the output of the CIC Decimation filter for further convenience or post processing. Available functions are:

- 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 37-9.

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 37.3.6 Coherency Protection on page 399).

Figure 37-9. Decimator



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 desired. 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.

37.3.5 ADC Conversion Time

The conversion time is the time taken from the SOC to the ADC interrupt. The conversion time provided in the table is expressed in ADC_clk cycles. Because the SOC is an asynchronous signal to ADC_clk, there can be a skew error on all the conversion times.

Table 37-17. ADC Conversion Time

ADC Mode	Conversion Time in ADC_clk Cycles
Single Sample, Fast Filter, Continuous	(DR1 x 4)+3
Fast FIR	DR1 x (DR2 +3) +6
Single Sample, Fast Filter, Continuous with offset correction	(DR1 x 4)+6
Fast FIR with offset correction	DR1 x (DR2 +3) +7
Single Sample, Fast Filter, Continuous with offset correction	(DR1 x 4)+GVAL+5
Fast FIR with offset correction	DR1 x (DR2 +3) +GVAL+7

37.3.6 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.

37.3.6.1 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.

37.3.6.2 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.

37.3.7 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.

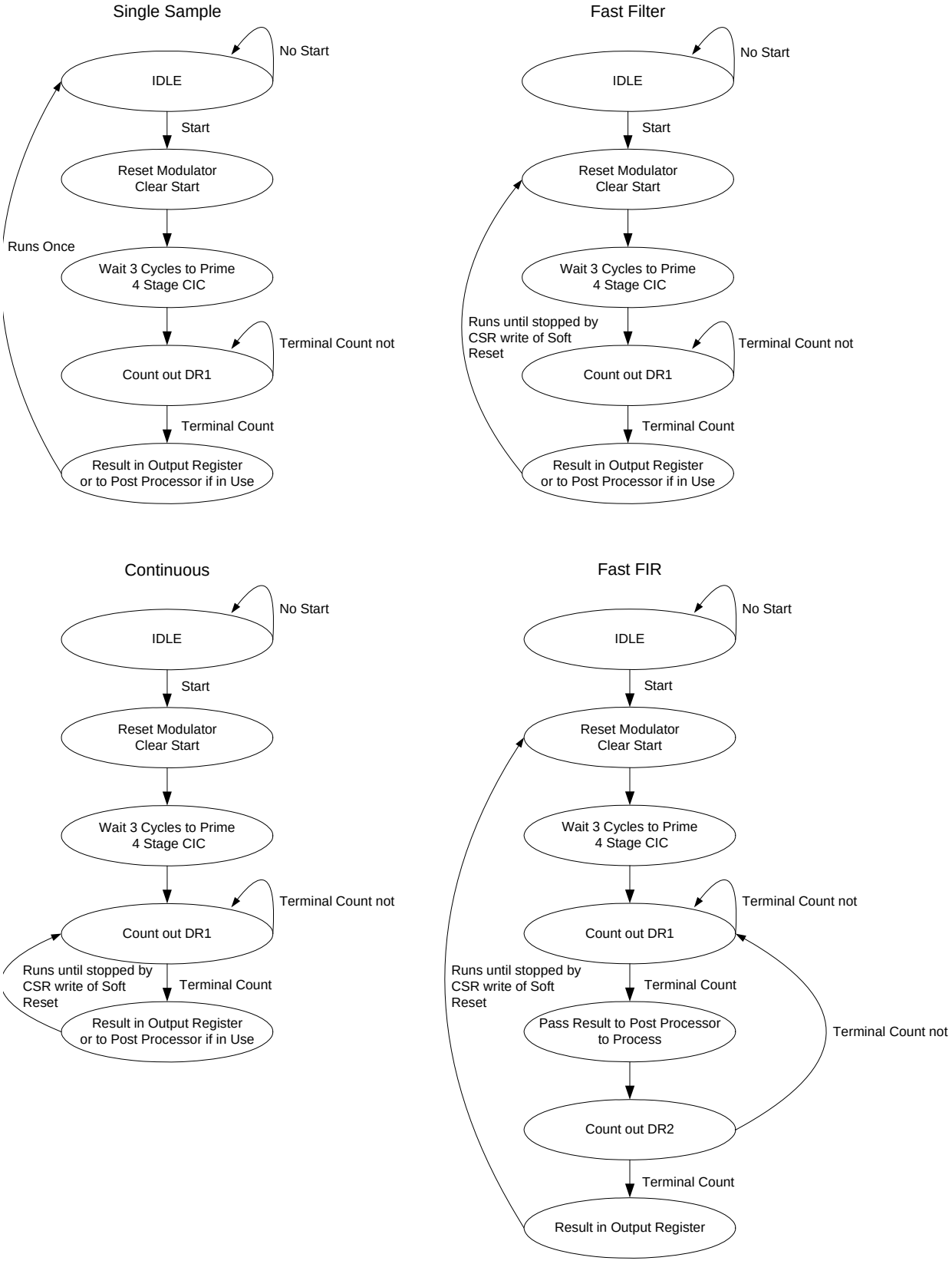
If signaled to run Continuous, the filter resets the channel then runs continuously from that point forward, until signaled to stop, with no intervening resets of the channel. The hardware blocks the first three decimation periods but then provides a result every decimation cycle thereafter.

Fast FIR mode is very much like 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 37-10 on page 401 shows the state diagram of various modes of operation of the decimator.

Figure 37-10. Decimator Modes



Section G: Program and Debug

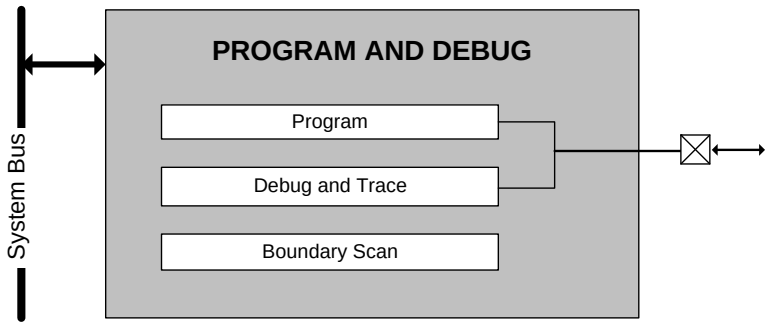
JTAG (4- or 5-wire) or serial wire debugger (SWD) (2-wire) interfaces are used to program and debug. The single wire viewer (SWV) can also be used for “printf” style debugging. By combining SWD and SWV, you can implement a full debugging interface with just three pins. These standard interfaces enables debugging or programming the PSoC® device with a variety of hardware solutions from Cypress or third-party vendors.

This section includes the following chapters:

- Test Controller chapter on page 405
- 8051 Debug on-Chip chapter on page 417
- Nonvolatile Memory Programming chapter on page 427

Top Level Architecture

Program and Debug Block Diagram



38. Test Controller

The PSoC® 3 architecture includes a test controller used for the following purposes:

- Access to I/O pins for boundary scan testing.
- Access to the device memory and registers (via the PHUB) through the PSoC 3 Debug on-Chip (DOC) module for functional testing, device programming, and program debugging.

The test controller connects to off-chip devices via the Joint Test Action Group (JTAG) interface or the serial wire debug (SWD) interface. These interfaces use I/O port pins; the exact number of pins depends on the type of interface used.

38.1 Features

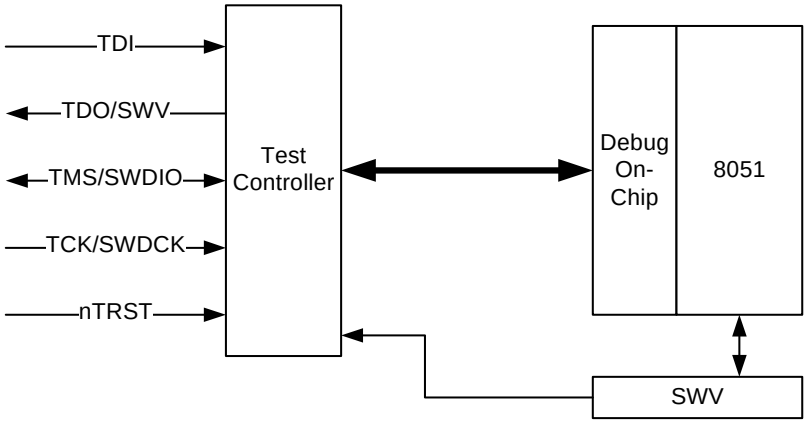
The test controller has the following features:

- Supports JTAG or SWD interface to a debug host
- SWD interface available on either GPIO or USB pins
- Supports boundary scan in accordance with the JTAG IEEE Standard 1149.1-2001 “Test Access Port and Boundary-Scan Architecture”
- Supports additional JTAG instructions/registers beyond IEEE Standard 1149, for access to the rest of the device
- Interfaces to PSoC 3 debug modules for access to the rest of the device for program and debug operations

38.2 Block Diagram

In PSoC 3 architecture, the test controller translates JTAG instructions/registers or SWD accesses to register accesses in the Debug on-Chip (DOC) module. See Figure 38-1.

Figure 38-1. PSoC 3 Test Controller Block Diagram



The abbreviations used in the figure are:

TC - Test Controller.

AHB - Advanced High-Performance Bus, def acto standard from ARM.

PHUB - Peripheral HUB, advanced multi spoke bus controller which allows many different functional blocks to communicate without involving of CPU for setting up the bus transaction.

DMA - Direct Memory Access controller.

SRAM - Static Random Access Memory.

SPC - System Performance controller implements R/W interface with non-volatile memory.

NVL - non-volatile latch.

38.3 Background Information

The following information helps to familiarize you with the JTAG interface and the IEEE 1149 Specification and the Serial Wire Debug (SWD) interface. An understanding of the state machine architecture in these interfaces is necessary to interact with the JTAG and SWD blocks in the test controller.

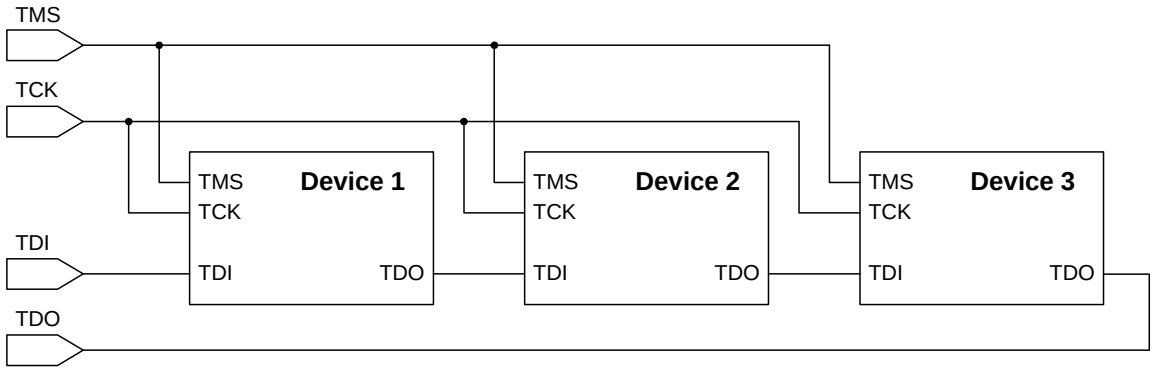
38.3.1 JTAG Interface

In response to higher pin densities on ICs, the Joint Test Action Group (JTAG) proposed a method to test circuit boards by controlling the pins on the ICs (and reading their values) via a separate test interface. The solution, later formalized as IEEE Standard 1149.1-2001, is based on the concept of a serial shift register routed across all of the pins of the IC – hence the name “boundary scan.” The circuitry at each pin is supplemented with a multipurpose element called a boundary scan cell. In PSoC 3 devices, most GPIO and SIO port pins have a boundary scan cell associated with them (see GPIO and SIO block diagrams in the I/O System chapter on page 159).

