MAX32655 User Guide

UG7419; Rev 0; 03/2021

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32655 microcontroller. Detailed information for all registers and fields in the device are covered. Guidance is given for managing all the peripherals, clocks, power, and startup for the device family.

MAX32655 User Guide

Table of Contents

MAX32655 User Guide ---- 2
1. Overview ---- 18
1.1 Block Diagram ---- 19
2. Memory, Register Mapping, and Access ---- 20
2.1 Memory, Register Mapping, and Access Overview ---- 20
2.2 Field Access Definitions ---- 25
2.3 Standard Memory Regions ---- 25
2.4 Device Memory Instances ---- 28
2.5 AHB Interfaces ---- 29
2.5.1.1 I-Code ---- 29
2.5.1.2 D-Code ---- 29
2.5.1.3 System ---- 29
2.5.2.1 Standard DMA ---- 29
2.5.2.2 SPI0 ---- 29
2.5.2.3 AHB Slave Base Address Map ---- 30
2.6 Peripheral Register Map ---- 30
2.7 Error Correction Coding (ECC) Module ---- 31
3. System, Power, Clocks, Reset ---- 32
3.1 Oscillator Sources ---- 32
3.1.1.1 100MHz Internal Primary Oscillator (IPO) ---- 32
3.1.1.2 32MHz External RF Oscillator (ERFO) ---- 32
3.1.1.3 60MHz Internal Secondary Oscillator (ISO) ---- 34
3.1.1.4 7.3728MHz Internal Baud Rate Oscillator (IBRO) ---- 34
3.1.1.5 32.768kHz External Real-Time Clock Oscillator (ERTCO) ---- 34
3.1.1.6 8kHz-30kHz Internal Nano-Ring Oscillator (INRO) ---- 34
3.2 System Oscillator (SYS_OSC) ---- 35
3.3 Operating Modes ---- 38
3.3.2.1 Entering SLEEP ---- 38
3.3.3.1 Entering LPM ---- 40
3.3.4.1 Entering UPM ---- 42
3.3.5.1 Entering STANDBY ---- 44
3.3.6.1 Entering BACKUP ---- 46
3.3.7.1 Entering PDM ---- 48
3.4 Device Resets ---- 50
3.5 Unified Internal Cache Controllers (ICC) ---- 51
3.6 RAM Memory Management ---- 54
3.7 Miscellaneous Control Registers (MCR) ---- 54
3.8 Single Inductor Multiple Output Power Supply (SIMO) ---- 57

3.9 Low-Power General Control Registers (LPGCR) 64
3.10 Power Sequencer Registers (PWRSEQ) 66
3.11 Trim System Initialization Registers (TRIMSIR) 73
3.12 Global Control Registers (GCR) 75
3.13 Error Correction Coding (ECC) 91
3.14 System Initialization Registers (SIR) 91
3.15 Function Control Registers (FCR) 93
4. Interrupts and Exceptions 97
4.1 CM4 Interrupt and Exception Features 97
4.2 CM4 Interrupt Vector Table 97
4.3 RV32 Interrupt Vector Table 99
5. General-Purpose I/O and Alternate Function Pins (GPIO) 101
5.1 Instances 101
5.2 Configuration 102
5.3 Reference Tables 104
5.4 Usage 105
5.5 Configuring GPIO (External) Interrupts 106
5.6 Registers 108
6. Flash Controller (FLC) 117
6.1 Instances 117
6.2 Usage 117
6.3 Registers 119
7. Debug Access Port (DAP) 125
7.1 Instances 125
7.2 Access Control 125
7.3 Pin Configuration 125
8. Semaphores 126
8.1 Instances 126
8.2 Multiprocessor Communications 126
8.3 Registers 127
9. Standard DMA (DMA) 132
9.1 Instances 132
9.2 DMA Channel Operation (DMA_CH) 132
9.3 Usage 136
9.4 Count-To-Zero (CTZ) Condition 136
9.5 Chaining Buffers 137
9.6 DMA Interrupts 139

9.7 Channel Timeout Detect ---- 139
9.8 Memory-to-Memory DMA ---- 140
9.9 DMA Registers ---- 140
9.10 DMA Channel Registers ---- 141
10. Analog-to-Digital Converter (ADC) and Comparators ---- 147
10.1 Features ---- 147
10.2 Instances ---- 147
10.3 Architecture ---- 148
10.4 Clock Configuration ---- 149
10.5 Power-Up Sequence ---- 149
10.6 Conversion ---- 150
10.7 Reference Scaling and Input Scaling ---- 151
10.8 Data Limits and Out of Range Interrupts ---- 152
10.9 Power-Down Sequence ---- 154
10.10 Comparator Operation ---- 154
10.11 ADC Registers ---- 155
10.12 Low-Power Comparator Registers ---- 159
11. UART (UART) ---- 161
11.1 Instances ---- 162
11.2 DMA ---- 163
11.3 UART Frame ---- 163
11.4 FIFOs ---- 163
11.5 Interrupt Events ---- 164
11.6 LPUART Wakeup Events ---- 166
11.7 Inactive State ---- 166
11.8 Receive Sampling ---- 167
11.9 Baud Rate Generation ---- 167
11.9.4.1 Configurating a LPUART for Low-Power Modes of Operation ---- 169
11.10 Hardware Flow Control ---- 169
11.10.2.1 RTC/CTS Handling for Application Controlled HFC ---- 171
11.11 Registers ---- 172
12. Serial Peripheral Interface (SPI) ---- 179
12.1 Instances ---- 180
12.2 Formats ---- 181
12.3 Pin Configuration ---- 183
12.4 Clock Configuration ---- 184
12.5 Registers ---- 187
13. I²C Master/Slave Serial Communications Peripheral (I2C) ---- 198

13.1 I²C Master/Slave Features ---- 198
13.2 Instances ---- 198
13.3 I²C Overview ---- 199
13.4 Configuration and Usage ---- 201
13.4.4.1 Hs-Mode Timing ---- 202
13.4.4.2 Hs-Mode Clock Configuration ---- 202
13.4.6.1 I²C Master Mode Receiver Operation ---- 205
13.4.6.2 I²C Master Mode Transmitter Operation ---- 205
13.4.6.3 I²C Multi-Master Operation ---- 205
13.4.7.1 Slave Transmitter ---- 207
13.4.7.2 Slave Receivers ---- 209
13.5 Registers ---- 215
14. Inter-Integrated Sound Interface (I²S) ---- 229
14.1 Instances ---- 229
14.2 Details ---- 230
14.3 Master and Slave Mode Configuration ---- 231
14.4 Clocking ---- 231
14.5 Data Formatting ---- 232
14.6 Transmit and Receive FIFOs ---- 235
14.7 Interrupt Events ---- 238
14.8 Direct Memory Access ---- 239
14.9 Block Operation ---- 239
14.10 Registers ---- 239
15. 1-Wire Master (OWM) ---- 245
15.1 1-Wire Master Features ---- 245
15.2 1-Wire Pins and Configuration ---- 246
15.3 1-Wire Protocol ---- 246
15.3.1.1 Physical Layer ---- 247
15.3.1.2 Link Layer ---- 247
15.3.1.3 Network Layer ---- 250
15.4 1-Wire Operation ---- 253
15.5 1-Wire Data Reads ---- 254
15.6 Registers ---- 255
16. Real-Time Clock (RTC) ---- 259
16.1 Overview ---- 259
16.2 Instances ---- 260
16.3 Register Access Control ---- 260
16.4 RTC Alarm Functions ---- 261
16.5 RTC Calibration ---- 263
16.6 Registers ---- 265

17. Timers (TMR/LPTMR) ---- 269
17.1 Instances ---- 270
17.2 Basic Timer Operation ---- 270
17.3 32-Bit Single / 32-Bit Cascade / Dual 16-Bit ---- 271
17.4 Timer Clock Sources ---- 271
17.5 Timer Pin Functionality ---- 272
17.6 Wakeup Events ---- 274
17.7 Operating Modes ---- 275
17.7.5.1 Capture Event ---- 284
17.7.5.2 Rollover Event ---- 285
17.8 Registers ---- 292
18. Wakeup Timer (WUT) ---- 300
18.1 Basic Operation ---- 300
18.2 One-Shot Mode (0) ---- 301
18.3 Continuous Mode (1) ---- 302
18.4 Registers ---- 303
19. Watchdog Timer (WDT) ---- 306
19.1 Instances ---- 307
19.2 Usage ---- 307
19.3 WDT Feed Sequence ---- 308
19.4 WDT Events ---- 309
19.5 Initializing the WDT ---- 311
19.6 Resets ---- 312
19.7 Registers ---- 312
20. Pulse Train Engine (PT) ---- 317
20.1 Instances ---- 317
20.2 Features ---- 317
20.3 Engine ---- 317
20.3.1.1 Pulse Train Mode ---- 317
20.3.1.2 In Pulse Train Mode, Set the Bit Pattern ---- 318
20.3.1.3 Synchronize Two or More Outputs, if Needed ---- 318
20.3.1.4 Pulse Train Loop Mode ---- 318
20.3.1.5 Pulse Train Loop Delay ---- 318
20.3.1.6 Pulse Train Automatic Restart Mode ---- 318
20.4 Enabling and Disabling a Pulse Train Output ---- 319
20.5 Atomic Pulse Train Output Enable and Disable ---- 319
20.6 Halt and Disable ---- 320
20.7 Interrupts ---- 320
20.8 Registers ---- 320

20.8.1.1 Pulse Train Engine Global Enable/Disable Register ---- 321
20.8.1.2 Pulse Train Engine Safe Enable Register ---- 323
20.8.1.3 Pulse Train Engine Safe Disable Register ---- 324
20.8.1.4 Pulse Train Registers ---- 324
21. CRC ---- 327
21.1 Instances ---- 327
21.2 Usage ---- 327
21.3 Polynomial Generation ---- 328
21.4 Calculations Using Firmware Writes to FIFO ---- 328
21.5 Calculations Using DMA ---- 329
21.6 Registers ---- 330
22. AES ---- 332
22.1 Instances ---- 332
22.2 Encryption of 128-Bit Blocks of Data Using FIFO ---- 332
22.3 Encryption of 128-Bit Blocks Using DMA ---- 332
22.4 Encryption of Blocks Less Than 128-Bits ---- 334
22.5 Decryption ---- 334
22.6 Interrupt Events ---- 334
22.7 Registers ---- 335
23. TRNG Engine ---- 338
23.1 TRNG Registers ---- 338
24. Bluetooth 5 Low Energy (LE) Radio ---- 340
24.1 Power-Efficient Design ---- 340
24.2 Bluetooth Hardware Accelerator ---- 340
24.3 Arm Cordio®-B50 Software Stack ---- 340
24.4 Configuration ---- 342
24.5 Documentation ---- 342
25. Secure Boot ---- 343
25.1 Development Tools ---- 343
25.2 Secure Boot ---- 343
25.3 Building the Application Image ---- 344
25.4 Root Key Management ---- 345
25.5 Program Loading ---- 346
26. Revision History ---- 347

List of Figures

Figure 1-1: MAX32655 Block Diagram 19
Figure 2-1: CM4 Code Memory Mapping 21
Figure 2-2: RISC-V IBUS Code Memory Mapping 22
Figure 2-3: CM4 Peripheral and Data Memory Mapping 23
Figure 2-4: RV32 DBUS Peripheral and Data Memory Mapping 24
Figure 2-5: Unique Serial Number Format 26
Figure 3-1: Example 32MHz Crystal Capacitor Determination 33
Figure 3-2: MAX32655 Clock Block Diagram 37
Figure 3-3: SLEEP Mode Clock Control 39
Figure 3-4: Low-Power Mode (LPM) Clock and State Retention Diagram 41
Figure 3-5: Micro Power Mode (UPM) Clock and State Retention Block Diagram 43
Figure 3-6: STANDBY Mode Clock and State Retention Block Diagram 45
Figure 3-7: BACKUP Mode Clock and State Retention Block Diagram 47
Figure 3-8: Power Down Mode (PDM) Clock and State Retention Block Diagram 49
Figure 9-1: DMA Block-Chaining Flowchart 138
Figure 10-1: Analog-to-Digital Converter Block Diagram 148
Figure 10-2: ADC Limit Engine 153
Figure 11-1: UART Block Diagram 162
Figure 11-2: UART Frame Structure 163
Figure 11-3: UART Interrupt Functional Diagram 164
Figure 11-4: Oversampling Example 167
Figure 11-5: UART Baud Rate Generation 167
Figure 11-6: LPUART Timing Generation 168
Figure 11-7: HFC Physical Connection 170
Figure 11-8: HFC Signaling for Transmitting to an External Receiver 171
Figure 12-1: SPI Block Diagram 180
Figure 12-2: 4-Wire SPI Connection Diagram 182
Figure 12-3: Generic 3-Wire SPI Master to Slave Connection 183
Figure 12-4: Dual Mode SPI Connection Diagram 184
Figure 12-5: SCK Clock Rate Control 185
Figure 12-6: SPI Clock Polarity 186
Figure 13-1: I^2C Write Data Transfer 200
Figure 13-2: I^2C SCL Timing for Standard, Fast and Fast-Plus Modes 201
Figure 14-1: I^2S Master Mode 230
Figure 14-2: I^2S Slave Mode 230
Figure 14-3:Audio Interface I^2S Signal Diagram 231
Figure 14-4: Audio Mode with Inverted Word Select Polarity 233
Figure 14-5: Audio Master Mode Left-Justified First Bit Location 233
Figure 14-6: MSB Adjustment when Sample Size is Less Than Bits Per Word 234
Figure 14-7: LSB Adjustment when Sample Size is Less Than Bits Per Word 234
Figure 14-8: I^2S Mono Left Mode 235
Figure 14-9: I^2S Mono Right Mode 235
Figure 15-1: 1-Wire Signal Interface 247
Figure 15-2: 1-Wire Reset Pulse 248
Figure 15-3: 1-Wire Write Time Slot 249
Figure 15-4: 1-Wire Read Time Slot 249
Figure 15-5: 1-Wire ROM ID Fields 250
Figure 16-1: MAX32655 RTC Block Diagram (12-bit Sub-Second Counter) 259