The interface used to control the values in the boundary scan cells is called the Test Access Port (TAP) and is commonly known as the JTAG interface. It consists of three signals: Test Data In (TDI), Test Data Out (TDO), and Test Mode Select (TMS). Also included is a clock signal (TCK) that clocks the other signals.

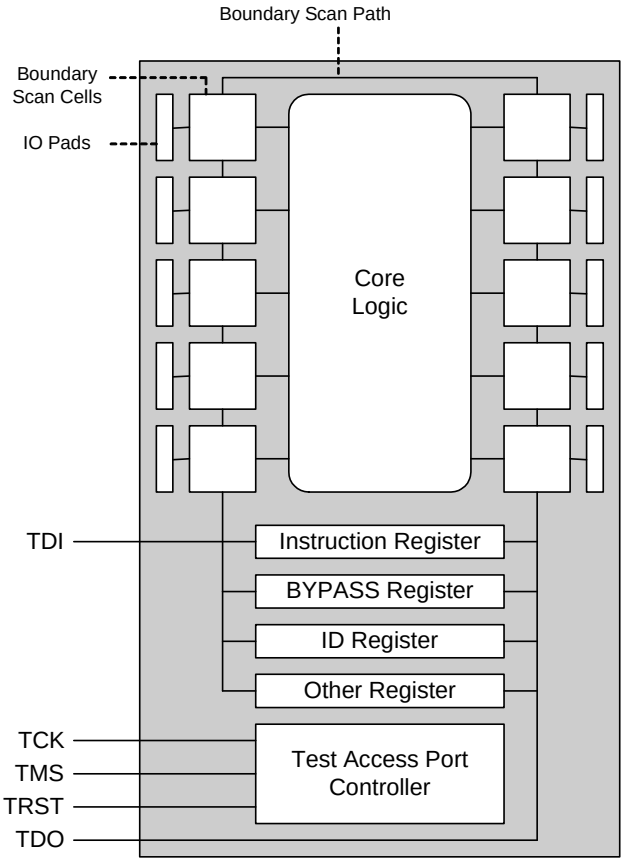
TDI, TMS, and TCK are all inputs to the device and TDO is output from the device. This interface enables testing multiple ICs on a circuit board, in a daisy-chain fashion, as shown in Figure 38-2.

Figure 38-2. JTAG Interface to Multiple ICs on a Circuit Board



Within each device, the JTAG interface architecture is shown in Figure 38-3. Data at TDI is shifted in, through one of several available registers, and out to TDO.

Figure 38-3. JTAG Interface Architecture

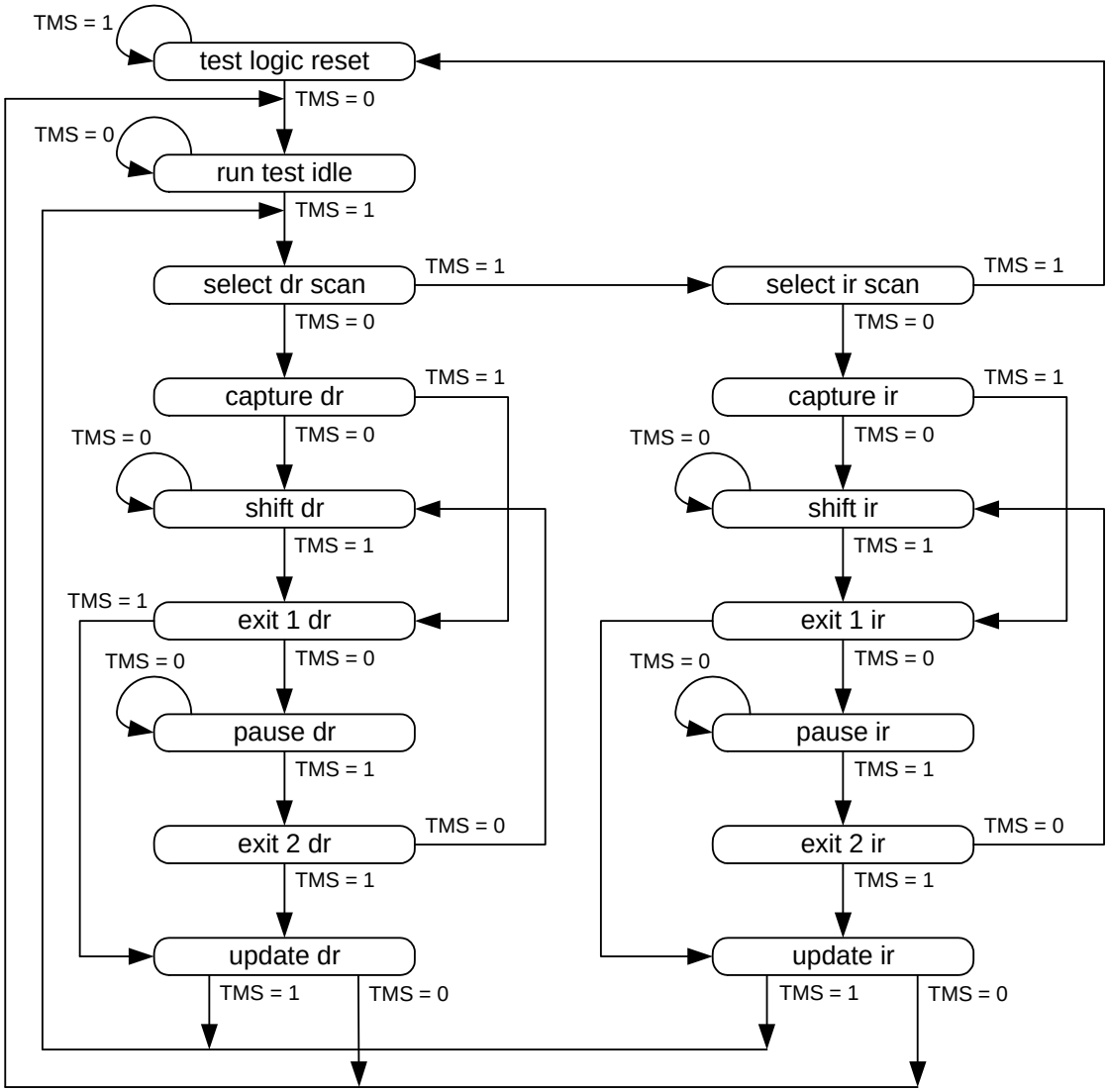


The TMS signal controls a state machine in the TAP. The state machine controls which register (including the boundary scan path) is in the TDI-to-TDO shift path, as shown in Figure 38-4 on page 408.

The following terms apply:

- **ir** – the instruction register
- **dr** – one of the other registers (including the boundary scan path), as determined by the contents of the instruction register
- **capture** – transfer the contents of a dr to a shift register, to be shifted out on TDO (read the dr)
- **update** – transfer the contents of a shift register, shifted in from TDI, to a dr (write the dr)

Figure 38-4. TAP State Machine



The registers in the TAP are:

- **Instruction** – Typically two to four bits wide, holds the current instruction that defines which data register is placed in the TDI-to-TDO shift path.
- **Bypass** – one bit wide, directly connects TDI with TDO, causing the device to be bypassed for JTAG purposes.
- **ID** – 32 bits wide, used to read the JTAG manufacturer/ part number ID of the device.
- **Boundary Scan Path (BSR)** – Width equals the number of I/O pins that have boundary scan cells, used to set or read the states of those I/O pins.

Other registers may be included in accordance with device manufacturer specifications.

The standard set of instructions (values that can be shifted into the instruction register), as specified in IEEE 1149, are:

- **EXTEST** – Causes TDI and TDO to be connected to the boundary scan path (BSR).

  The device is changed from its normal operating mode to a test mode. Then, the device's pin states can be sampled using the capture dr JTAG state, and new values can be applied to the pins of the device using the update dr state.
- **SAMPLE** – Causes TDI and TDO to be connected to the BSR, but the device is left in its normal operating mode

  During this instruction, the BSR can be read by the capture dr JTAG state to take a sample of the functional data entering and leaving the device.
- **PRELOAD** – Causes TDI and TDO to be connected to the BSR, but device is left in its normal operating mode.

The instruction is used to preload test data into the BSR before loading an EXTEST instruction.

Optional, but commonly available, instructions are:

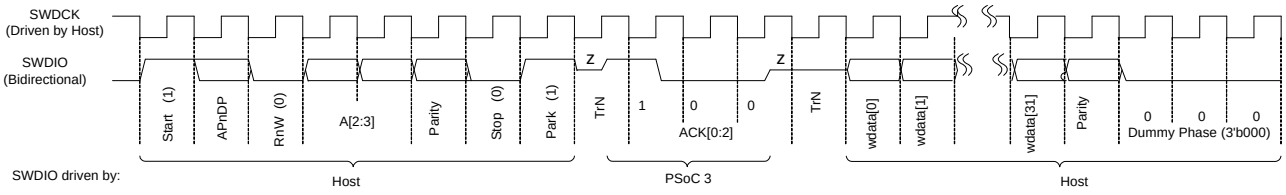
- **IDCODE** – Causes TDI and TDO to be connected to an IDCODE register.
- **INTEST** – Causes TDI and TDO to be connected to the BSR. While the EXTEST instruction allows access to the device pins, INTEST enables similar access to the core-logic signals of a device.

For more information, see the IEEE Standard, available at http://www.ieee.org.

38.3.2 Serial Wire Debug Interface

The SWD interface has two signals: data (SWDIO) and clock for data (SWDCK). The host programmer always drives the clock line, whereas either the programmer or PSoC 3 device drives the data line. Host programmer and PSoC 3 communicate in packet format through the SWD interface. Write packet refers to the SWD packet transaction in which the host writes data to PSoC 3. Read packet refers to the SWD packet transaction in which the host reads data from PSoC 3. The Write packet and Read packet formats are illustrated in Figure 38-5 and Figure 38-6, respectively

Figure 38-5. SWD 'Write Packet' Timing Diagram



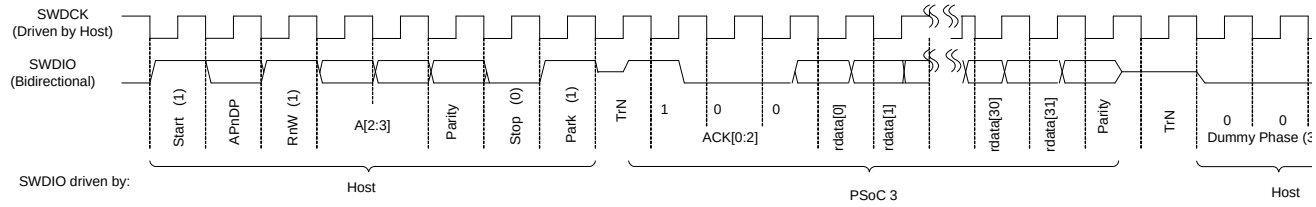
a.) Host Write Operation: Host sends data on the SWDIO line on the falling edge of SWDCK; PSoC 3 reads that data on the next SWDCK rising edge (Example: 8-bit header data, Write data(wdata[31:0]), Dummy phase (3'b000))

b.) Host Read Operation: PSoC 3 sends data on the SWDIO line on the rising edge of SWDCK; host reads that data on the next SWDCK falling edge (Example: 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 3 drives 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 3 and the host will not drive the line during the entire second TrN phase (indicated as 'z'). Host starts 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 3 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 38-6. SWD 'Read Packet' Timing Diagram



a.) Host Write Operation: Host sends data on the SWDIO line on the falling edge of SWDCK; PSoC 3 reads that data on the next SWDCK rising edge (Example: 8-bit header data, Dummy phase (3'b000))

b.) Host Read Operation: PSoC 3 sends data on the SWDIO line on the rising edge of SWDCK; the Host reads that data on the next SWDCK falling edge (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 3 drives 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 3 and the host will not drive the line during the entire second TrN phase (indicated as 'z'). Host starts sending the Dummy phase (3'b000) on the next falling edge of SWDCK after the 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 3 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 38-5 and Figure 38-6). Each data transfer consists of three phases:

- **Packet request** – External host programmer issues a request to PSoC 3.
- **Acknowledge response** – PSoC 3 sends an acknowledgement to the host.
- **Data** – This is valid only when a packet request is followed by a valid (OK) acknowledge response.

The data transfer is either:

- PSoC 3 to host, following a read request – RDATA
- Host to PSoC 3, following a write request – WDATA

In Figure 38-5 and Figure 38-6, 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 PSoC 3, or ‘0’ for a write to PSoC 3.
4. The ADDR bits (A[3:2]) are register select bits for access port or debug port. See Table 38-5 for address bit definitions.
5. The parity bit has the parity of APnDP, RnW, and ADDR. This is even parity bit. If number of logical 1’s 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 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 38-1. Note that the ACK in the current SWD transfer reflects the status of the previous transfer. OK ACK means the previous packet was 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 38-1. ACK Response for SWD Transfers

ACK[2:0]	JTAG	SWD
OK	010	001
WAIT	001	010
FAULT	100	100

   a. For a WAIT response, if the transaction is a read, the host ignores the data read in the data phase. PSoC 3 does not drive the line and the host must not check the parity bit as well.

   b. For a WAIT response, if the transaction is a write, PSoC® 3 ignores the data phase. However, 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. A WAIT response indicates that the PSoC 3 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 retry.

   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 3 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 PSoC 3 to change the Drive modes on their respective SWDIO line. During the first TrN phase after packet request, PSoC 3 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. The second TrN phase is one-and-a-half cycle long. Neither the host nor PSoC 3 should drive SWDIO line during both phases as indicated by ‘z’ in Figure 38-5 and Figure 38-6.
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 38-5 and Figure 38-6, there is a “DUMMY” phase, which is three SWD clock cycles with SWDIO line held low. The dummy phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 3 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 50 or more cycles with SWDIO high. To return to the idle state, SWDIO must be clocked low once. The host programmer can begin a new SWD packet transaction from the idle state.

38.4 How It Works

The PSoC 3 JTAG and SWD interfaces comply with standard specifications and offer extensions unique to PSoC 3 architecture.

38.4.1 Clocking

The clock signal (TCK) for JTAG mode and the data signal clock (SWDCK) for SWD interface share 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.

38.4.2 PSoC 3 JTAG Instructions

The PSoC 3 JTAG interface complies with the IEEE 1149.1-2001 specification, and provides additional instructions. The instruction register is 4 bits wide. Instructions are listed in Table 38-2.

Table 38-2. Additional PSoC 3 JTAG Instructions

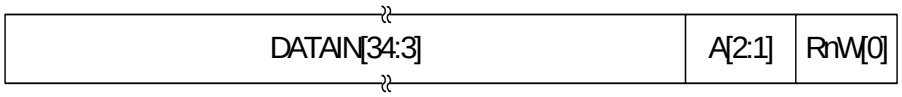
Bit Code	Instruction	PSoC 3 Function
1111	BYPASS	See IEEE 1149.1-2001
1110	IDCODE	See IEEE 1149.1-2001
0010	SAMPLE / PRELOAD	See IEEE 1149.1-2001. SAMPLE and PRELOAD share the same bit code.
0000	EXTEST	See IEEE 1149.1-2001
0100	INTEST	Same as EXTEST
0101	CLAMP	Connects TDI and TDO to the BYPASS register, and sets the pins to the current contents of the boundary scan register
1010	DPACC	Connects TDI and TDO to the DP/AP Access register, for accesses to the Debug Port registers.
1011	APACC	Connects TDI and TDO to the DP/AP Access register, for accesses to the Access Port registers.
1100	SLEEP	Notifies the device power manager that it may power down the Test Controller (TC) if necessary. If this instruction is not set then the TC cannot be put to sleep.

38.4.3 DP/AP Access Register

PSoC 3 architecture has a DP/AP Access register that is 35 bits wide. This register is used to transfer data between the JTAG or SWD interface and the Debug Port and Access Port registers. The SWD interface enables direct reads and writes of the DP/AP Access register; the JTAG interface uses the DPACC and APACC instructions.

In the JTAG update dr state or when writing the register from the SWD interface, the structure is as shown in the following figure.

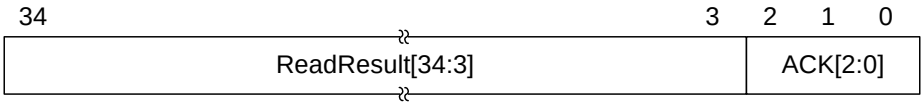
Figure 38-7. Writing the Register



- **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.
- **Bit 0 – RnW** – 1 = read (from device to debug host); 0 = write (to device from debug host)

In the JTAG capture dr state, or when reading the register from the SWD interface, the structure is as shown in the following figure.

Figure 38-8. 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 – Depending on the interface, the ACK response is as indicated in Table 38-3.

Table 38-3. ACK Response for JTAG/SWD Transfers

ACK[2:0]	JTAG	SWD
OK	010	001
WAIT	001	010
FAULT	100	100

38.4.4 JTAG/SWD Addresses (PSoC 3)

In the PSoC 3 architecture, the 2-bit address, transferred by either JTAG or SWD as described above, is used to access Debug Port, Access Port, and ID Code registers as shown in Table 38-4.

38.4.5 Debug Port and Access Port Registers (PSoC 3)

The DP and AP registers listed in Table 38-4 are part of TC. JTAG has separate instructions (DPACC, APACC) to distinguish between AP and DP access; the SWD protocol uses the APnDP bit for this.

Table 38-4. Debug Port and Access Port Registers (PSoC 3)

Register Name	Register Type	Address (A[3:2])	Function
IDCODE	DP	00	32-bit Device JTAG IDCODE register. IDCODE register address bits are applicable only for SWD interface. JTAG interface has a separate IDCODE instruction to access IDCODE register directly.
DBGPRT_CFG	DP	01	8-bit Debug Port Configuration register - Bits [2:1] determine if the transfer size is 8, 16, or 32 bits. Bit [3] enables the auto address increment functionality. For 8-bit transfer size, write 2'b00 to Bits [2:1]; for 16-bit transfer size, write 2'b01 to Bits [2:1]; and for 32-bit transfer size, write 2'b10 to Bits [2:1]. To enable auto increment of address, write a 1'b1 to Bit [3]. Remaining bits are written '0'.
READBUFF	DP	11	Port Acquire key is written to this 32-bit register to acquire debug port for programming PSoC 3 through the SWD interface.
TRNS_ADDR	AP	01	24-bit Transfer Address register that holds the address that is used for PSoC 3 register access.
DATA_RW	AP	11	32-bit Data register that holds the data to be read from/written to the address specified by the TRNS_ADDR register.

38.4.6 Register Access Examples

The following directions show how to access an address, using either the JTAG or the SWD interface. Assume the address value to be 0xADD8E5.

1. To use JTAG to write the address value to the TRNS_ADDR register, the debug host must:
   a. Shift the APACC instruction into the Instruction register.
   b. Shift a 0 (write) followed by a 01 (Access Port register select) followed by a 0x00ADD8E5 (32-bit address), into the 35-bit DP/AP Access register. For each element, the LS bit is shifted first.
   c. Go to the JTAG update dr state.
2. To use SWD to write the address value to the TRNS_ADDR register, the debug host must:
   a. Send a packet request where APnDP = 1, RnW = 0, and ADDR = 01.
   b. Get an ACK response from the PSoC device.
   c. In the data phase, send 0x00ADD8E5.
3. To write a value 0xDA to address 0xADD8E5:
   a. Write 0x00ADD8E5 to the TRNS_ADDR register, as described above.
   b. Write 0x000000DA to the DATA_RW register, as described above except that the address is 11 instead of 01.
   c. The test controller initiates a write transfer request to the PSoC 3 DOC.
4. To read from address 0xADD8E5:
   a. Write 0x00ADD8E5 to the TRNS_ADDR register, as described above.
   b. Read the DATA_RW register as described above except that the address is 11 instead of 01 and the RnW bit is 1 instead of 0.
   c. The test controller initiates a read transfer request to the PSoC 3 DOC; the data read from DATA_RW is invalid.
   d. Wait at least 5 TCK/SWDCK clock cycles, to avoid a WAIT response.
   e. Read the DATA_RW register again. The data is now valid.

Up to four sequential addresses can be written/read by setting the transfer size in DBGPRT_CFG. If the TRNS_ADDR auto-increment bit is set, sequential addresses can be written/read without updating TRNS_ADDR at each access.

To do this, invoke the DPACC instruction through either JTAG or SWD. And then access the appropriate Debug Port configuration registers.

For PSoC 3 there is only one Debug Port configuration register, DBGPRT_CFG. Bits [2:1] determine if the transfer size is 8,16 or 31 bits. Bit [3] enables the auto address increment functionality. For 8-bit increment write 2'b00 to Bits [2:1], for 16 bit increment write 2'b01 to Bits [2:1], and for 32 bit increment write 2'b10 to Bits [2:1]. To enable auto increment write a 1'b1 to Bit [3].

38.4.7 Boundary Scan Pin Order

For the 100-pin TQFP device, the boundary scan path (BSR) is connected to the I/O pins around the part from TDI (P1[4]) through TDO (P1[3]), in the order shown in the following table.

Table 38-5. Boundary Scan Pin Order

BSR#	Pin	BSR#	Pin	BSR#	Pin	BSR#	Pin	BSR#	Pin	BSR#	Pin
1	P1[5]	13	P15[1]	25	P15[3]	37	P0[7]	49	P15[5]	61	P6[5]
2	P1[6]	14	P3[0]	26	P12[2]	38	P4[2]	50	P2[0]	62	P6[6]
3	P1[7]	15	P3[1]	27	P12[3]	39	P4[3]	51	P2[1]	63	P6[7]
4	P12[6]	16	P3[2]	28	P4[0]	40	P4[4]	52	P2[2]	64	XRES_N
5	P12[7]	17	P3[3]	29	P4[1]	41	P4[5]	53	P2[3]	65	P5[0]
6	P5[4]	18	P3[4]	30	P0[0]	42	P4[6]	54	P2[4]	66	P5[1]
7	P5[5]	19	P3[5]	31	P0[1]	43	P4[7]	55	P2[5]	67	P5[2]
8	P5[6]	20	P3[6]	32	P0[2]	44	P6[0]	56	P2[6]	68	P5[3]
9	P5[7]	21	P3[7]	33	P0[3]	45	P6[1]	57	P2[7]	69	P1[2]
10	P15[6]	22	P12[0]	34	P0[4]	46	P6[2]	58	P12[4]		
11	P15[7]	23	P12[1]	35	P0[5]	47	P6[3]	59	P12[5]		
12	P15[0]	24	P15[2]	36	P0[6]	48	P15[4]	60	P6[4]		

Similar boundary scan paths exist on the 68-pin QFN and 48-pin SSOP parts.