Figure 16-2: RTC Interrupt/Wakeup Diagram Wakeup Function....262
Figure 16-3: Internal Implementation of 4kHz Digital Trim....264
Figure 17-1: MAX32655 TimerA Output Functionality, Modes 0/1/3/5....273
Figure 17-2: MAX32655 TimerA Input Functionality, Modes 2/4/6/7/8/14....274
Figure 17-3: Timer I/O Signal Naming Conventions....275
Figure 17-4: One-Shot Mode Diagram....278
Figure 17-5: Continuous Mode Diagram....280
Figure 17-6: Counter Mode Diagram....282
Figure 17-7: Capture Mode Diagram....285
Figure 17-8: Compare Mode Diagram....287
Figure 17-9: Gated Mode Diagram....289
Figure 17-10: Capture/Compare Mode Diagram....291
Figure 18-1: One-Shot Mode Diagram....301
Figure 18-2: Continuous Mode Diagram....302
Figure 19-1: Windowed Watchdog Timer Block Diagram....307
Figure 19-2: WDT Early Interrupt and Reset Event Sequencing Details....310
Figure 19-3: WDT Late Interrupt and Reset Event Sequencing Details....311
Figure 24-1: MAX32655 Bluetooth Stack Overview....341
Figure 25-1: MAX32655 Secure Boot Flow....344
Figure 25-2: MAX32655 Application Image....345

List of Tables

Table 2-1: Field Access Definitions 25
Table 2-2: System SRAM Configuration 27
Table 2-3: AHB Slave Base Address Map 30
Table 2-4: APB Peripheral Base Address Map 30
Table 3-1: Oscillators, Descriptions, and Nominal Frequencies 35
Table 3-2: Reset Sources and Effect on Oscillator Status 36
Table 3-3: Reset Sources and Effect on System Oscillator Selection and Prescaler 36
Table 3-4 System RAM Retention in BACKUP Mode 46
Table 3-5: Wakeup Sources for Each Operating Mode in the MAX32655 50
Table 3-6: Reset and Low-Power Mode Effects 50
Table 3-7: Internal Cache Controller Register Summary 52
Table 3-8: ICC0 Cache Information Register 53
Table 3-9: ICC0 Memory Size Register 53
Table 3-10: ICC0 Cache Control Register 53
Table 3-11: ICC0 Invalidate Register 53
Table 3-12 RAM Block Size and Base Address 54
Table 3-13: Miscellaneous Control Register Summary 54
Table 3-14: Error Correction Coding Enable Register 55
Table 3-15: IPO Manual Register 55
Table 3-16: Output Enable Register 55
Table 3-17: Comparator 0 Control Register 55
Table 3-18: Miscellaneous Control Register 56
Table 3-19: GPIO3 Pin Control Register 57
Table 3-20: SIMO Power Supply Device Pin Connectivity 58
Table 3-21: SIMO Controller Register Summary 58
Table 3-22: SIMO Buck Voltage Regulator A Control Register 59
Table 3-23: SIMO Buck Voltage Regulator B Control Register 59
Table 3-24: SIMO Buck Voltage Regulator C Control Register 60
Table 3-25: SIMO Buck Voltage Regulator D Control Register 60
Table 3-26: SIMO High Side FET Peak Current V_{REGO_A} V_{REGO_B} Register 61
Table 3-27: SIMO High Side FET Peak Current V_{REGO_C} V_{REGO_D} Register 61
Table 3-28: SIMO Maximum High Side FET Time On Register 61
Table 3-29: SIMO Buck Cycle Count V_{REGO_A} Register 62
Table 3-30: SIMO Buck Cycle Count V_{REGO_B} Register 62
Table 3-31: SIMO Buck Cycle Count V_{REGO_C} Register 62
Table 3-32: SIMO Buck Cycle Count V_{REGO_D} Register 62
Table 3-33: SIMO Buck Cycle Count Alert V_{REGO_A} Register 62
Table 3-34: SIMO Buck Cycle Count Alert V_{REGO_B} Register 62
Table 3-35: SIMO Buck Cycle Count Alert V_{REGO_C} Register 63
Table 3-36: SIMO Buck Cycle Count Alert V_{REGO_D} Register 63
Table 3-37: SIMO Buck Regulator Output Ready Register 63
Table 3-38: SIMO Zero Cross Calibration VREGO_A Register 63
Table 3-39: SIMO Zero Cross Calibration VREGO_B Register 64
Table 3-40: SIMO Zero Cross Calibration VREGO_C Register 64
Table 3-41: SIMO Zero Cross Calibration VREGO_D Register 64
Table 3-42 Low-Power Control Register Summary 64
Table 3-43: Reset Control Register 64
Table 3-44: Clock Disable Register 65
Table 3-45: Power Sequencer Register Summary 66
Table 3-46: Low-Power Control Register 67

Table 3-47: GPIO0 Low-Power Wakeup Status Flags 68
Table 3-48: GPIO0 Low-Power Wakeup Enable Registers 68
Table 3-49: GPIO1 Low-Power Wakeup Status Flags 68
Table 3-50: GPIO1 Low-Power Wakeup Enable Registers 69
Table 3-51: GPIO2 Low-Power Wakeup Status Flags 69
Table 3-52: GPIO2 Low-Power Wakeup Enable Registers 69
Table 3-53: GPIO3 Low-Power Wakeup Status Flags 69
Table 3-54: GPIO3 Low-Power Wakeup Enable Registers 70
Table 3-55: Low-Power Peripheral Wakeup Status Flags 70
Table 3-56: Low-Power Peripheral Wakeup Enable Registers 70
Table 3-57: Low-Power VBTLE Power Down Register 72
Table 3-58: Low-Power General Purpose Register 0 72
Table 3-59: Low-Power General Purpose Register 1 73
Table 3-60: Trim System Initialization Register Summary 73
Table 3-61: RTC Trim System Initialization Register 73
Table 3-62: SIMO Trim System Initialization Register 73
Table 3-63: IPO Low Trim System Initialization Register 74
Table 3-64: Control Trim System Initialization Register 74
Table 3-65: INRO Trim System Initialization Register 74
Table 3-66: Global Control Register Summary 75
Table 3-67: System Control Register 76
Table 3-68: Reset Register 0 77
Table 3-69: Clock Control Register 78
Table 3-70: Power Management Register 80
Table 3-71: Peripheral Clock Divisor Register 81
Table 3-72: Peripheral Clock Disable Register 0 81
Table 3-73: Memory Clock Control Register 83
Table 3-74: Memory Zeroize Control Register 83
Table 3-75: System Status Flag Register 84
Table 3-76: Reset Register 1 84
Table 3-77: Peripheral Clock Disable Register 1 85
Table 3-78: Event Enable Register 87
Table 3-79: Revision Register 87
Table 3-80: System Status Interrupt Enable Register 87
Table 3-81: Error Correction Coding Error Register 88
Table 3-82: Error Correction Coding Correctable Error Detected Register 88
Table 3-83: Error Correction Coding Interrupt Enable Register 88
Table 3-84: Error Correction Coding Error Address Register 88
Table 3-85: Bluetooth LDO Control Register 89
Table 3-86: Bluetooth LDO Delay Count Register 90
Table 3-87: General Purpose Register 91
Table 3-88: Error Correction Coding Enable Register Summary 91
Table 3-89: Error Correction Coding Enable Register 91
Table 3-90: System Initialization Register Summary 91
Table 3-91: System Initialization Status Register 92
Table 3-92: System Initialization Address Error Register 92
Table 3-93: System Initialization Function Status Register 92
Table 3-94: System Initialization Security Function Status Register 93
Table 3-95: Function Control Register Summary 93
Table 3-96: Function Control 0 Register 93
Table 3-97: Automatic Calibration 0 Register 94
Table 3-98: Automatic Calibration 1 Register 95
Table 3-99: Automatic Calibration 2 Register 95

Table 3-100: RV32 Boot Address Register 95
Table 3-101: RV32 Control Register 95
Table 3-102: ERFO Kick Start Register 96
Table 4-1: MAX32655 CM4 Interrupt Vector Table 97
Table 4-2: MAX32655 RV32 Interrupt Vector Table 99
Table 5-1: MAX32655 GPIO Pin Count 102
Table 5-2: MAX32655 GPIO Pin Function Configuration 103
Table 5-3: MAX32655 Input Mode Configuration 103
Table 5-4: MAX32655 Output Mode Configuration 103
Table 5-5: MAX32655 GPIO Alternate Function Configuration Reference 104
Table 5-6: MAX32655 GPIO Output/Input Configuration Reference 104
Table 5-7: MAX32655 GPIO Interrupt Configuration Reference 104
Table 5-8: MAX32655 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration Reference 105
Table 5-9: MAX32655 GPIO Port Interrupt Vector Mapping 106
Table 5-10: MAX32655 GPIO Wakeup Interrupt Vector 107
Table 5-11: GPIO Register Summary 108
Table 5-12: GPIO Port n Configuration Enable Bit 0 Register 109
Table 5-13: GPIO Port n Configuration Enable Atomic Set Bit 0 Register 109
Table 5-14: GPIO Port n Configuration Enable Atomic Clear Bit 0 Register 109
Table 5-15: GPIO Port n Output Enable Register 109
Table 5-16: GPIO Port n Output Enable Atomic Set Register 110
Table 5-17: GPIO Port n Output Enable Atomic Clear Register 110
Table 5-18: GPIO Port n Output Register 110
Table 5-19: GPIO Port n Output Atomic Set Register 110
Table 5-20: GPIO Port n Output Atomic Clear Register 110
Table 5-21: GPIO Port n Input Register 111
Table 5-22: GPIO Port n Interrupt Mode Register 111
Table 5-23: GPIO Port n Interrupt Polarity Register 111
Table 5-24: GPIO Port n Input Enable Register 111
Table 5-25: GPIO Port n Interrupt Enable Register 112
Table 5-26: GPIO Port n Interrupt Enable Atomic Set Register 112
Table 5-27: GPIO Port n Interrupt Enable Atomic Clear Register 112
Table 5-28: GPIO Port n Interrupt Status Register 112
Table 5-29: GPIO Port n Interrupt Clear Register 112
Table 5-30: GPIO Port n Wakeup Enable Register 113
Table 5-31: GPIO Port n Wakeup Enable Atomic Set Register 113
Table 5-32: GPIO Port n Wakeup Enable Atomic Clear Register 113
Table 5-33: GPIO Port n Interrupt Dual Edge Mode Register 113
Table 5-34: GPIO Port n Pad Configuration 1 Register 113
Table 5-35: GPIO Port n Pad Configuration 2 Register 113
Table 5-36: GPIO Port n Configuration Enable Bit 1 Register 114
Table 5-37: GPIO Port n Configuration Enable Atomic Set Bit 1 Register 114
Table 5-38: GPIO Port n Configuration Enable Atomic Clear Bit 1 Register 114
Table 5-39: GPIO Port n Configuration Enable Bit 2 Register 114
Table 5-40: GPIO Port n Configuration Enable Atomic Set Bit 2 Register 115
Table 5-41: GPIO Port n Configuration Enable Atomic Clear Bit 2 Register 115
Table 5-42: GPIO Port n Hysteresis Enable Register 115
Table 5-43: GPIO Port n Output Drive Strength Bit 0 Register 115
Table 5-44: GPIO Port n Output Drive Strength Bit 0 Register 115
Table 5-45: GPIO Port n Output Drive Strength Bit 1 Register 115
Table 5-46: GPIO Port n Pulldown/Pullup Strength Select Register 115
Table 5-47: GPIO Port n Voltage Select Register 116
Table 6-1: MAX32655 Internal Flash Memory Organization 117