38.5 Test Controller Interface Pins

Two NV latch bits determine the state of the JTAG/SWD interface pins at reset. The settings of the bits are shown in the following table.

Table 38-6. JTAG / SWD Interface Bit Settings

CNVL_DPS[1:0]	Port Configuration
00	5 – Wire JTAG (nTRST is included)
01 (default)	4 – Wire JTAG (nTRST is not used)
10	Serial wire debug (SWD)
11	Debug Port Disabled (GPIO)

In addition, PSoC 3 architecture contains a single wire viewer (SWV) module, whose interface consists of a single output signal SWV. The test controller routes SWV to the JTAG/SWD pins; SWV shares a pin with the JTAG TDO signal. When the pins are configured for SWD mode then SWV is also routed to the TDO/SWV pin. For further details on the SWV, see the 8051 Debug on-Chip chapter on page 417.

38.6 Test Controller Acquisition

If the debug port is disabled, 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 38-7. 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

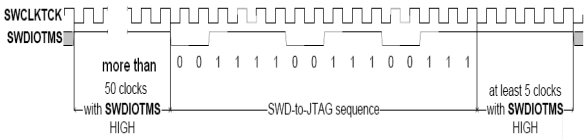
The SWD frame should be transmitted at least twice, ignoring the ACK on the first transmit; it should be transmitted until the ACK response is OK. The SWD interface will be in the idle state, ready for the next write. The debug host can then either continue using the SWD interface or switch to the JTAG interface. See PSoC 3 Device Programming Specifications for detailed timing diagrams on the test controller acquisition.

38.6.1 Changing Interface from SWD to JTAG

To switch programming interface from SWD to JTAG (4-wire) operation, the steps are as follows:

1. Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that the current interface is in its reset state. The serial wire interface detects the 16-bit SWD-to-JTAG sequence only when it is in the reset state.
2. Send the 16-bit SWD-to-JTAG select sequence on **SWDIOTMS**. The 16-bit SWD-to-JTAG select sequence is 0b0011_1100_1110_0111, MSB first. This can be represented as either:
   a. 0x3CE7 transmitted MSB first.
   b. 0xE73C transmitted LSB first.

Figure 38-9. SWD to JTAG Switching Sequence



3. Send at least five **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that if programming interface is already in JTAG operation before sending the select sequence, the JTAG TAP enters the Test-Logic-Reset state.

38.6.2 Changing Interface from JTAG to SWD

To switch programming interface from JTAG to SWD operation, the sequence is:

1. Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that the current interface is in its reset state. The JTAG interface only detects the 16-bit JTAG-to-SWD sequence starting from the Test-Logic-Reset state.
2. Send the 16-bit JTAG-to-SWD select sequence on **SWDIOTMS**. The 16-bit JTAG-to-SWD select sequence is 0b0111_1001_1110_0111, MSB first. This can be represented as either:
   a. 0x79E7 transmitted MSB first
   b. 0xE79E transmitted LSB first.

Figure 38-10. JTAG-to-SWD Sequence

Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that if programming interface is already in SWD operation before sending the select sequence, the SWD interface enters line reset state.

38.6.3 Boundary Scan

To perform a boundary scan:

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. Shift to the JTAG interface.
4. Start doing boundary scan operations.

38.6.4 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 or, if desired, shift to the JTAG interface.
4. Start writing or reading PSoC 3 devices to or from registers and memory for functional tests.

38.6.5 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 or if desired shift to the JTAG interface.
4. Start writing or reading to or from PSoC 3 SPC registers to program flash/EEPROM. See the Nonvolatile Memory Programming chapter on page 427 for details.

38.6.6 Program Debug/Trace

To perform a debug or trace:

1. At any time during normal operation, use the SWD or JTAG interface as set up by the NV latch bits.
2. Start writing or reading to or from the PSoC 3 DOC debug module registers to do program debug/trace operations.

In PSoC 3, the test controller can access the DOC only when the DOC is enabled. This is done under program control by setting the DEBUG_ENABLE bit in the MLOGIC_DBG_DBE[0] register. The bit is 0 at reset.

39. 8051 Debug on-Chip

PSoC® 3 debug modules consist of the Debug on-Chip (DOC) and the single wire viewer (SWV). The DOC interfaces between the CPU and the Test Controller (TC). It is used to debug and trace code execution and to troubleshoot device configuration. The DoC exists only on the 8051-based PSoC 3 family.

The SWV module allows target resident code to communicate diagnostic information to the outside world through a single pin. Usage examples include data monitoring, viewing OS task switches, printf debugging, and call graph profiling.

39.1 Features

The DOC is capable of taking over the 8051 CPU and using its PHUB interface to access any address accessible by the CPU. It provides the following features:

- TC interface for access via either JTAG or SWD
- Access of CPU internal memory and SFRs, and the Program Counter (PC) (see the 8051 Core chapter on page 37)
- CPU halt
- CPU single step through instructions
- 8 program address breakpoints
- 1 memory access breakpoint
- Watchdog trigger breakpoint
- Breakpoint chaining
- Trace CPU instruction execution:
  - Trace CPU program counter (PC), accumulator (ACC), and one byte from CPU internal memory or SFR
  - 2048 instruction trace buffer if tracing PC only
  - 1024 instruction trace buffer if tracing PC, ACC, and a memory/SFR byte
  - Continuous, triggered, or windowed mode
  - On trace buffer full, either CPU halt or overwrite oldest trace
  - When not tracing, trace buffer is available as normal SRAM

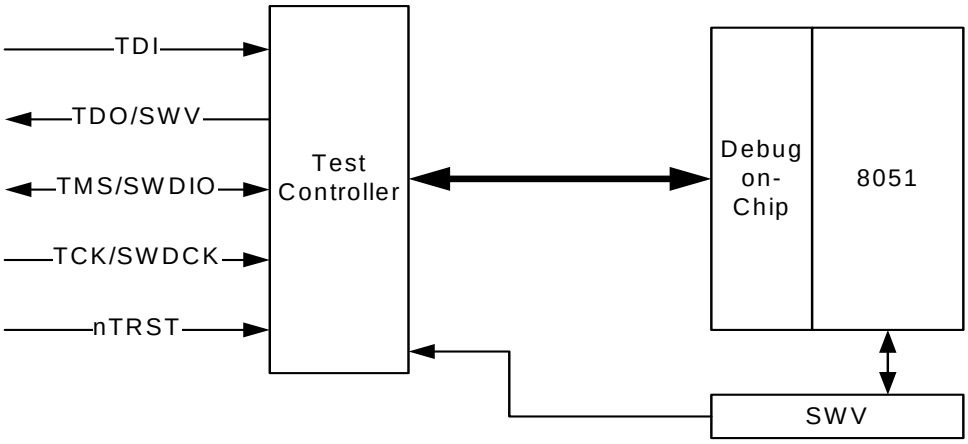
The SWV has the following features:

- 32 stimulus port registers
- Simple, efficient packing and serializing protocol
- Two pin output modes, UART or Manchester encoding

39.2 Block Diagram

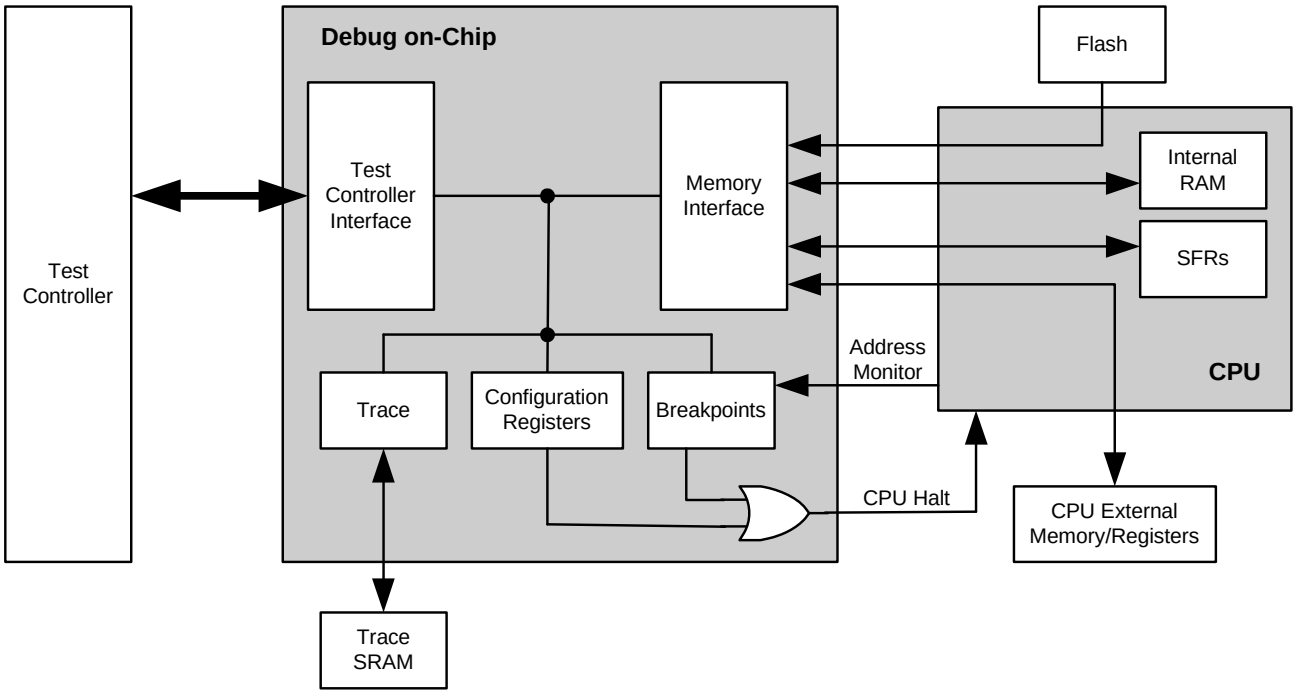
In PSoC 3 devices, the TC translates JTAG instructions and registers or SWD accesses to register accesses in the Debug on-Chip (DOC). SWV data is output through the TC onto a single pin SWV. The SWV pin is shared with the JTAG TDO signal. Figure 39-1 shows a block diagram of the relationship among the TC, DOC, CPU, and SWV. (See the Test Controller chapter on page 405.)

Figure 39-1. TC, DOC, CPU, and SWV



The DOC module interfaces between the CPU and the TC, as shown in Figure 39-2. The memory interface is used for system reads and writes through the CPU.

Figure 39-2. DOC, CPU, and TC Block Diagram



39.3 How it Works

DOC functions are controlled by accessing a series of registers within the DOC. The DOC registers are accessible only through the TC interface; they are not accessible through the PHUB.

39.3.1 Enabling and Activating

Before any DOC operations can be done, debugging must be enabled by the CPU. Bit 0, debug_enable, in the MLOGIC_DEBUG register, is used by the CPU to enable or disable debug, as shown in Table 39-1.

Table 39-1. MLOGIC_DEBUG, debug_enable, Bit 0

Setting	Description
1	Enable debug
0 (default)	Disable debug

Then, the DOC module must be activated by the TC interface. Bit 0, DBG_ACT, of the debug control register, DOC_DBG_CTRL, is used by the TC to activate or deactivate debug, as shown in Table 39-2.

Table 39-2. DOC_DBG_CTRL, DBG_ACT, Bit 0

Setting	Description
1	Activate debug
0 (default)	Deactivate debug

Setting the DBG_ACT bit also pauses the Central Timewheel (CTW). This makes certain that CTW interrupts or Watchdog Resets (WDR) do not occur while the CPU is paused. Every write to the debug control register, DOC_DBG_CTRL, must contain in bits [7:4] the debug key value 0b1011, otherwise the write is ignored.

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.

39.3.2 Halting, Stepping

Bit 1, HALT, of the debug control register, DOC_DBG_CTRL, controls CPU halt, as shown in Table 39-3.

Table 39-3. DOC_DBG_CTRL, HALT, Bit 1

Setting	Description
1	Halt the CPU
0 (default)	Un-halt the CPU

Bit 2, STEP, of the debug control register, DOC_DBG_CTRL, is used to control CPU stepping, as shown in Table 39-4.

Table 39-4. DOC_DBG_CTRL, STEP, Bit 2

Setting	Description
1	Single-step the CPU
0 (default)	No effect

After writing a 1 to this bit, the CPU executes one instruction then halts. The STEP bit is reset to 0. The following applies:

- The HALT bit must be set to 1 to write the STEP bit.
- The debug enable and the debug activate bits must both be set to 1 to write the HALT or STEP bit.
- Every write to the debug control register, DOC_DBG_CTRL, must contain in bits [7:4] the debug key value 0b1011, otherwise the write is ignored.

39.3.3 Accessing PSoC Memory And Registers

The TC receives debug commands via JTAG or SWD, and passes them to the DOC. Based on the received address, either the TC or DOC registers are accessed, or the address and data are passed on to the DOC memory interface. The DOC has multiple memory interfaces; it decodes the incoming address and sends it to the correct memory interface address output. When the memory access is complete, the DOC signals the TC either that the write is complete or that data from a read command is available.

TC commands specify the data size as 8, 16, or 32 bits wide; along with the address and data, the data size is passed to the DOC memory interface. All accesses to the TC and DOC registers are 32 bits wide; the data size setting does not apply.

The DOC memory interface is used for system reads and writes through the CPU. The DOC memory interface takes over the CPU memory interfaces, allowing the DOC to perform reads and writes to memory as if the request were coming from the CPU. There are four memory interfaces: flash, CPU internal memory, CPU SFRs, and CPU external memory/registers. There is also a CPU Program Counter (PC) interface. The interface chosen for a read or write transaction depends on the address. See the following table.

Table 39-5. PSoC Memory and Registers

Address Range	Description
0x050000 – 0x0500FF	CPU internal memory
0x05014E – 0x05014F	CPU PC (16-bit register)
0x050180 – 0x0501FF	CPU SFR space
0x050200 – 0x05FFFF	TC and DOC registers
all other addresses	CPU external memory/registers

The CPU must be halted before reading or writing the PC.

39.3.4 Breakpoints

The PSoC 3 DOC has eight program address breakpoints, one memory access breakpoint, and a watchdog trigger breakpoint. These breakpoints are used to halt the CPU at specified events in the execution of the program.

39.3.4.1 Program Address Breakpoints

The program address breakpoints use the eight registers DOC_PA_BKPT0 through DOC_PA_BKPT7. To set a program address breakpoint, write the address compare value to bits [15:0] and set the breakpoint enable in bit 16. When the CPU Program Counter (PC) matches the compare value on an instruction fetch, the CPU is halted.

Because the PC address is compared, the memory access breakpoint is also able to act as a program access breakpoint.

Before the CPU is unhalted, the breakpoint enable bit must be reset to '0'.

39.3.4.2 Memory Access Breakpoint

To set the memory access breakpoint, use the DOC_MEM_BKPT register. Write the address compare value to bits [23:0], and write one of the following to the configuration bits [25:24]: See Table 39-6.

Table 39-6. DOC_MEM_BKPT, Configuration Bits [25:24]

Setting	Description
00 (default)	Memory breakpoint disabled
01	Break on read only
10	Break on write only
11	Break on read or write

The compare address value is compared with all CPU addresses — PC, internal memory, SFRs, and external memory. When any of those addresses matches the compare value and the read or write access matches the configuration bits, the CPU is halted. Note that because the PC address is compared, the memory access breakpoint can also act as a program access breakpoint.

Before the CPU is unhalted, the configuration bits must be reset to disabled, 00.

39.3.4.3 Watchdog Trigger Breakpoint

If the Watchdog is triggered during debug mode, Watchdog Reset (WRES) can be used as a breakpoint to halt the CPU instead of resetting the system. Bit 10, WDRBKPT, of the breakpoint configuration register DOC_BKPT_CFG, is used to enable or disable this feature. See Table 39-7.

Table 39-7. DOC_BKPT_CFG, WDRBKPT, Bit 10

Setting	Description
1	Disable WRES break in debug mode
0 (default)	Enable WRES break in debug mode

When enabled with the Watchdog triggered, the DoC halts the CPU. When the CPU is unhalted, WRES is asserted to the system.

The BKPT_CFG register also contains a bit Watchdog Reset Triggered (bit 11, WDR_TRG). This is a read-only bit that can be used to determine if the CPU was halted because a WDR occurred. The bit is set to 1 if the CPU was halted due to WDR, otherwise it is read as a 0.

39.3.4.4 Breakpoint Chaining

Breakpoint chaining is used to halt the CPU after a series of breakpoints occur. Besides the memory access breakpoint, all program address breakpoints can be chained.

To set up breakpoint chaining, do the following:

1. Set all of the required compare values in registers DOC_PA_BKPT0 – DOC_PA_BKPT7, and DOC_MEM_BKPT.
2. Set the memory access read/write configuration bits in DOC_MEM_BKPT.
3. Do not set the breakpoint enable bits in DOC_PA_BKPT0 – DOC_PA_BKPT7.
4. After the compare values are set, set the breakpoint chain enable bit, BC_ENA, in DOC_BKPT_CFG[0], to 1.
5. Set the chain include bits in DOC_BKPT_CFG to 1 for each breakpoint included in the chain. There is one bit for each breakpoint, as shown in Table 39-8.

Table 39-8. DOC_BKPT_CFG

Chain Include Bit	Corresponding Breakpoint
DOC_BKPT_CFG [1], CBKPT_0	DOC_PA_BKPT0
DOC_BKPT_CFG [2], CBKPT_1	DOC_PA_BKPT1
DOC_BKPT_CFG [3], CBKPT_2	DOC_PA_BKPT2
DOC_BKPT_CFG [4], CBKPT_3	DOC_PA_BKPT3
DOC_BKPT_CFG [5], CBKPT_4	DOC_PA_BKPT4
DOC_BKPT_CFG [6], CBKPT_5	DOC_PA_BKPT5
DOC_BKPT_CFG [7], CBKPT_6	DOC_PA_BKPT6
DOC_BKPT_CFG [8], CBKPT_7	DOC_PA_BKPT7
DOC_BKPT_CFG [9], CBKPT_8	DOC_MEM_BKPT

The chain include bits cannot be set unless BC_ENA is set to 1.

The breakpoints are chained in numeric order; a lower numbered breakpoint must occur before a higher numbered

breakpoint. For example, PA_BKPT5 cannot be set as the trigger for PA_BKPT3. When the last breakpoint in the chain is triggered, the CPU is halted.

When breakpoint chaining is enabled, breakpoints not in the chain can still be enabled; the CPU can be halted either on an individual breakpoint or on a chain of breakpoints.

The register DOC_BKPTCS is used to determine the breakpoints in the chain that have or have not yet triggered. There is one bit for each breakpoint. See Table 39-9.

Table 39-9. DOC_BKPTCS

Remaining Breakpoint in Chain (RBIC) Bit	Corresponding Breakpoint
DOC_BKPTCS [0]	DOC_PA_BKPT0
DOC_BKPTCS [1]	DOC_PA_BKPT1
DOC_BKPTCS [2]	DOC_PA_BKPT2
DOC_BKPTCS [3]	DOC_PA_BKPT3
DOC_BKPTCS [4]	DOC_PA_BKPT4
DOC_BKPTCS [5]	DOC_PA_BKPT5
DOC_BKPTCS [6]	DOC_PA_BKPT6
DOC_BKPTCS [7]	DOC_PA_BKPT7
DOC_BKPTCS [8]	DOC_MEM_BKPT

If a bit is set to 1, that breakpoint is part of the chain and has not yet triggered. If all bits are set to 0 and breakpoint chaining is enabled, then all breakpoints in the chain have triggered, and the CPU is halted.

Before the CPU is unhalted, the breakpoint chain enable bit, BC_ENA, must be reset to 0.

39.3.5 CPU Reset

The CPU can be held in a reset state by setting the bit RST, in DOC_CPU_RST[0]. This setting has no effect on overall system resets.

39.3.6 Tracing Program Execution

When CPU program tracing is enabled, as each CPU instruction is executed, copies of various CPU registers are written to a trace buffer. This operation is done in real time; neither CPU nor system speed is affected. The trace buffer can be examined to review program execution history.

The trace buffer size is 4096 bytes. If tracing is not being done, the trace buffer can be used as an additional 4K of SRAM, operating in the same manner as the rest of SRAM.

To enable tracing, set the trace control bits, TRC_CNTRL, in DOC_TRC_CFG[1:0]. SeeTable 39-10.

Table 39-10. DOC_TRC_CFG[1:0], TRC_CNTRL

Setting	Description
00 (default)	Trace disabled – trace buffer is available for use as SRAM
01	Trace in Continuous mode
10	Trace in Trigger mode
11	Trace in Window mode

The following applies:

- In Continuous mode, trace runs constantly until the CPU is halted.
- In Trigger mode, trace starts running when the CPU PC equals the compare value in breakpoint register #6, DOC_PA_BKPT6. The breakpoint enable bit in this register need not be set. Trace then runs constantly until the CPU is halted.
- In Window mode, trace starts running when the CPU PC equals the compare value in breakpoint register #6, DOC_PA_BKPT6. Trace then runs constantly until the PC equals the compare value in breakpoint register #7, DOC_PA_BKPT7 or until the CPU is halted. The breakpoint enable bits in these registers need not be set. Trace restarts if the PC equals breakpoint register #6 again. If both breakpoint registers have the same value, no tracing is done.

The CPU registers written to the trace buffer are controlled by bit 2, TRC_FLTR, in DOC_TRC_CFG[2]. See Table 43-11.

Table 39-11. DOC_TRC_CFG[2], TRC_FLTR

Setting	Registers Written to Trace Buffer	Maximum Number of Instructions Traced
0 (default)	PC, accumulator (ACC), and one byte of CPU internal memory/SFR – 32 bits total	1024
1	PC only – 16 bits	2048

The address of the internal memory/SFR byte is set in the TRC_PMEM bits of register DOC_TRC_CFG[15:8]. Values 0x00 – 0x7F address the lower 128 bytes of CPU internal memory. Values 0x80 – 0xFF address the CPU SFRs.