Table 6-2: Valid Addresses Flash Writes 118
Table 6-3: Flash Controller Register Summary 119
Table 6-4: Flash Controller Address Pointer Register 120
Table 6-5: Flash Controller Clock Divisor Register 120
Table 6-6: Flash Controller Control Register 120
Table 6-7: Flash Controller Interrupt Register 121
Table 6-8: Flash Controller Data 0 Register 122
Table 6-9: Flash Controller Data Register 1 122
Table 6-10: Flash Controller Data Register 2 122
Table 6-11: Flash Controller Data Register 3 122
Table 6-12: Flash Controller Access Control Register 123
Table 6-13: Flash Write/Lock 0 Register 123
Table 6-14: Flash Write/Lock 1 Register 123
Table 6-15: Flash Read Lock 0 Register 123
Table 6-16: Flash Read Lock 1 Register 124
Table 7-1: MAX32655 DAP Instances 125
Table 8-1: MAX32655 Semaphore Instances 126
Table 8-2: Semaphore Register Summary 127
Table 8-3: Semaphore 0 Register 127
Table 8-4: Semaphore 1 Register 127
Table 8-5: Semaphore 2 Register 128
Table 8-6: Semaphore 3 Register 128
Table 8-7: Semaphore 4 Register 128
Table 8-8: Semaphore 5 Register 128
Table 8-9: Semaphore 6 Register 128
Table 8-10: Semaphore 7 Register 129
Table 8-11: Semaphore Interrupt 0 Register 129
Table 8-12: Semaphore Mailbox 0 Register 129
Table 8-13: Semaphore Interrupt 1 Register 130
Table 8-14: Semaphore Mailbox 1 Register 130
Table 8-15: Semaphore Status Register 130
Table 9-1: MAX32655 DMA and Channel Instances 132
Table 9-2: DMA Source and Destination by Peripheral 133
Table 9-3: Data Movement from Source to DMA FIFO 134
Table 9-4: Data Movement from the DMA FIFO to Destination 135
Table 9-5: DMA Channel Timeout Configuration 139
Table 9-6: DMA Register Summary 140
Table 9-7: DMA Interrupt Flag Register 140
Table 9-8: DMA Interrupt Enable Register 141
Table 9-9: Standard DMA Channel 0 to Channel 3 Register Summary 141
Table 9-10: DMA Channel Registers Summary 141
Table 9-11: DMA_CHn Control Register 142
Table 9-12: DMA Status Register 144
Table 9-13: DMA Channel n Source Register 145
Table 9-14: DMA Channel n Destination Register 145
Table 9-15: DMA Channel n Count Register 145
Table 9-16: DMA Channel n Source Reload Register 145
Table 9-17: DMA Channel n Destination Reload Register 146
Table 9-18: DMA Channel n Count Reload Register 146
Table 10-1: MAX32655 ADC Input Pins for the 81-CTBGA Package 147
Table 10-2: MAX32655 ADC Clock Frequency and ADC Conversion Time with the System Clock set to the IPO 149
Table 10-3: ADC Data Register Alignment Options 150
Table 10-4: MAX32655 Input and Reference Scale Support by ADC Input Channel 151

Table 10-5: ADC Registers Summary 155
Table 10-6: ADC Control Register 155
Table 10-7: ADC Status Register 157
Table 10-8: ADC Data Register 157
Table 10-9: ADC Interrupt Control Register 157
Table 10-10: ADC Limit 0 to 3 Registers 158
Table 10-11: Low-Power Comparator Registers Summary 159
Table 10-12: Low-Power Comparator n Registers 160
Table 11-1: MAX32655 UART/LPUART Instances 162
Table 11-2: MAX32655 Interrupt Events 164
Table 11-3: Frame Error Detection for Standard UARTs and LPUART 165
Table 11-4: Frame Error Detection for LPUARTs with UARTn_CTRL.fdm = 1 and UARTn_CTRL.dpfe_en = 1 165
Table 11-5: MAX32655 Wakeup Events 166
Table 11-6: LPUART Low Baud Rate Generation Examples (UARTn_CTRL.fdm = 1) 169
Table 11-7: UART/LPUART Register Summary 172
Table 11-8: UART Control Register 172
Table 11-9: UART Status Register 174
Table 11-10: UART Interrupt Enable Register 175
Table 11-11: UART Interrupt Flag Register 175
Table 11-12: UART Clock Divisor Register 176
Table 11-13: UART Oversampling Control Register 176
Table 11-14: UART Transmit FIFO Register 176
Table 11-15: UART Pin Control Register 176
Table 11-16: UART Data Register 177
Table 11-17: UART DMA Register 177
Table 11-18: UART Wakeup Enable 178
Table 11-19. UART Wakeup Flag Register 178
Table 12-1: MAX32655 SPI Instances 180
Table 12-2: MAX32655 SPI Peripheral Pins 181
Table 12-3: Four-Wire Format Signals 181
Table 12-4: Three-Wire Format Signals 182
Table 12-5: SPI Modes Clock Phase and Polarity Operation 186
Table 12-6: SPIn Register Summary 187
Table 12-7: SPI FIFO32 Register 188
Table 12-8: SPI 16-bit FIFO Register 188
Table 12-9: SPI 8-bit FIFO Register 188
Table 12-10: SPI Control 0 Register 188
Table 12-11: SPI Control 1 Register 190
Table 12-12: SPI Control 2 Register 190
Table 12-13: SPI Slave Select Timing Register 191
Table 12-14: SPI Master Clock Configuration Registers 192
Table 12-15: SPI DMA Control Registers 193
Table 12-16: SPI Interrupt Status Flags Registers 194
Table 12-17: SPI Interrupt Enable Registers 195
Table 12-18: SPI Wakeup Status Flags Registers 196
Table 12-19: SPI Wakeup Enable Registers 196
Table 12-20: SPI Slave Select Timing Registers 197
Table 13-1: MAX78000 I^2C Peripheral Pins 198
Table 13-2: I^2C Bus Terminology 199
Table 13-3: Calculated I^2C Bus Clock Frequencies 202
Table 13-4: I^2C Slave Address Format 203
Table 13-5: Register Summary 215
Table 13-6: I^2C Control Register 216

Table 13-7: I²C Status Register 217
Table 13-8: I²C Interrupt Flag 0 Register 218
Table 13-9: I²C Interrupt Enable 0 Register 220
Table 13-10: I²C Interrupt Flag 1 Register 221
Table 13-11: I²C Interrupt Enable 1 Register 221
Table 13-12: I²C FIFO Length Register 222
Table 13-13: I²C Receive Control 0 Register 222
Table 13-14: I²C Receive Control 1 Register 222
Table 13-15: I²C Transmit Control 0 Register 223
Table 13-16: I²C Transmit Control 1 Register 225
Table 13-17: I²C Data Register 225
Table 13-18: I²C Master Control Register 225
Table 13-19: I²C SCL Low Control Register 226
Table 13-20: I²C SCL High Control Register 226
Table 13-21: I²C Hs-Mode Clock Control Register 226
Table 13-22: I²C Timeout Register 227
Table 13-23: I²C DMA Register 227
Table 13-24: I²C Slave Address Register 227
Table 14-1: MAX32655 I²S Instances 229
Table 14-2: MAX32655 I²S Pin Mapping 229
Table 14-3: I²S Mode Configuration 231
Table 14-4: Data Ordering for Byte Data Size (Stereo Mode) 236
Table 14-5: Data Ordering for Half-Word Data Size (Stereo Mode) 236
Table 14-6: Data Ordering for Word Data Size (Stereo Mode) 236
Table 14-7: Configuration for Typical Audio Width and Samples per WS Clock Cycle 237
Table 14-8. I²S Interrupt Events 238
Table 14-9: I²S Register Summary 239
Table 14-10: I²S Control 0 Register 240
Table 14-11: I²S Master Mode Configuration Register 241
Table 14-12: I²S DMA Control Register 242
Table 14-13: I²S FIFO Register 243
Table 14-14: I²S Interrupt Flag Register 243
Table 14-15: I²S Interrupt Enable Register 243
Table 15-1: MAX32655 1-Wire Master Peripheral Pins 246
Table 15-2: 1-Wire ROM Commands 250
Table 15-3: 1-Wire Slave Device ROM ID Field 251
Table 15-4: OWM Register Summary 255
Table 15-5: OWM Configuration Register 255
Table 15-6: OWM Clock Divisor Register 256
Table 15-7: OWM Control Status Register 256
Table 15-8: OWM Data Buffer Register 257
Table 15-9: OWM Interrupt Flag Register 257
Table 15-10: OWM Interrupt Enable Register 258
Table 16-1: RTC Seconds, Sub-Seconds, Time-of-Day Alarm and Sub-Seconds Alarm Register Details 260
Table 16-2: RTC Register Access 260
Table 16-3: MAX32655 RTC Square Wave Output Configuration 263
Table 16-4: RTC Register Summary 265
Table 16-5: RTC Seconds Counter Register 265
Table 16-6: RTC Sub-Second Counter Register (12-bit) 265
Table 16-7: RTC Time-of-Day Alarm Register 265
Table 16-8: RTC Sub-Second Alarm Register 266
Table 16-9: RTC Control Register 266
Table 16-10: RTC 32KHz Oscillator Digital Trim Register 268

Table 16-11: RTC 32KHz Oscillator Control Register 268
Table 17-1: MAX32655 TMR/LPTMR Instances 270
Table 17-2: MAX32655 TMR/LPTMR Instances Capture Events 270
Table 17-3: TimerA/TimerB 32-Bit Field Allocations 271
Table 17-4: MAX32655 Wakeup Events 274
Table 17-5: MAX32655 Operating Mode Signals for Timer 0 and Timer 1 275
Table 17-6: MAX32655 Operating Mode Signals for Timer 2 and Timer 3 276
Table 17-7: MAX32655 Operating Mode Signals for Low-Power Timer 0 and Low-Power Timer 1 276
Table 17-8: Timer Register Summary 292
Table 17-9: Timer Count Register 292
Table 17-10: Timer Compare Register 293
Table 17-11: Timer PWM Register 293
Table 17-12: Timer Interrupt Register 293
Table 17-13: Timer Control 0 Register 294
Table 17-14: Timer Non-Overlapping Compare Register 297
Table 17-15: Timer Control 1 Register 297
Table 17-16: Timer Wakeup Status Register 299
Table 18-1: MAX32655 WUT Clock Period 300
Table 18-2: Wakeup Timer Register Summary 303
Table 18-3: Wakeup Timer Count Register 304
Table 18-4: Wakeup Timer Compare Register 304
Table 18-5: Wakeup Timer PWM Register 304
Table 18-6: Wakeup Timer Interrupt Register 304
Table 18-7: Wakeup Timer Control Register 304
Table 18-8: Wakeup Timer Non-Overlapping Compare Register 305
Table 19-1: MAX32655 WDT Instances Summary 307
Table 19-2: WDT Event Summary 309
Table 19-3: WDT Register Summary 313
Table 19-4: WDT Control Register 313
Table 19-5: WDT Reset Register 316
Table 19-6: WDT Clock Source Select Register 316
Table 19-7: WDT Count Register 316
Table 20-1: Pulse Train Engine Register Summary 320
Table 20-2: Pulse Train Engine Global Enable/Disable Register 321
Table 20-3: Pulse Train Engine Resync Register 321
Table 20-4:Pulse Train Engine Stopped Interrupt Flag Register 322
Table 20-5: Pulse Train Engine Interrupt Enable Register 323
Table 20-6: Pulse Train Engine Safe Enable Register 323
Table 20-7: Pulse Train Engine Safe Disable Register 324
Table 20-8: Pulse Train Engine Configuration Register 324
Table 20-9: Pulse Train Mode Bit Pattern Register 325
Table 20-10: Pulse Train n Loop Configuration Register 325
Table 20-11: Pulse Train n Automatic Restart Configuration Register 325
Table 21-1: MAX32655 CRC Instances 327
Table 21-2: Organization of Calculated Result in CRC_VAL.value 328
Table 21-3: Common CRC Polynomials 328
Table 21-4: CRC Register Summary 330
Table 21-5: CRC Control Register 330
Table 21-6: CRC Data Input 32 Register 330
Table 21-7: CRC Polynomial Register 331
Table 21-8: CRC Value Register 331
Table 22-1: MAX32655 AES Instances 332
Table 22-2: Interrupt Events 334

Table 22-3: AES Register Summary 335
Table 22-4: AES Control Register 335
Table 22-5: AES Status Register 336
Table 22-6: AES Interrupt Flag Register 336
Table 22-7: AES Interrupt Enable Register 337
Table 22-8: AES FIFO Register 337
Table 23-1: Register Summary 338
Table 23-2: TRNG Control Register 338
Table 23-3: TRNG Status Register 338
Table 23-4: TRNG Data Register 339
Table 25-1: MAX32655 Bootloader Instances 343
Table 25-2: MAX32655 Application Image Structure 345
Table 26-1: Revision History 347

1. Overview

The MAX32655 microcontroller (MCU) is an advanced system-on-chip featuring an Arm® Cortex®-M4F CPU for efficient computation of complex functions and algorithms. The SoC integrates power regulation and management with a Single Inductor Multiple Output (SIMO) buck regulator system. The latest generation Bluetooth 5.2 Low Energy (LE) radio, supporting LE Audio, Angle of Arrival (AoA), and Angle of Departure (AoD) for direction finding, long-range (coded), and high-throughput modes are also available.