The registers are written to the trace buffer after the instruction is executed; the values shown in Table 39-12 are saved.

Table 39-12. Saved Values in the Trace Buffer

Register	Value Written to Trace Buffer
PC	Address (in flash) of the first byte of the instruction
ACC	Value after instruction execution
Internal memory/ SFR byte	Value after instruction execution

For example, if the instruction is POP 0xE0 (0xE0 is the SFR address of ACC), and TRC_PMEM = 0x81 (the SFR address of SP), then, in addition to the address of the instruction, the trace contains the value popped from the stack and the new value of the stack pointer.

When tracing in trigger point or windowed mode, the internal memory/SFR byte must be initialized before tracing can begin. This is done as follows:

1. Load TRC_PMEM bits with the byte address.
2. Write a value to the memory/SFR at that address. If the value cannot be changed, read it first, then write the read value back to the address.
3. Set the TRC_CNTRL bits to enable tracing.

The CPU can be halted when the trace buffer is full. This is controlled by the bit TRC_FULL, in DOC_TRC_CFG[3]. See Table 39-13.

Table 39-13. DOC_TRC_CFG, TRC_FULL

Setting	Description
0 (default)	Don't halt CPU, oldest trace is overwritten
1	Halt CPU

When the CPU is unhalted, tracing may restart. Because the CPU is halted when the buffer is full, tracing always restarts at the beginning of the buffer.

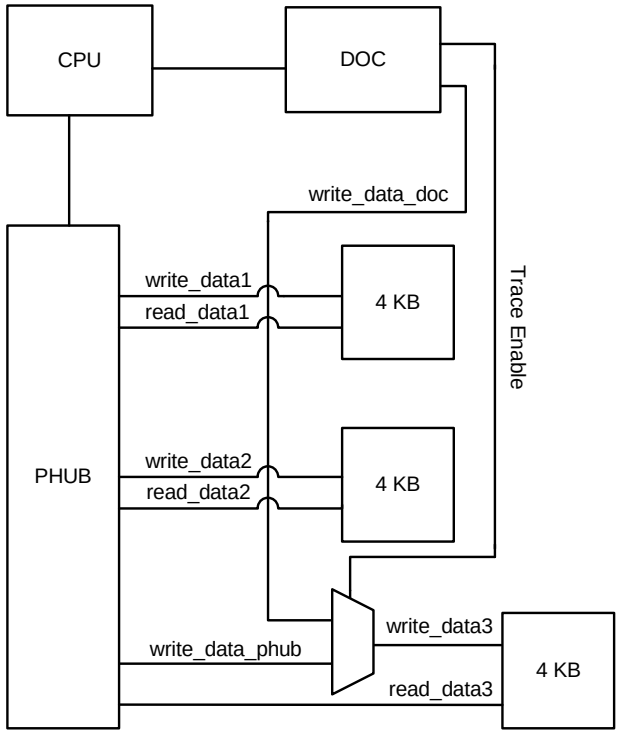
In Overwrite mode, the overwrite address is not available; therefore, when the trace buffer is examined, it is impossible to tell which trace is the oldest.

39.3.6.1 Reading Traces

The trace buffer memory is located in the CPU external memory space at address 0x002000; in a part with 8K SRAM the trace buffer is contiguous to SRAM.

Writes to this memory are done either by the DOC in trace mode or through the PHUB (by the CPU or DMAC). All reads from the memory are done through the PHUB. See Figure 39-3.

Figure 39-3. Trace Buffer Memory



At anytime, an external debug system can read the trace data via the JTAG/SWD interface, TC, DOC, CPU, and PHUB.

39.3.6.2 Trace Time Stamp

Two registers are included in the DOC for time stamping (counting the number of cycles in a trace window), as shown in Table 39-14. The registers can be used only in trace Window mode. Both registers are 32 bits wide.

Table 39-14. DOC Time Stamping Registers

Register	Value
ENTR_TS	Number of clock cycles from when the trace is enabled and CPU started and the trace window is enabled
EXIT_TS	Number of clock cycles from when the trace is enabled and CPU started to the exit point of the trace window

39.3.7 DOC Registers

The DOC registers are accessible only through the TC interface; they are not accessible through the PHUB.

Table 39-15 shows these registers.

Table 39-15. DOC Registers

Register	Size (Bits)	Description
DOC_DBG_CTRL	8	Debug control
DOC_PA_BKPTx	24	Program address breakpoint 0 to 7
DOC_MEM_BKPT	32	Memory access breakpoint
DOC_BKPT_CFG	16	Breakpoint configuration
DOC_BKPTCS	16	Breakpoint chain status
DOC_TRC_CFG	16	Trace configuration
DOC_PC	16	CPU Program Counter
DOC_CPU_RST	8	CPU reset control
DOC_ENTR_TS	32	Entry time stamp
DOC_EXIT_TS	32	Exit time stamp

39.4 Serial Wire Viewer

In addition to the DOC, the PSoC 3 includes a Serial Wire Viewer (SWV) module. The SWV allows target resident code to communicate diagnostic information to the outside world through a single pin. Usage examples include data monitoring, viewing OS task switches, printf debugging, and call graph profiling.

SWV data is output through the TC onto a single pin SWV. The SWV pin is shared with the JTAG TDO signal and GPIO pin P1[3]. To connect the pin to SWV, set to SWD mode the NV latch bits that determine the state of the JTAG/SWD interface pins at reset. (See the Test Controller chapter on page 405.) SWV can be used simultaneously with SWD, but not with JTAG.

The SWV is composed of two CoreSight™ components produced by ARM (http://www.arm.com). The components are the Instrumentation Trace Macrocell (ITM) and the Serial Wire Output (SWO). Both components have multiple data, control and status registers. For details on these registers see the ARM document *CoreSight™ Components Technical Reference Manual*.

To control the bit rate of the output, use the Prescaler, in register SWV_SWO_CAOSD[12:0] (register CODR in the ARM document). The SWV is clocked by BUS_CLK, which must be divided down to the desired bit rate. The actual divisor is the value of the Prescaler plus one. The maximum frequency that can exist on any port pin is 33 MHz.

To select an output protocol, use the Pin Protocol bits, in register SWV_SWO_SPP[1:0] (register SPPR in the ARM document). See Table 39-16.

Table 39-16. SWV_SWO_SPP[1:0], Pin Protocol Bits

Setting	Description
00	Reserved
01 (default)	Manchester
10	UART
11	Reserved

To output instrumentation trace data to the SWV pin, do the following:

1. Set the bit ITMEN, SWV_ITM_CR[0], which enables the module.
2. Set bits in the enable register, SWV_ITM_TER, corresponding to any of the 32 stimulus registers SWV_ITM_SPR_DATA[0..31] to be used.
3. Write the data to an enabled stimulus register SWV_ITM_SPR_DATA[0..31]. All stimulus registers are 32 bits in size; data written to a stimulus register is loaded into a FIFO to be transmitted as a 4-byte Software Instrumentation Trace (SWIT) packet.

Other types of packets may also be sent by the SWV under firmware program control, such as time stamps and synchronization packets. For details on these packets see the ARM document *CoreSight™ Components Technical Reference Manual*.

39.4.1 SWV Protocols

Both Manchester and UART protocols operate over a single pin (SWV) and do not require separate clock or control pins. Every data packet is in 8-bit multiples (bytes) when either protocol is selected.

For a trace capture device to correctly interpret trace data from SWV, it must be able to decode where data exists on the various pin protocols. This section describes how protocols must be decoded to establish the underlying transmit data.

39.4.1.1 Manchester Encoding

In the Manchester protocol, the SWV outputs up to eight bytes of data between a start bit and a stop bit, as shown in Figure 39-4. Table 39-17 describes Manchester pin protocol encoding.

Figure 39-4. Manchester Encoded Data Sequence

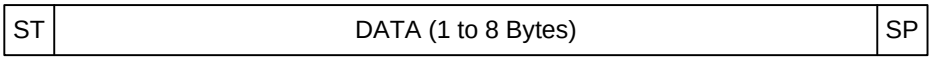
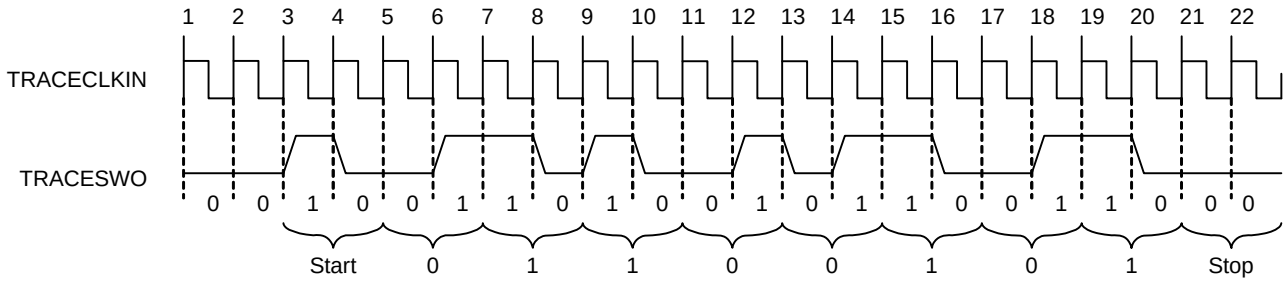


Table 39-17. Manchester Pin Protocol Encoding

Pin	Logic '0'	Logic '1'	Idle State (No Data)	Valid Data
TRACESWO	LOW-HIGH (01)	HIGH-LOW (10)	LOW (00)	Start bit: HIGH-LOW transition/Logic '1' between 1 and 8 bytes of data Stop bit: output LOW, not a valid Manchester symbol

Figure 39-5 shows how a sequence of bytes is transmitted using Manchester bit encoding.

Figure 39-5. Manchester Encoding Example



39.4.1.2 UART Encoding

In the UART protocol, data is sent out in packets of ten bits, with start and stop bits, as shown in Figure 39-6 and Table 39-18. Capture devices are expected to operate at the same clocking speed as the SWV pin and synchronize by waiting for a start bit.

Figure 39-6. UART Encoding Example

Table 39-18. UART Encoding

Pin	Logic '0'	Logic '1'	Idle State (No Data)	Valid Data
TRACESWO	LOW	HIGH	HIGH	10 bit sequence: Logic '0' 8 data bits Logic '1'

39.4.2 SWV Registers

SWV registers are as listed in Table 39-19.

Table 39-19. SWV Registers

Register	Size (Bits)	Description
SWV_SWO_CAOSD	32	Output speed divisor
SWV_SWO_SPP	32	Output protocol (Manchester or UART)
SWV_ITM_CR	32	ITM control
SWV_ITM_TER	32	Enable for each of the stimulus ports
SWV_ITM_SPRxx	32	Stimulus ports 0 to 31
SWV_ITM_SCR	32	Synchronization packet control

40. Nonvolatile Memory Programming

PSoC® 3 devices have three types of nonvolatile memory: flash, electronically erasable programmable read only memory (EEPROM), and nonvolatile latch (NVL). These can all be programmed by either the CPU running a boot loader program or by an external system via the JTAG/SWD interface. See PSoC 3 Device Programming Specifications for details about device programming and programming specifications.