The device offers large internal memory with 512KB flash and 128KB SRAM, with optional error correction coding on one 32KB SRAM bank. This 32KB bank can be optionally retained in backup mode. An 8KB user OTP area is available, and 8 bytes are retained, even during power down.

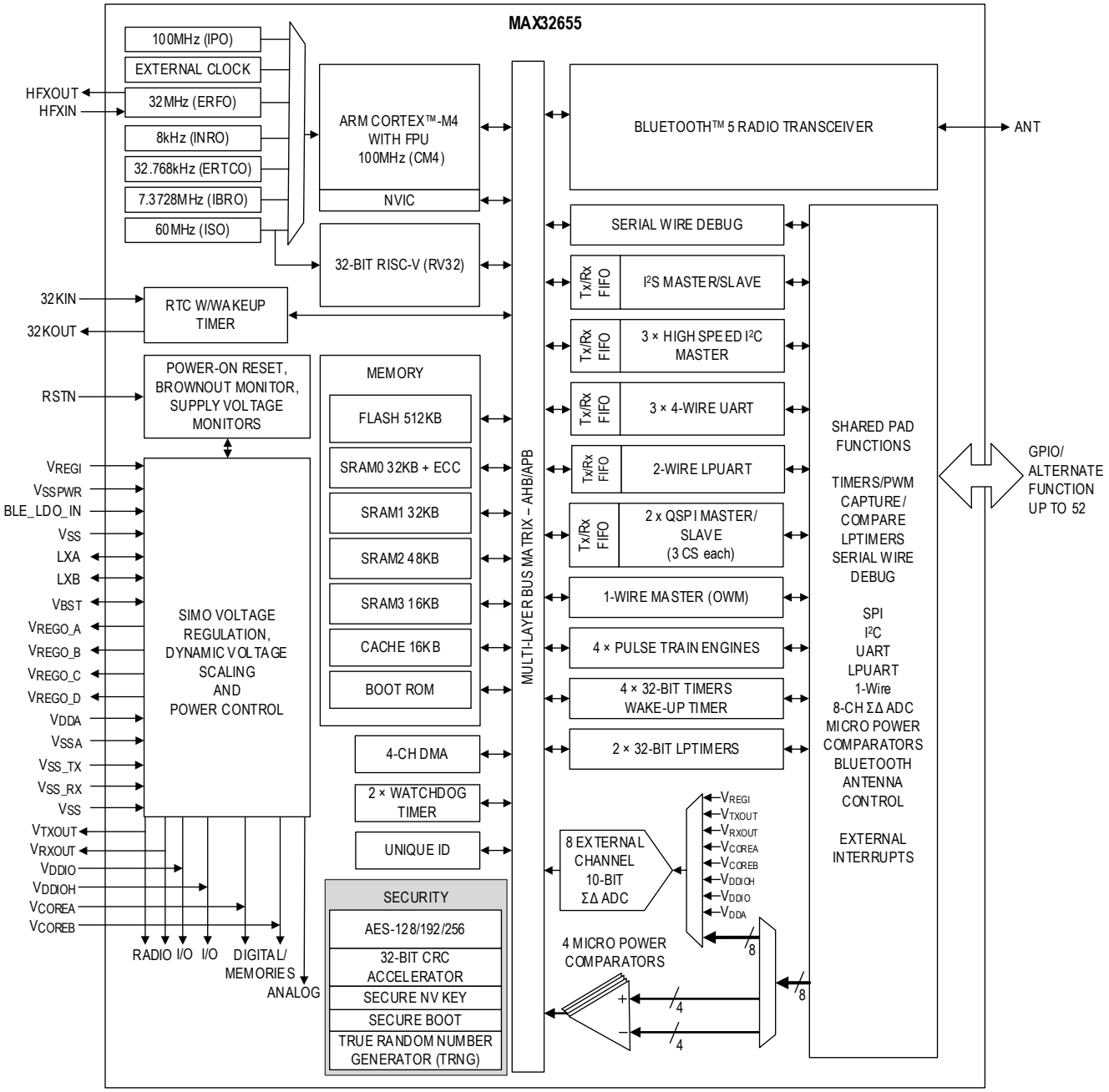
Many high-speed interfaces are supported on the device including multiple QSPI, UART, and I^2C serial interfaces, and one I^2S port for connecting to an audio codec. An eight-input, 10-bit ADC is available to monitor analog input from external analog sources.

The device is available in an 81-CTBGA, 8mm × 8mm, 0.8mm pitch.

The high-level block diagram for the MAX32655 is shown in *Figure 1-1*.

1.1 Block Diagram

Figure 1-1: MAX32655 Block Diagram



2. Memory, Register Mapping, and Access

2.1 Memory, Register Mapping, and Access Overview

The Arm Cortex-M4 architecture defines a standard memory space for unified code and data access. This memory space is addressed in units of single bytes but is most typically accessed in 32-bit (4 byte) units. It may also be accessed, depending on the implementation, in 8-bit (1 byte) or 16-bit (2 byte) widths. The total range of the memory space is 32 bits wide (4GB addressable total), from addresses 0x0000 0000 to 0xFFFF FFFF.

It is important to note, however, that the architectural definition does not require the entire 4GB memory range to be populated with addressable memory instances.

Figure 2-1: CM4 Code Memory Mapping

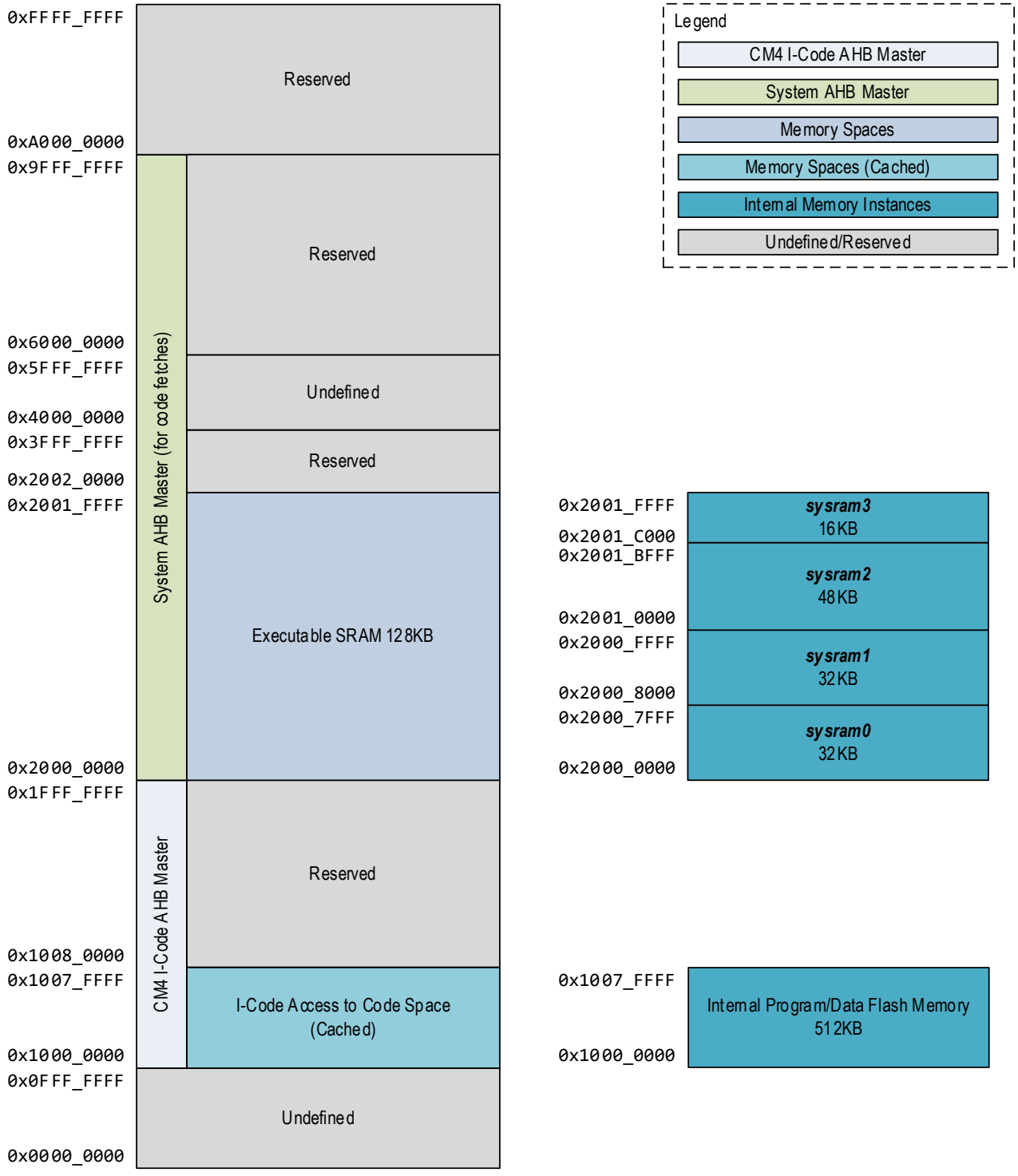


Figure 2-2: RISC-V IBUS Code Memory Mapping

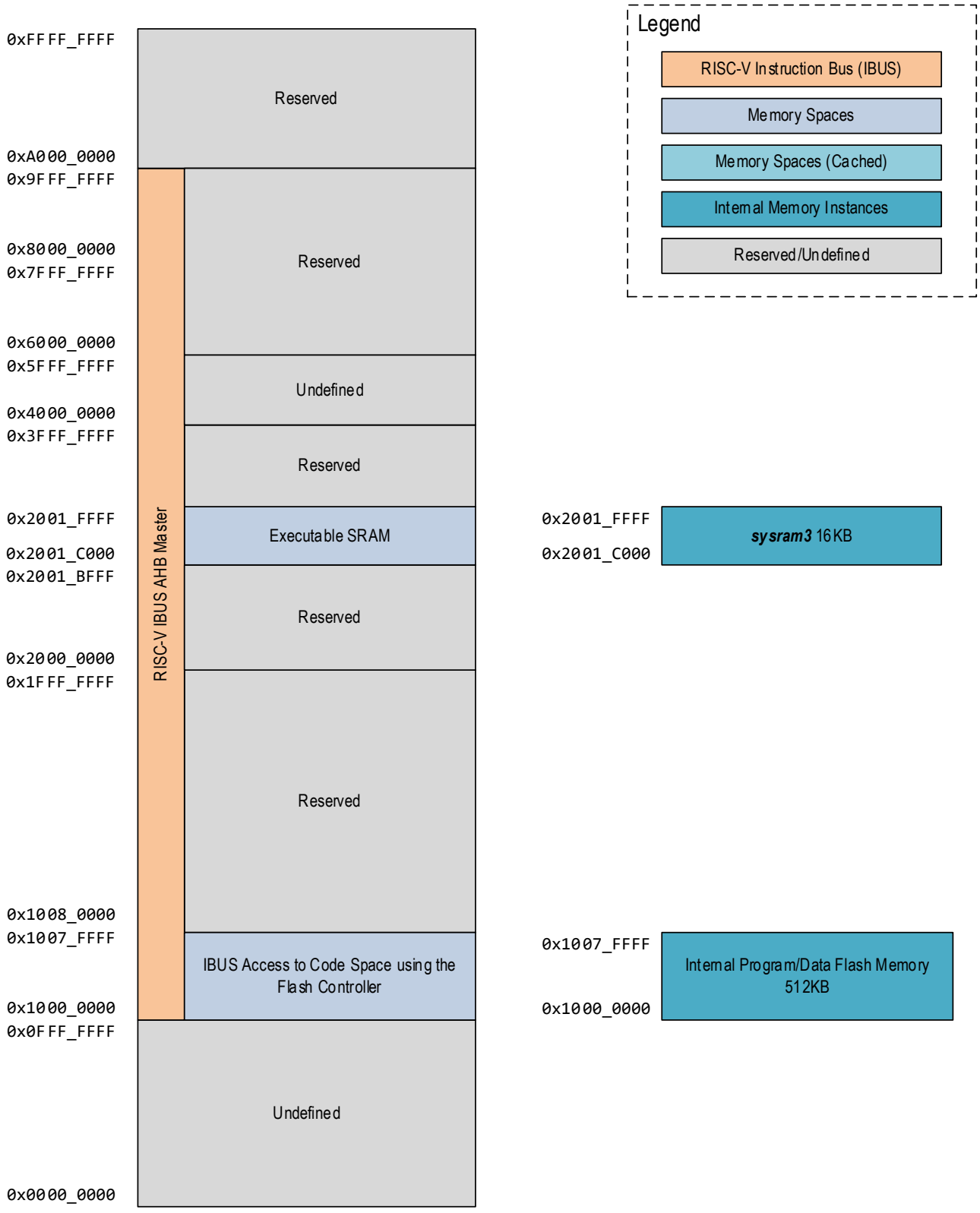


Figure 2-3: CM4 Peripheral and Data Memory Mapping

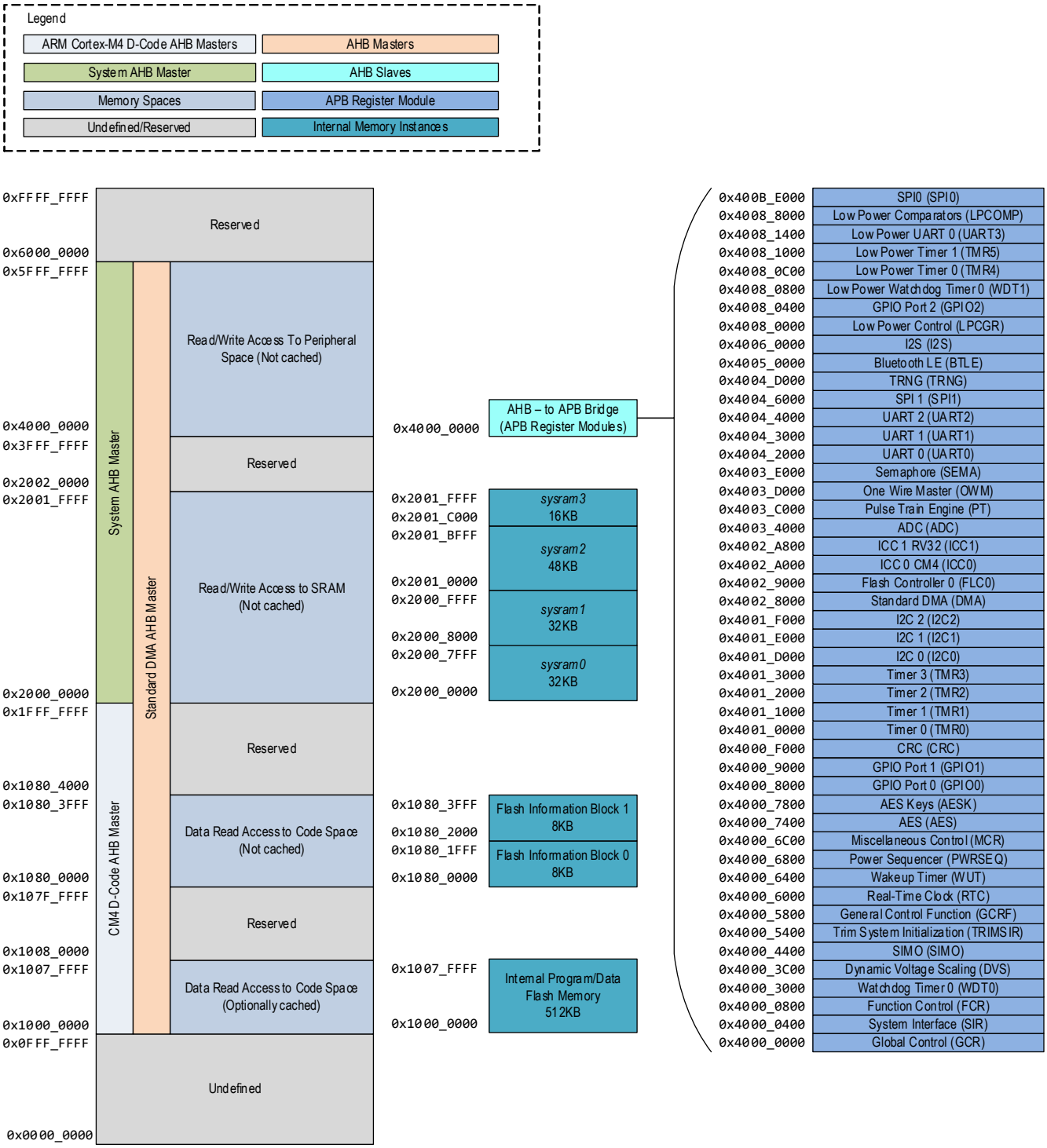
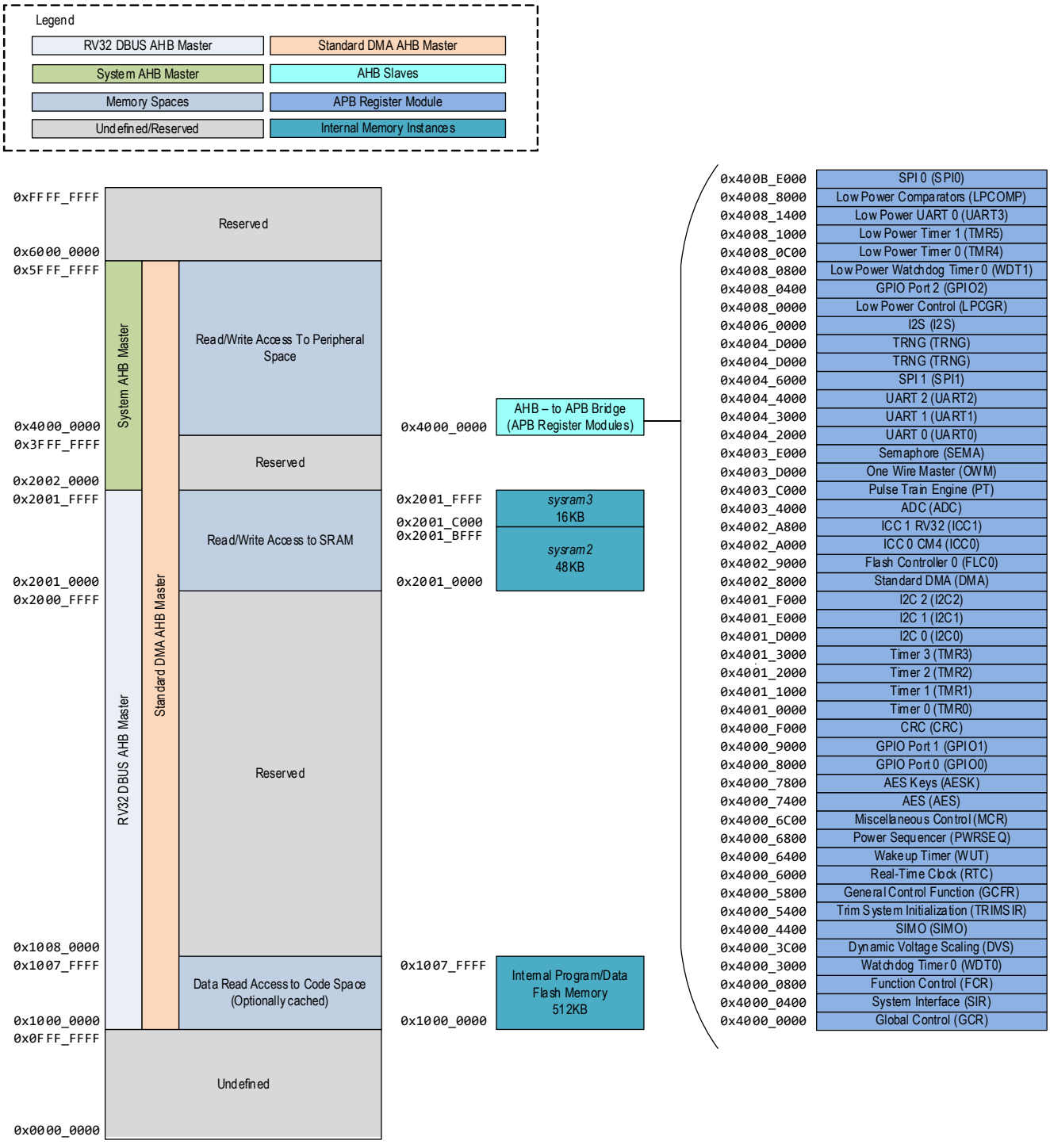


Figure 2-4: RV32 DBUS Peripheral and Data Memory Mapping



2.2 Field Access Definitions

All fields accessible by user software have distinct access capabilities. Each register table contained in this user guide has an access type defined for each field. The definition of each field access type is presented in *Table 2-1*.

Table 2-1: Field Access Definitions

Access Type	Definition
RO	**Reserved** This access type is reserved for static fields. Reads of this field return the reset value. Writes are ignored.
DNM	**Reserved. Do Not Modify** Reads of this field return indeterminate values. Software must first read this field and write the same value whenever writing to this register.
R	**Read Only** Reads of this field return a value. Writes to the field have no effect on device operation.
W	**Write Only** Reads of this field return indeterminate values. Writes to the field may change the field's state and may affect device operation.
R/W	**Unrestricted Read/Write** Reads of this field return a value. Writes to the field may change the field's state and may affect device operation.
RC	**Read-to-Clear** Reads of this field may return a value. Any read of this register clears the field to 0. Writes to the field have no effect on device operation.
RS	**Read-to-Set** Reads of this field may return a value. Any read of this register sets the field to 1. Writes to the field have no effect on device operation.
R/W0	**Read-Write-0-Only** Reads of this field may return a value. Writing 0 to this field may change the field's state and may affect device operation. Writing 1 to the field has no effect on device operation.
R/W1	**Read-Write-1-Only** Reads of this field may return a value. Writing 1 to this field may change the field's state and may affect device operation. Writing 0 to the field has no effect on device operation.
R/W1C	**Read-Write-1-to-Clear** Reads of this field may return a value. Writing 1 to this field clears this field to 0. Writing 0 to the field has no effect on device operation.
W1C	**Write-1-to-Clear** Reads of this field return indeterminate values. Writing 1 to this field clears this field to 0. Writing 0 to the field has no effect on device operation.
R/W0S	**Read-Write-0-to-Set** Reads of this field may return a value. Writing 0 to this field sets this field to 1. Writing 1 to the field has no effect on device operation.

2.3 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 and RISC-V architectures; the use of many of these is optional for the system integrator. At a minimum, the MAX32655 must contain some code and data memory for application code and variable/stack use for CPU0, as well as certain components that are part of the instantiated Cortex-M4 core.

2.3.1 Code Space

The code space area of memory is designed to contain the primary memory used for code execution by the device. This memory area is defined from byte address range 0x0000 0000 to 0x1FFF FFFF (0.5GB maximum). Two different standard core bus masters are used by the Cortex-M4 core and Arm debugger to access this memory area. The I-Code AHB bus master is used for instruction decode fetching from code memory, while the D-Code AHB bus master is used for data fetches from code memory. This is arranged so that data fetches avoid interfering with instruction execution. Additionally, the RV32 uses the IBUS for instruction fetches from code memory and the DBUS for data fetches from code memory.

The MAX32655 code memory mapping is illustrated in *Figure 2-1*. The code space memory area contains the main internal flash memory, which holds most of the instruction code executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1007 FFFF.

This program memory area must also contain the default system vector table and the initial settings for all system exception handlers and interrupt handlers. The reset vector for the device is 0x0000 0000 and contains the device ROM code that transfers execution to user code at address 0x1000 0000.

The code space memory on the MAX32655 also contains the mapping for the flash information block, from 0x1080 0000 to 0x1080 3FFF. However, this mapping is generally only present during Maxim Integrated production test; it is disabled once the information block is loaded with valid data and the info block lockout option is set. This memory is accessible for data reads only and cannot be used for code execution. The flash information block is user read only accessible and contains the Unique Serial Number (USN).

2.3.2 Information Block Flash Memory

The information block is a separate area of the Internal Flash Memory and is 16,384 Bytes. The information block is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information. The information block also contains the device's Unique Serial Number (USN). The USN is a 104-bit field. USN bits 0 thru 7 contain the die revision.

Figure 2-5: Unique Serial Number Format

		Bit Position																														
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address	0x10800000	USN bits 16 - 0																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	0x10800004	x	USN bits 47-17																														
	0x10800008	USN bits 64 - 48																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	0x1080000C	x	USN bits 95 - 65																														
	0x10800010	x	x	x	x	x	x	x	x	x	USN bits 103 - 96								x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Reading the USN requires the information block to be unlocked. Unlocking the information block does not enable write access to the block but allows the contents of the USN to be read from the block. Unlock the information block using the following steps:

1. Write 0x3A7F 5CA3 to *FLCn_ACNTL*.
2. Write 0xA1E3 4F20 to *FLCn_ACNTL*.
3. Write 0x9608 B2C1 to *FLCn_ACNTL*.
4. Information block is now read-only accessible.