40.1 Features

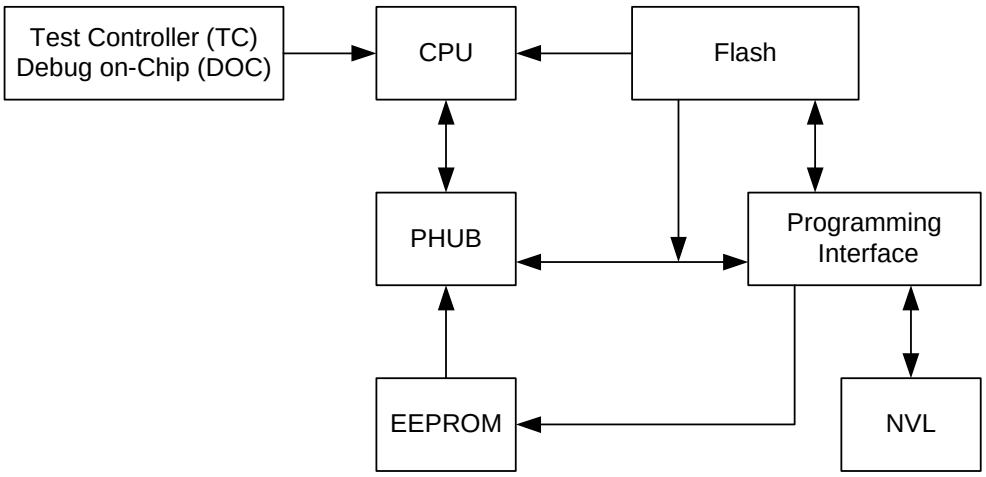
The nonvolatile memory programming system has the following features:

- Simple command/status register interface
- Flash can be programmed at the 288-byte row level
- Each row of flash has 256 bytes of data plus an additional 32 bytes for ECC/configuration
- EEPROM can be programmed at the 16-byte row level
- All configuration NVL bytes can be programmed simultaneously
- A single write once NVL byte can be programmed

40.2 Block Diagram

Figure 40-1 is a block diagram of the flash programming system.

Figure 40-1. Flash Block Diagram



40.3 How It Works

All programming operations are done through a simple command/status register interface summarized in Table 40-1.

Table 40-1. 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 was 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.

40.3.1 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 40-2. See 40.3.1.1 Command Code Descriptions on page 429 for details.

Table 40-2. 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
0x07	Program row	Flash, EEPROM	Any	Programs a row with data in row latch
0x08	Erase sector	Flash, EEPROM	Any	Erases a 64-row sector
0x09	Erase all	Flash	TC only	Erases all flash, including ECC and 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 JTAG/SWD interface and the test controller (see the Test Controller chapter on page 405).

Some commands require an array ID as a parameter. Array ID codes are shown in Table 40-3.

Table 40-3. 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
0x80	User NVL array
0xF8	Write Once NVL array

A flash array has, at most, 64 KB plus ECC bytes. PSoC 3 architecture has one flash array, the size of which is 16 KB, 32 KB, or 64 KB plus ECC bytes; therefore, the only valid array ID is 0x00.

An EEPROM array has, at most, 2 KB. PSoC 3 devices have one EEPROM array, the size of which is 512 bytes, 1 KB, or 2 KB.

PSoC 3 devices have one user NVL array and 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 40-4. Row Latch Sizes

Array Type	Size (Bytes)
Flash, with ECC Enabled	256
Flash, with ECC Disabled	288 (256 data bytes plus 32 configuration bytes)
EEPROM	16

40.3.1.1 Command Code Descriptions

The following are descriptions of the command codes listed in Table 40-2 on page 428.

- 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 40-4.

- 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 40-4.

- 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 ECC/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 ECC/configuration space is selected. For example, if the address is 0x80000B and N is 0x08, the command reads 9 ECC/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 ECC/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 ECC/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 ECC/configuration bytes), valid address ranges are:

  - Data space – 0x000000 – 0x00FFFF (64 KB)
  - ECC/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).

  For flash, data bytes and ECC/configuration bytes are both programmed. If ECC is enabled then the ECC syndrome bytes are automatically generated and loaded into the ECC/configuration bytes of the row latch before programming takes place.

  The die temperature parameters can be acquired by sending the Get Temperature command (see the Temperature Sensor chapter on page 371).

- 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.

- Command 0x07 – Program Row

  Command Parameter Bytes – Array ID, Row ID high, Row ID low

  This command programs the addressed flash/EEPROM row with the data in the row latch. If the row ID is greater than the array size (in rows), the row ID wraps (the upper bits are ignored).

  The row must have been previously erased (commands 0x08 and 0x09).

  For flash, data bytes and ECC/configuration bytes are both programmed. If ECC is enabled, the ECC syndrome bytes are automatically generated and loaded into the ECC/configuration bytes of the row latch before programming takes place.

- 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 associated ECC/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 ECC/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 40.3.3 Flash Protection Settings on page 431). 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.

- 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. If ECC is disabled, the computation includes all data from the user space and the ECC / configuration space. If ECC is enabled, the computation includes only data from the user space.

If the array ID is All Flash, the checksum computed includes all flash data on all flash arrays on the device. The rest of the command parameters are ignored. The checksum value is returned MS byte first.

40.3.1.2 Command Failure Codes

In response to commands, a success/failure code is returned in the SPC_SR register: These codes are described in Table 40-5.

Table 40-5. 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

40.3.2 Register Summary

All programming operations are done through a simple command/status register interface, shown in Table 40-1 on page 428.

40.3.3 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 40-2, 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.

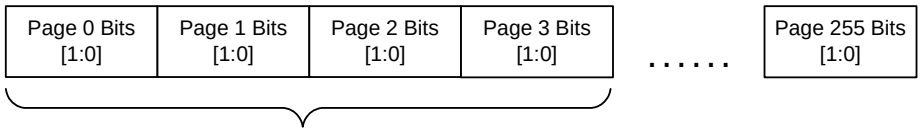
Table 40-6. Protection Modes

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 40-2. Flash Protection Bits



Byte 0 in Flash Hidden Row 0:
Contains protection data for Flash rows 0 through 3.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the PSoC device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.

AND See *Boolean Algebra*.

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

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

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

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

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

B

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

bandwidth

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

bias

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

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

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

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

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

block

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

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

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

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

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

buffer

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

bus

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

byte A digital storage unit consisting of 8 bits.

C

C	A high level programming language.
capacitance	A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.
capture	To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file.
chaining	Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another.
checksum	The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a pre-determined value.
clear	To force a bit/register to a value of logic ‘0’.
clock	The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks.
clock generator	A circuit that is used to generate a clock signal.
CMOS	The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor.
comparator	An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements.
compiler	A program that translates a high level language, such as C, into machine language.
configuration	In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance.
configuration space	In PSoC devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’.
crowbar	A type of over-voltage protection that rapidly places a low resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value.
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 non-packaged integrated circuit (IC), normally cut from a wafer.
digital	A signal or function, the amplitude of which is characterized by one of two discrete values: ‘0’ or ‘1’.
digital blocks	The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
digital logic	A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system.
digital-to-analog (DAC)	A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation.
direct access	The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data.
duty cycle	The relationship of a clock period *high time* to its *low time*, expressed as a percent.

E

emulator Duplicates (provides an emulation of) the functions of one system with a different system, so that the second system appears to behave like 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 on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux) 1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

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

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise 1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR See *Boolean Algebra*.

NOT See *Boolean Algebra*.

O

OR See *Boolean Algebra*.

oscillator A circuit that may be crystal controlled and is used to generate a clock frequency.

output The electrical signal or signals which are produced by an analog or digital block.

P

parallel The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.

parameter Characteristics for a given block that have either been characterized or may be defined by the designer.

parameter block A location in memory where parameters for the SSC instruction are placed prior to execution.

parity A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).

path 1. The logical sequence of instructions executed by a computer.
2. The flow of an electrical signal through a circuit.

pending interrupts	An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.
phase	The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).
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 is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.
Power On Reset (POR)	A circuit that forces the PSoC device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*.
program counter	The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.
protocol	A set of rules. Particularly the rules that govern networked communications.
PSoC®	Cypress's Programmable System-on-Chip (PSoC®) devices.
PSoC blocks	*See analog blocks* and *digital blocks*.
PSoC Creator™	The software for Cypress's next generation Programmable System-on-Chip technology.
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 measure.

R

RAM	An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.
register	A storage device with a specific capacity, such as a bit or byte.
reset	A means of bringing a system back to a know state. See *hardware reset* and *software reset*.
resistance	The resistance to the flow of electric current measured in ohms for a conductor.
revision ID	A unique identifier of the PSoC device.
ripple divider	An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to 2^n - 1.
rising edge	*See positive edge*.
ROM	An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.
routine	A block of code, called by another block of code, that may have some general or frequent use.
routing	Physically connecting objects in a design according to design rules set in the reference library.
runt pulses	In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling	The process of converting an analog signal into a series of digital values or reversed.
schematic	A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.
seed value	An initial value loaded into a linear feedback shift register or random number generator.
serial	1. Pertaining to a process in which all events occur one after the other. 2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.
set	To force a bit/register to a value of logic 1.
settling time	The time it takes for an output signal or value to stabilize after the input has changed from one value to another.

shift	The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.
shift register	A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.
sign bit	The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.
signal	A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.
silicon ID	A unique identifier of the PSoC silicon.
skew	The difference in arrival time of bits transmitted at the same time, in parallel transmission.
slave device	A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.
software	A set of computer programs, procedures, and associated documentation 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 is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.
SROM	An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash.
stack	A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.
stack pointer	A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.
state machine	The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.
sticky	A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The PSoC SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous 1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the PSoC device and reading this manual.

user modules Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital PSoC Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

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

24-bit data pointer registers
    8051 64
32.768 kHz crystal oscillator 127
50% duty cycle mode 132
8051 95
    24-bit data pointer registers 64
    active interrupt 94
    arithmetic instructions 40
    arithmetic logic unit 40
    bit addressing mode 39
    block diagram 37
    boolean instructions 43
    core enhancements 38
        DPTR extension SFR 38
        Dual DPTR 38
        vectored interrupt controller interface 38
    CPU halt mechanisms 66
    CY wrapper logic 39
    data transfer instructions 42
    direct addressing mode 39
    dual data pointer registers 63
    external data memory space 66
    features 37
    how it works 38, 379
    immediate constants mode 39
    indirect addressing mode 39
    instruction set 38, 39
    instruction set details 44
    internal data space 39
    interrupt controller interface 38
    interrupt enable register 65
    interrupt nesting 94
    interrupt vector addresses 95
    IO port access registers 65
    jump instructions 43
    logical instructions 41
    memory map 38
    program branching instructions 43
    program memory space 66
    PSoC 3 37
    register addressing mode 39
    register specific instructions mode 39
    special function registers 62
    wrapper logic 37

A

ACC (SFR 0xE0) register 40
active interrupt
    8051 94
active mode
    entering 146
    exiting 146
    low power modes 146
    PSoC 146
active mode boost converter 140
alternative active mode 147
    entering 147
    exiting 147
    low power modes 147
analog I/O 168
analog subsystem
    PSoC 27
analog subsystem components
    PSoC 27
analog system 311
    architecture 311
application diagram
    boost converter 139
arithmetic instructions
    8051 40
arithmetic logic unit
    8051 40
asynchronous clocks 134