To re-lock the information block to prevent access, simply write any 32-bit word (with a value other than one of the three values required for the unlock sequence above) to *FLCn_ACNTL*.

2.3.3 SRAM Space

The SRAM area of memory is intended to contain the primary SRAM data memory of the device and is defined from byte address range 0x2000 0000 to 0x3FFF FFFF (0.5GB maximum). This memory can be used for general purpose variable and data storage, code execution, and the CM4 stack as well as the RV32 stack.

The MAX32655 CM4's data memory mapping is illustrated in *Figure 2-1*. The MAX32655 RV32's data memory mapping is illustrated in *Figure 2-2*.

The system SRAM configuration is defined in *Table 2-2*.

The SRAM memory area contains the main system SRAM. The size of the internal general-purpose data SRAM is 128KB. The SRAM is divided into four blocks and the contiguous address range is 0x2000 0000 to 0x2001 FFFF.

The SRAM area on the MAX32655 can be used for data storage and code execution by the CM4. The RV32 is limited to use of *sysram2* and *sysram3* for code and data storage in SRAM.

Note: After a POR, the CM4 has access to all four SRAM regions. sysram2 and sysram3 can be configured to restrict access from the CM4 to prevent unintended modifications of these SRAM instances by the CM4. Set the FCR_URVCTRL.memsel field to 1 to set the RV32 core as the exclusive master for sysram2 and sysram3.

Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the CM4 and RV32 stack must be located, as it is the only general-purpose SRAM memory on the device capable of this function.

Table 2-2: System SRAM Configuration

System RAM Block #	Size	Start Address	End Address	CM4 Accessible	RV32 Accessible
sysram0	32KB	0x2000 0000	0x2000 7FFF	✓	No
sysram1	32KB	0x2000 8000	0x2000 FFFF	✓	No
sysram2	48KB	0x2001 0000	0x2001 BFFF	Configurable	✓
sysram3	16KB	0x2001 C000	0x2001 FFFF	Configurable	✓ (Optional ICC1)

The MAX32655 specific AHB Bus Masters can access the SRAM to use as general storage or working space.

The entirety of the SRAM memory space on the MAX32655 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). This means that the CPU can access the entire SRAM either using standard byte/word/doubleword access or bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 128KB bit banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area translates into a single bit access (read or write) in the bit-banding primary area. Reading from the location performs a single bit read, while writing either a 1 or 0 to the location performs a single bit set or clear.

Note: The Arm Cortex-M4 core translates the access in the bit-banding alias area into the appropriate read cycle (for a single bit read) or a read-modify-write cycle (for a single bit set or clear) of the bit-banding primary area. This means that bit-banding is a core function (i.e., not a function of the SRAM memory interface layer or the AHB bus layer), and thus is only applicable to accesses generated by the core itself. Reads/writes to the bit-banding alias area by other (non-Arm-core) bus masters do not trigger a bit-banding operation and instead result in an AHB bus error.

2.3.4 Peripheral Space

The peripheral space area of memory is intended for mapping of control registers, internal buffers/working space, and other features needed for the firmware control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32655, all device-specific module registers are mapped to this memory area, as well as any local memory buffers or FIFOs required by modules.

As with the SRAM region, there is a dedicated 1MB area at the bottom of this memory region (from 0x4000 0000 to 0x400F FFFF) used for bit-banding operations by the Arm core. Four-byte-aligned read/write operations in the peripheral bit-banding alias area (32MB in length, from 0x4200 0000 to 0x43FF FFFF) are translated by the core into read/mask/shift or read/modify/write operation sequences to the appropriate byte location in the bit-banding area.

Note: The bit-banding operation within peripheral memory space is, like bit-banding function in SRAM space, a core remapping function. As such, it is only applicable to operations performed directly by the Arm core. If another memory bus master accesses the peripheral bit-banding alias region, the bit-banding remapping operation does not take place. In this case, the bit-banding alias region appears to be a non-implemented memory area (causing an AHB bus error).

On the MAX32655, access to the region that contains most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge. This allows the peripheral modules to operate on the lower power APB bus matrix. This also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must necessarily have a faster response time since it handles main application instruction and data fetching.

2.3.5 External RAM Space

The external RAM space area of memory is intended for use in mapping off-chip external memory and is defined from byte address range 0x6000 0000 to 0x9FFF FFFF (1GB maximum). *The MAX32655 does not implement this memory area.*

2.3.6 External Device Space

The external device space area of memory is intended for use in mapping off-chip device control functions onto the AHB bus. This memory space is defined from byte address range 0xA000 0000 to 0xDFFF FFFF (1GB maximum). The MAX32655 does not implement this memory area.

2.3.7 System Area (Private Peripheral Bus)

The system area (private peripheral bus) memory space contains register areas for functions that are only accessible by the Arm core itself (and the Arm debugger, in certain instances). It is defined from byte address range 0xE000 0000 to 0xE00F FFFF. This APB bus is restricted and can only be accessed by the Arm core and core-internal functions. It cannot be accessed by other modules which implement AHB memory masters, such as the DMA interface.

In addition to being restricted to the core, application code is only allowed to access this area when running in the privileged execution mode (as opposed to the standard user thread execution mode). This helps ensure that critical system settings controlled in this area are not altered inadvertently or by errant code that should not have access to this area.

Core functions controlled by registers mapped to this area include the SysTick timer, debug and tracing functions, the NVIC controller, and the flash breakpoint controller.

2.3.8 System Area (Vendor Defined)

The system area (vendor defined) memory space is reserved for vendor (system integrator) specific functions not handled by another memory area. It is defined from byte address range 0xE010 0000 to 0xFFFF FFFF. The MAX32655 does not implement this memory region.

2.4 Device Memory Instances

This section details physical memory instances on the MAX32655 (including internal flash memory and SRAM instances) accessible as standalone memory regions using either the AHB or APB bus matrix. Memory areas that are only accessible through FIFO interfaces, or memory areas consisting of only a few registers for a specific peripheral, are not covered here.

2.4.1 Main Program Flash Memory

The main program flash memory is 512KB and consists of 64 logical pages of 8,192 bytes per page.

2.4.2 Instruction Cache Memory

The MAX32655 includes a dedicated internal cache controller with 16,384 bytes of cache memory for the CM4 core. Optionally, *sysram3* can be used as an internal cache for the RV32.

The instruction cache memory is used to cache instructions fetched through the I-Code bus, including instructions fetched by the core from the internal flash memory.

2.4.3 System SRAM

The system SRAM is 128KB in size and can be used for general purpose data storage, the Arm Cortex-M4 system stack and code execution, and the RISC-V system stack and code execution.

2.4.4 AES Key and Working Space Memory

The AES key memory and working space for AES operations (including input and output parameters) are in a dedicated register file memory tied to the AES engine block. This AES memory is mapped into AHB space for rapid firmware access.

2.5 AHB Interfaces

This section details memory accessibility on the AHB and the organization of AHB master and slave instances.

2.5.1 Arm Core AHB Interfaces

2.5.1.1 I-Code

This AHB master is used by the Arm core for instruction fetching from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus master is used to fetch instructions from the internal flash memory. Instructions fetched by this bus master are returned by the instruction cache, which in turn triggers a cache line fill cycle to fetch instructions from the internal flash memory when a cache miss occurs.

2.5.1.2 D-Code

This AHB master is used by the Arm core for data fetches from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus master has access to the internal flash memory and information block.

2.5.1.3 System

This AHB master is used by the Arm core for all instruction fetches, and data read and write operations involving the SRAM data cache. The APB mapped peripherals (through the AHB-to-APB bridge) and AHB mapped peripheral and memory areas are also accessed using this bus master.

2.5.2 AHB Slaves

2.5.2.1 Standard DMA

The Standard DMA AHB slave has access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

2.5.2.2 SPI0

The SPI1 AHB slave has access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

2.5.2.3 AHB Slave Base Address Map

Table 2-3 contains the base address for each of the AHB slave peripherals. The base address for a given peripheral is the start of the register map for the peripheral. For a given peripheral, the address for a register within the peripheral is defined as the peripheral's AHB Base Address plus the register's offset.

Table 2-3: AHB Slave Base Address Map

AHB Slave Register Name	Register Prefix	AHB Base Address	AHB End Address
SPI0	SPI0_	0x400B E000	0x400B E3FF

2.6 Peripheral Register Map

2.6.1 APB Peripheral Base Address Map

Table 2-4 contains the base address for each of the APB mapped peripherals. The base address for a given peripheral is the start of the register map for the peripheral. For a given peripheral, the address for a register within the peripheral is defined as the APB peripheral base address plus the register's offset.

Table 2-4: APB Peripheral Base Address Map

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Function Control	FCR_	0x4000 0800	0x4000 0BFF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Dynamic Voltage Scaling Controller	DVS_	0x4000 3C00	0x4000 3C3F
Single Input Multiple Output	SIMO_	0x4000 4400	0x4000 47FF
Trim System Initialization	TRIMSIR_	0x4000 5400	0x4000 57FF
General Control Function	GCFR_	0x4000 5800	0x4000 5BFF
Real time Clock	RTC_	0x4000 6000	0x4000 63FF
Wakeup Timer	WUT_	0x4000 6400	0x4000 67FF
Power Sequencer	PWRSEQ_	0x4000 6800	0x4000 6BFF
Miscellaneous Control	MCR_	0x4000 6C00	0x4000 6FFF
AES	AES_	0x4000 7400	0x4000 77FF
AES Key	AESK_	0x4000 7800	0x4000 7BFF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
CRC	CRC_	0x4000 F000	0x4000 FFFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
Timer 3	TMR3_	0x4001 3000	0x4001 3FFF
I^2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^2C 1	I2C1_	0x4001 E000	0x4001 EFFF
I^2C 2	I2C2_	0x4001 F000	0x4001 FFFF
Standard DMA	DMA_	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Instruction-Cache Controller 0 (CM4)	ICC0_	0x4002 A000	0x4002 A7FF
Instruction Cache Controller 1 (RV32)	ICC1_	0x4002 A800	0x4002 AFFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
ADC	ADC_	0x4003 4000	0x4003 4FFF
Pulse Train Engine	PT_	0x4003 C000	0x4003 C09F
One Wire Master	OWM0_	0x4003 D000	0x4003 DFFF
Semaphore	SEMA_	0x4003 E000	0x4003 EFFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
UART 2	UART2_	0x4004 4000	0x4004 4FFF
SPI1	SPI1_	0x4004 6000	0x4004 7FFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
Bluetooth Registers and IQ RAMs	BTLE_	0x4005 0000	0x4005 FFFF
I^2S	I2S_	0x4006 0000	0x4006 0FFF
Low-Power General Control	LPGCR_	0x4008 0000	0x4008 03FF
GPIO Port 2	GPIO2_	0x4008 0400	0x4008 05FF
GPIO Port 3	GPIO3_	0x4008 0600	0x4008 07FF
Low-Power Watchdog Timer 0 (WDT1)	WDT1_	0x4008 0800	0x4008 0BFF
Low-Power Timer 0 (Timer 4)	TMR4_	0x4008 0C00	0x4008 0FFF
Low-Power Timer 1 (Timer 5)	TMR5_	0x4008 1000	0x4008 13FF
Low-Power UART 0 (UART 3)	UART3_	0x4008 1400	0x4008 17FF
Low-Power Comparators	LPCOMP_	0x4008 8000	0x4008 83FF

2.7 Error Correction Coding (ECC) Module

This device features an Error Correction Coding (ECC) module that helps ensure data integrity by detecting and correcting bit corruption of the system RAM 0 (*sysram0*) memory array. More specific, this feature is single error correcting, double error detecting (SEC-DED). It corrects any single bit flip, detects 2-bit errors, and features a transparent zero wait state operation for reads.

The ECC works by creating check bits for all data written to *sysram0*. These check bits are then stored along with the data. During a read, both the data and check bits are used to determine if one or more bits have become corrupt. If a single bit is corrupted, this can be corrected. If two bits are corrupted, it is detected, but not corrected.

If only one bit is determined to be corrupt, reads contain the “corrected” value. Reading memory does not correct the errored value stored at the read memory location. It is up to the application firmware to determine the appropriate time and method to write the correct data to memory. It is strongly recommended that the application firmware correct the memory as soon as possible to minimize the chance of a second bit from becoming corrupt, resulting in data loss. Since ECC error checking only occurs during a read operation, it is recommended that the software periodically reads critical memory so that errors can be identified and corrected.

2.7.1 SRAM

A check bit RAM is used to store *sysram0*'s check bits enabling ECC SEC-DED for *sysram0*. The check bit RAM is not mapped to the user memory space and is not available for application usage.

2.7.2 Limitations

Any read from non-initialized memory can trigger an ECC error since the random check bits most likely do not match the random data bits contained in the memory. Writing *sysram0* to all zeroes prior to enabling ECC functionality can prevent this at the expense of the time required.

3. System, Power, Clocks, Reset

There are several clocks used by different peripherals and subsystems. These clocks are highly configurable by firmware, allowing developers to select the combination of application performance and power savings required for the target systems. Support for selectable core operating voltage is provided and the Internal Primary Oscillator (IPO) frequency is scaled based on the specific core operating voltage range selected.

3.1 Oscillator Sources

3.1.1.1 100MHz Internal Primary Oscillator (IPO)

The MAX32655 includes a 100MHz internal high-speed oscillator, referred to in this document at the Internal Primary Oscillator (IPO). This is the fastest oscillator and draws the most power.

The IPO can optionally be powered down in *LPM* by setting the *GCR_PM.ipo_pd* field to 1.

The IPO can be selected as SYS_OSC. To use this oscillator as SYS_OSC, the following steps must be followed:

1. Enable the IPO by setting *GCR_CLKCTRL.ipo_en* to 1.
2. Wait until the *GCR_CLKCTRL.ipo_rdy* field reads 1, indicating the IPO is operating.
3. Set *GCR_CLKCTRL.sysclk_sel* to 4.
4. Set *GCR_CLKCTRL.sysclk_rdy* field reads 1. The IPO is now operating as the SYS_OSC.

3.1.1.2 32MHz External RF Oscillator (ERFO)

This is the oscillator that directly drives the Bluetooth radio. It can also be selected as the SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 3-1* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}. To enable this oscillator, the Bluetooth LDOs, both the LDORX and LDOTX must be enabled by setting the following fields:

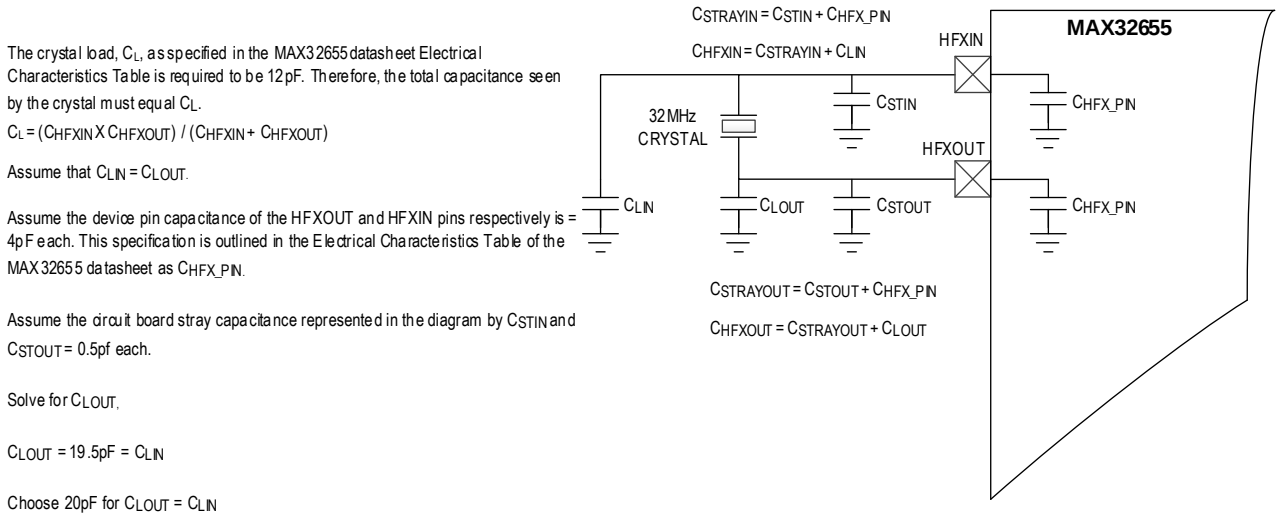
- *GCR_BTLELDOCTRL.ldorxen*
- *GCR_BTLELDOCTRL.ldotxen*

To use this oscillator as SYS_OSC, the following steps must be followed:

1. Enable the internal secondary oscillator (ISO) by setting *GCR_CLKCTRL.iso_en* to 1.
2. Wait until *GCR_CLKCTRL.iso_rdy* reads 1. The ISO is now operating.
3. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 1.
4. Wait until *GCR_CLKCTRL.erfo_rdy* reads 1. The ERFO is now operating.
5. Set *GCR_CLKCTRL.sysclk_sel* = 2. This will select the ERFO as the SYS_OSC.
6. Wait until *GCR_CLKCTRL.sysclk_rdy* is set. The ERFO is now operating as the SYS_OSC.

Note: The ISO must remain enabled while the ERFO is operating as the SYS_OSC.

Figure 3-1: Example 32MHz Crystal Capacitor Determination



3.1.1.2.1 ERFO Kick Start

The device includes an internal kick start circuit that improves the startup time for the ERFO. This kick start circuit enables the software to programmatically determine the optimal settings required for the fastest startup time for a given external oscillator. The ERFO startup time is crystal, temperature, and layout dependent, therefore it is suggested that the software settings required for optimal startup time are performed on the final system level design.

3.1.1.2.2 Programmatic Determination of the Optimum Kick Start Settings

The following steps describe a methodology for determining the optimum settings for the kick start circuit in a given application:

1. Disable the ERFO by setting *GCR_CLKCTRL*.*erfo_en* to 0.
2. Disable the kick start circuit by setting the following fields to 0:
   a. *FCR_ERFOKS*.*ksclksel*
   b. *FCR_ERFOKS*.*kserfodriver*
   c. *FCR_ERFOKS*.*kserfo_cnt*
3. Select either the ISO or IPO to use to kick start the ERFO as follows:
   a. For the ISO, set the *FCR_ERFOKS*.*ksclksel* field to 2.
   b. For the IPO, set the *FCR_ERFOKS*.*ksclksel* field to 3.
4. Enable the BTLE LDOs by setting the *GCR_BTLELDOCTRL*.*ldorxen* and *GCR_BTLELDOCTRL*.*ldotxen* fields to 1.
   Note: The BTLE LDOs must be enabled to use the ERFO regardless if the BTLE peripheral is used.
5. Enable the oscillator selected in step 3, used to kick start the ERFO, by setting the appropriate clock enable as follows:
   a. For the IPO, set *GCR_CLKCTRL*.*ipo_en* to 1 and read the *GCR_CLKCTRL*.*ipo_rdy* until it reads 1.
   b. For the ISO, set *GCR_CLKCTRL*.*iso_en* to 1 and read the *GCR_CLKCTRL*.*iso_rdy* until it reads 1.

6. Configure a timer for counter mode using the oscillator selected in step 3 as the timer clock. See *Timers (TMR/LPTMR)* for details of timer configuration and modes.
7. The number of kick start pulses supported, range from 1 to 127. The following steps should be performed by incrementing the kick start count by 1 on each iteration while measuring the amount of time it takes to start up the ERFO. Once the fastest startup time is determined, the settings should be saved and used any time the ERFO is started by the software.
    a. Set the kick start pulse count, *FCR_ERFOKS*.*kserfo_cnt*, to the number of kick start pulses to test. This is the iteration number from 1 to 127.
    b. Enable the kick start circuit by setting *FCR_ERFOKS*.*kserfo_en* to 1.
    b. Start the configured timer.
    c. Enable the ERFO by setting *GCR_CLKCTRL*.*erfo_en* to 1.
    d. Read the *GCR_CLKCTRL*.*erfo_rdy* field until it reads 1.
    e. Stop the timer and determine the elapsed time.
    f. Disable the kick start circuit by setting *FCR_ERFOKS*.*kserfo_en* to 0.
    g. Repeat steps *a* to *f* until all supported kick start pulses are tested and determine the optimum kick start pulses for the fastest ERFO startup time.

3.1.1.2.3 Using the ERFO Kick Start

To use the ERFO kick start circuit perform the following steps when enabling the ERFO:

1. Set the kick start pulse count by setting the *FCR_ERFOKS*.*kserfo_cnt* field.
2. Enable the kick start circuit by setting *FCR_ERFOKS*.*kserfo_en* to 1.
3. Enable the ERFO by setting *GCR_CLKCTRL*.*erfo_en* to 1.
4. Read the *GCR_CLKCTRL*.*erfo_rdy* field until it reads 1.
5. Disable the kick start circuit by setting *FCR_ERFOKS*.*kserfo_en* to 0.

3.1.1.3 60MHz Internal Secondary Oscillator (ISO)

This is a low-power internal secondary oscillator that can be selected as SYS_OSC. This oscillator is automatically selected as SYS_OSC after a System Reset or POR.

3.1.1.4 7.3728MHz Internal Baud Rate Oscillator (IBRO)

This is an exceptionally low-power internal oscillator that can be selected as SYS_OSC. This clock can optionally be used as a dedicated baud rate clock for the UARTs. This is useful if the SYS_OSC selected does not allow the targeted UART baud rate.

This oscillator can optionally be automatically powered down when in *LPM* and *UPM* by setting register bit *GCR_PM*.*ibro_pd*.

This oscillator is disabled by default at power-up.

3.1.1.5 32.768kHz External Real-Time Clock Oscillator (ERTCO)

This is an exceptionally low-power internal oscillator that can be selected as SYS_OSC. This oscillator can optionally use a 32.768kHz input clock instead of an external crystal. The internal 32.768kHz clock is available as an output on GPIO as an alternate function (SQWOUT).

This oscillator is the dedicated clock for the Real-Time Clock (RTC). If the RTC is enabled, the ERTCO must be enabled, independent of the selection of SYS_OSC. This oscillator is disabled at power-up.

3.1.1.6 8kHz-30kHz Internal Nano-Ring Oscillator (INRO)

This is an ultra-low-power internal oscillator that can be selected as SYS_OSC. This oscillator is enabled at power-up and cannot be disabled by firmware.

The frequency of this oscillator is configurable to 8kHz, 16kHz or 30kHz. Use the *TRIMSIR_INRO.inro_sel* field to select the desired frequency. The power-on reset frequency defaults to 30kHz.

3.2 System Oscillator (SYS_OSC)

The MAX32655 supports multiple clock sources as the System Oscillator (SYS_OSC). Each oscillator, description, and nominal frequency are shown in *Table 3-1*. The ERTCO requires an external crystal for operation. An external clock, EXT_CLK, source is supported on P0.3, Alternate Function 1.

Table 3-1: Oscillators, Descriptions, and Nominal Frequencies

Oscillator	Description	Nominal Frequency
IPO	Internal Primary Oscillator	100MHz
ISO	Internal Secondary Oscillator	60MHz
INRO	Internal Nano-Ring Oscillator	Configurable 8kHz, 16kHz, or 30kHz
IBRO	Internal Baud Rate Oscillator	7.3728MHz
ERTCO	External Real-Time Clock Oscillator	32.768kHz
EFRO	External RF Oscillator	32MHz
EXT_CLK	External Clock	Up to 80MHz

3.2.1 System Oscillator Selection

Set the system oscillator using the *GCR_CLKCTRL.sysclk_sel* field. Prior to selecting an oscillator as the system oscillator, the oscillator source must first be enabled and ready. See *Oscillator Sources* for each oscillator's detailed description for the required steps to enable the oscillator and select it as a system oscillator.

When the *GCR_CLKCTRL.sysclk_sel* is modified, the hardware clears the *GCR_CLKCTRL.sysclk_rdy* field and there is a delay until the switchover is complete. When the switchover to the selected SYS_OSC is complete, the *GCR_CLKCTRL.sysclk_rdy* field is set to 1 by the hardware. The software must verify the switchover is complete before continuing operation.

3.2.2 System Clock (SYS_CLK)

The selected SYS_OSC is the input to the system oscillator prescaler to generate the System Clock (SYS_CLK). The system oscillator prescaler divides the selected SYS_OSC by a prescaler using the *GCR_CLKCTRL.sysclk_div* field as shown in *Equation 3-1*.

Equation 3-1: System Clock Scaling

$$SYS\_CLK = \frac{SYS\_OSC}{2^{sysclk\_div}}$$

GCR_CLKCTRL.sysclk_div is selectable from 0 to 7, resulting in divisors of 1, 2, 4, 8, 16, 32, 64, or 128.

SYS_CLK drives the Arm Cortex-M4 with FPU core, the RV32 core and all Advanced High-Performance Bus (AHB) masters in the system. SYS_CLK generates the following internal clocks as shown below:

- Advanced High-Performance Bus (AHB) Clock
  - $HCLK = SYS\_CLK$
- Advanced Peripheral Bus (APB) Clock,
  - $PCLK = \frac{SYS\_CLK}{2}$

The Real-Time Clock (RTC) uses the 32.768kHz ERTCO for its clock source. Optionally, the RTC can run using an internal dedicated 8kHz nano-ring oscillator. See *Real-Time Clock (RTC)* for details on using this 8kHz nano-ring oscillator for the RTC.