B

B (SFR 0xF0) register 40
bit addressing mode
    8051 39
block diagram
    8051 37
    cache controller 67
    clock distribution system 129
    clock divider Implementation 131
    clock selection 267
    DMA controller 73
    external memory interface 111
    Flash programming system 107
    GPIO 160
    I/O drive mode 163
    internal low speed oscillator 125

internal main oscillator 125
interrupt controller 89
master clock mux 130
MHzECO 126
phase shifter 133
phase-locked loop 128
PHUB 73
port interrupt controller unit 172
RESET module 152
resync option 133
SIO 161
SRAM organization for the CY8C38 family 104
USB clock mux 130
USBIO 161
voltage monitoring 141
watchdog timer circuit 149
boolean instructions
8051 43
boost converter
application diagram 139
modes of operation 140
PSoC 139
status monitoring 140
bypassed clock source 133

C

capture mode
timing diagram 269
capture signal 268
central processing unit
cache controller 67
PHUB and DMA 73
clock
system 134
clock block
components 124
clock distribution 129
clock distribution module 131
clock divider 134
50% duty cycle mode 132
single cycle pulse mode 132
clock doubler 124
clock frequency
USB mode operations 131
clock generator
introduction 123
clock selection
block diagram 267
clock selection by timer block 266
clock signal naming 135
clock source
selection by timer block 266
clock sources
distribution 129
DSI 128
internal 124
phase-locked loop 128
clock synchronization 132
clock tree
bypassed clock source output 133
phase delayed clk_sync output 132
power gating 134
resynchronized clock output 132
types 129
unsynchronized divided clock output 133
clocks
asynchronous 134
high precision 123
low power mode operation 135
cock dividers
main part of clock distribution module 131
compare types in pulse width mode 276
core enhancements
8051 38
counter mode
register configuration 272
counters
features 265
CPU halt mechanisms
8051 66
CPU system 35
architecture 35
crystal oscillator
low power operation 127
CY wrapper logic
8051 39
CY8C38 family
Flash memory 108
PSoC 3 37
RAM implementation 105

D

data transfer instructions
8051 42
dead band feature 278
delay chain 133
development kits 29
device configuration of nonvolatile latch 99
digital I/O
control by DSI 166
digital system 181
architecture 182
digital-to-analog converter 27
direct addressing mode
8051 39
DMA controller 39
block diagram 73
document
glossary 433
overview 3
revision history 21
doubled clock 124

DPTR extension SFR
8051 core enhancements 38
DSI control of digital I/O 166
DSI input 167
DSI output to I/O port 166
dual data pointer registers
8051 63
Dual DPTR
8051 core enhancements 38

E

EEPROM
features 109
EEPROM memory
how erased 110
how it works 110
PSoC 109
EMIF
block diagram 111
external memory support 112
features 111
how it works 112
memory mapped peripherals 118
registers 112
sleep mode behavior 115
timing 116
EMIF register list 112
enable signal 268
enabling and disabling timer block 267
enabling interrupts
PSoC 90
erasing EEPROM memory 110
external crystal oscillators 126
external data memory space
8051 66
external memory interface
block diagram 111
connection to peripheral devices 118
features 111
how it works 112
introduction 111
register list 112
sleep mode 115
timing 116
external memory support in EMIF 112
external reset 152

F

fast start IMO (FIMO) 125
features
8051 37
counters 265
EEPROM 109
EMIF 111
Flash memory 107
I/O system 159
interrupt controller 89
low power modes 145
nonvolatile latch 99
PHUB 73
port interrupt controller unit 172
power supply and monitoring 137
pulse-width modulator 265
SRAM 103
timers 265
watchdog timer 149
flash interrupt
attempted flash write 108
ECC – multi-bit 108
ECC – single bit 108
Flash memory
CY8C38 family 108
features 107
how it works 108
PSoC 107
regions 108
free run mode
timing diagram 271
free run timer mode 271
frequency generation 123

G

gated timer in pulse width
timing diagram 273
general purpose input/output interface
PSoC 26
glossary 433
GPIO
block diagram 160
GPIO pins in creation of buttons and sliders 168

H

help, getting
development kits 29
hibernate mode 147
entering 147
exiting 148
low power modes 147
hibernate regulator 139
high impedance analog drive mode 163
high impedance digital drive mode 163
high voltage interrupt 141, 142
hot swap
SIO pin support 170
how it works
8051 . . . . . 38, 379
EEPROM memory 110
EMIF 112
external memory interface 112
Flash memory 108

I/O system 162
interrupt controller 90
PHUB 74
power regulators 139
PSoC RAM 105
watchdog timer 150

I

I/O drive mode 162
block diagram 163
high impedance analog 163
high impedance digital 163
open drain 164
resistive 163
resistive pull up and pull down 164
strong 164
I/O pins
low power mode behavior 172
overvoltage tolerance 171
I/O power supply 172
I/O system
analog I/O 168
CapSense 168
digital I/O controlled through DSI 166
DSI input 167
DSI output 166
features 159
how it works 162
I/O port reconfiguration 171
I/O power supply 172
introduction 159
LCD drive capabilities 168
low power mode effect on I/O pins 172
open drain modes 164
overvoltage tolerance 171
port interrupt controller unit 172
port interrupt controller unit pin configuration 173
port register of digital I/O 164
power up I/O configuration 171
register summary 174
resistive modes 163
resistive pull up and pull down mode 164
SFR to GPIO (PSoC3 only) 166
SIO functions 169
slew rate control 164
strong drive mode 164
usage modes and configuration 162
identifying reset sources 152
immediate constants mode
8051 39
imprecise power on reset 151
indirect addressing mode
8051 39
input signals of timer block 267
input voltage to boost converter 139
instruction set
8051 . . . . . 38, 39
instruction set details
8051 44
internal data space
8051 39
internal low speed oscillator 124, 125
internal main oscillator 124
internal regulators 139
interrupt controller
block diagram 89
features 89
how it works 90
interrupt execution 92
introduction 89
level interrupt 92
priority decoding unit 92
pulse interrupt 92
interrupt controller interface
8051 38
interrupt enable register
8051 65
interrupt execution
interrupt controller 92
interrupt nesting
8051 94
interrupt service routine 95
8051 95
sleep mode 95
interrupt vector addresses
8051 95
introduction
clock generator 123
I/O system 159
reset 151
timer 265
IO port access registers
8051 65

J

jump instructions
8051 43

K

kill signal 270

L

LCD drive
I/O system capabilities 168
level interrupt
interrupt controller 92
logic diagram
RESET module 152
logical instructions

8051 41
low power modes 145
active mode 146
alternative active mode 147
features 145
hibernate mode 147
power mode transitions diagram 145
sleep mode 147
timewheel 148
low voltage detect interrupt processing 142
low voltage interrupt 141

M

mapping diagram
digital system input to pad selection 166
master clock mux 130
memory 97
architecture 97
memory map
8051 38
memory mapped peripherals in EMIF 118
MHz crystal oscillator 126
modes
active mode in low power modes 146
power consumption-reducing 146

N

nonvolatile latch
device configuration 99
device configuration register map 99
features 99
programming 102
sleep mode 102
write once 100

O

on the fly duty cycle update 278
one shot mode 279
oscillators
external crystal 126
internal PSoC 124
summary table 128
overview, document 3, 21
getting started 29
revision history 21
overvoltage tolerance in I/O pins 171

P

peripheral device connection
external memory interface 118
phase delayed clk_sync 132
phase select and control 133
phase shifter 134
phase-locked loop 128
PHUB
block diagram 73
features 73
how it works 74
port interrupt controller
features 172
port interrupt controller unit 172
block diagram 172
pin configuration 173
port register control of digital I/O 164
power consumption-reducing modes 146
power gating
clock outputs 134
power mode transitions in low power modes 145
power on reset 151
imprecise 151
precision 151
power regulators
how it works 139
power supply and monitoring
features 137
precision power on reset 151
priority decoding unit
interrupt controller 92
product upgrades 29
program and debug 403
architecture 403
PSoC 27
program branching instructions
8051 43
program memory space
8051 66
Program Status Word SFR 40
PSoC
active mode 146
analog subsystem 27
analog subsystem components 27
boost converter 139
DMA controller 25
EEPROM memory 109
enabling interrupts 90
Flash memory on-chip 107
general purpose input/output interface 26
program and debug 27
special input/output interface 26
SRAM 103
supply pins 137
timer blocks 265
USB input/output interface 26
PSoC 3
8051 37
8051 features 94
CY8C38 family 37
major components 23
PSoC Ram
how it works 105

PSoC3
SFR to GPIO 166
PSW0xD0 register 40
pulse interrupt
interrupt controller 92
pulse width mode
comparator mode 276
compare types 276
pulse width modulator
dead band feature 278
on the fly duty cycle update 278
one shot mode 279
register configuration 276
pulse-width modulator
features 265

R

RAM implementation
CY8C38 family 105
real time clock 127
register addressing mode
8051 39
register map
nonvolatile latch 99
register specific instructions mode
8051 39
register summary
I/O system 174
registers
counter mode 272
EMIF 112
pulse width modulator 276
timer block 280
regulator
hibernate 139
internal 139
sleep 139
reset 151
external 152
identifying sources 152
introduction 151
software initiated 152
watchdog 151
RESET module
logic diagram 152
reset sources
description 151
reset summary table 154
resynchronized clock 132
RETI instruction 38
revision history 21

S

SFR – GPIO interface 39
single cycle pulse mode 132
SIO
block diagram 161
SIO functions
adjustable input level 169
hot swap 170
regulated output level 169
sleep mode
entering 147
exiting 147
external memory interface 115
gating off of clock network outputs 135
interrupt service routine 95
low power mode 147
low power modes 147
nonvolatile latch behavior 102
timer block behavior 280
sleep mode behavior in EMIF 115
sleep regulator 139
slew rate control in I/O system 164
software initiated reset 152
special function registers
8051 62
special input/output interface
PSoC 26
SRAM
features 103
PSoC 103
standby mode boost converter 140
status monitoring
boost converter 140
supply pins
PSoC 137
synchronization of clock 132
system clock operation 134
system integration
low power modes 145
reset 151
system wide resources 121
architecture 121

T

timer
introduction 265
timer block
enabling and disabling 267
input signals 267
modes 266
registers 280
selection of clock source 266
sleep mode behavior 280
timer blocks
PSoC 265
timer mode
free run 271
period mode 274
stop on interrupt mode 275

timer reset signal 270
timing diagram 270
timers
features 265
timewheel in low power modes 148
timing
EMIF 116
timing diagram
capture mode 269
free run mode 271
gated timer in IRQ mode 275
gated timer in period mode 274
gated timer in pulse width mode 273
timer reset signal 270
timing of EMIF 116
timing of external memory interface 116

U

unsynchronized divided clock 133
update 134
updating of clock divider 134
upgrades 29
usage modes and configuration of I/O system 162
USB clock 130
USB input/output interface
PSoC 26
USB mode operation
clock frequency 131
USBIO
block diagram 161

V

vectored interrupt controller interface
8051 core enhancements 38
voltage monitoring 141

W

watchdog reset 151
watchdog timer
clearing 150
disabling 150
enabling 150
features 149
how it works 150
operation in low power mode 150
protection settings 150
setting time period 150
wrapper logic
8051 37
write once latch 100