All oscillators are reset to their POR reset default state during:

- Power-On Reset
- System Reset

Oscillator settings are *not* reset during:

- Soft Reset
- Peripheral Reset

Table 3-2 shows each oscillator's enabled state for each type of reset source in the MAX32655. *Table 3-3* details the effect each reset source has on the System Clock selection and System Clock prescaler settings.

Table 3-2: Reset Sources and Effect on Oscillator Status

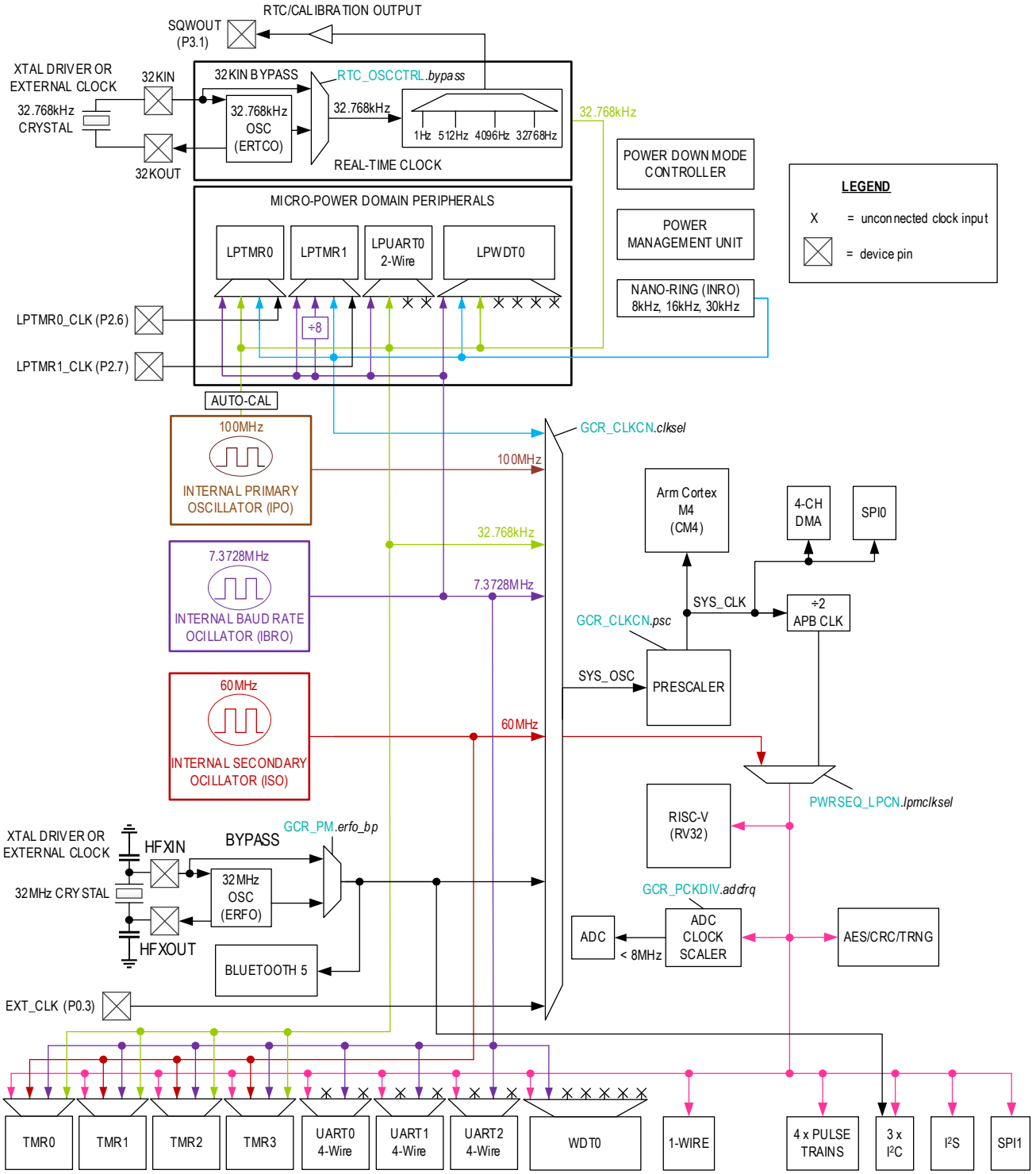
	Reset Source			
Oscillator	**POR**	**System**	**Soft**	**Peripheral**
IPO	Disabled	Disabled	Retains State	Retains State
ISO	Enabled	Enabled	Retains State	Retains State
ERFO				
IBRO	Enabled	Enabled	Enabled	Enabled
INRO	Enabled	Enabled	Enabled	Enabled
ERTCO	Disabled	Retains State	Retains State	Retains State

Table 3-3: Reset Sources and Effect on System Oscillator Selection and Prescaler

	Reset Source				
Clock Field	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
System Oscillator *GCR_CLKCTRL.sysclk_sel*	0 (IPO)	0 (IPO)	0 (IPO)	Retains State	Retains State
System Clock Prescaler *GCR_CLKCTRL.sysclk_div*	1	1	1	Retains State	Retains State

Figure 3-2 shows a high-level diagram of the MAX32655 clock tree.

Figure 3-2: MAX32655 Clock Block Diagram



3.3 Operating Modes

The MAX32655 includes multiple operating modes and the ability to fine tune power options to optimize performance and power. The system supports the following operating modes:

- *ACTIVE*
- *SLEEP*
- *LOW-POWER Mode* (*LPM*)
- *MICRO POWER Mode* (*UPM*)
- *STANDBY*
- *BACKUP*
- *POWER DOWN Mode* (*PDM*)

3.3.1 ACTIVE Mode

In this mode, both the CM4 and RV32 cores can execute application code, and all digital and analog peripherals are available on demand. Dynamic clocking disables peripheral not in use, providing the optimal mix of high performance and low-power consumption. The CM4 has access to all system RAM by default. The RV32 has access to *sysram2* and *sysram3*, and optionally can be configured to have exclusive access to these RAMs. Additionally, *sysram3* can be configured as a unified internal cache controller for the RV32, allowing simultaneous data access and code execution for the CM4 and RV32 from the internal flash memory.

3.3.2 SLEEP Mode

This mode consumes less power but wakes faster because the clocks can optionally be enabled.

The device status is as follows:

- The CM4 (CPU0) is sleeping.
- The RV32 (CPU1) is sleeping.
- Standard DMA is available for use.
- All peripherals are on unless explicitly disabled prior to entering *SLEEP*.

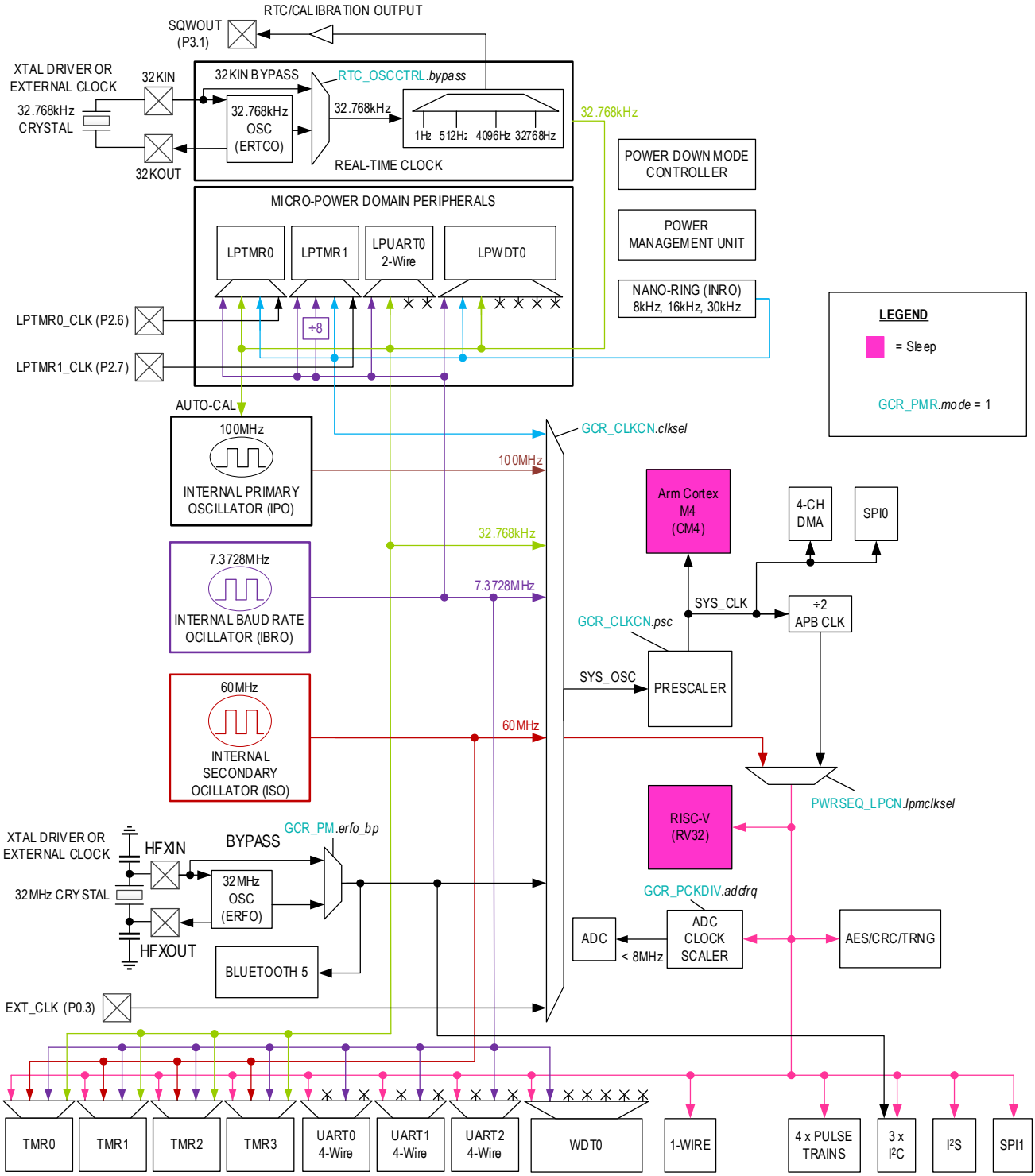
3.3.2.1 Entering SLEEP

Entering *SLEEP* requires both the CM4 and RV32 to cooperate to enter *SLEEP* mode. Synchronization is necessary for deterministic entry into *SLEEP*. Two methods are described below, allowing either core to request entry into *SLEEP*. Both methods use the Semaphore peripheral interrupt to communicate between the cores.

If the RV32 is driving entry to *SLEEP*, the RV32 notifies the CM4 of a request to enter *SLEEP* using *Multiprocessor Communications*. The CM4 receives the notification and then sends a confirmation through the Semaphore peripheral to the RV32. The CM4 should then enter *SLEEP* by setting SCR.*sleepdeep* to 0 and performing a WFI or WFE instruction. The RV32 should then enter *SLEEP* by performing a `WFI` instruction or by setting *GCR_PM*.*mode* to 1, followed by two NOP instructions.

Alternatively, the CM4 can initiate the request to enter *SLEEP* by sending the request to the RV32 using *Multiprocessor Communications*. The RV32 confirms the request through *Multiprocessor Communications* and performs a WFI instruction followed by two NOP instructions. The CM4 should then enter *SLEEP* by setting SCR.*sleepdeep* to 0 and performing a WFI/WFE instruction or by setting the *GCR_PM*.*mode* = 1.

Figure 3-3: SLEEP Mode Clock Control



3.3.3 Low-Power Mode (LPM)

This mode is suitable for running the RV32 processor to collect and move data from enabled peripherals. The device status in *LPM* is:

- The CM4, *sysram0*, and *sysram1* are in state retention.
- The RV32 can access the SPI, all UARTS, all timers, I^2C, 1-Wire, the pulse train engine, I^2S, CRC, AES, TRNG, the comparators, as well as *sysram2* and *sysram3*. *sysram3* can be configured to operate as the RV32 unified instruction cache.
- The transition from *LPM* to *ACTIVE* is faster than the transition from *BACKUP* to *ACTIVE* because system initialization is not required.
- The DMA can access flash.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- Choose the system PCLK or ISO as the clock source for the RV32 and all peripherals.
  - *PWRSEQ_LPCN*.*lpmclksel* defaults to use the ISO during LPM, setting this field to 1 uses the PCLK.
- The following oscillators are powered down:
  - IPO
- The following oscillators are enabled:
  - ERFO
  - IBRO
  - ERTCO
  - INRO
  - ISO

3.3.3.1 Entering LPM

Entry into *LPM* should be managed between the two cores using *Multiprocessor Communications* to ensure both cores are in a known state when entering *LPM*.

When the CM4 puts itself into DEEPSLEEP, the device automatically enters LPM and hardware sets the *GCR_PM*.*mode* to *LPM*. To place the CM4 in *LPM* mode, perform the following instructions.

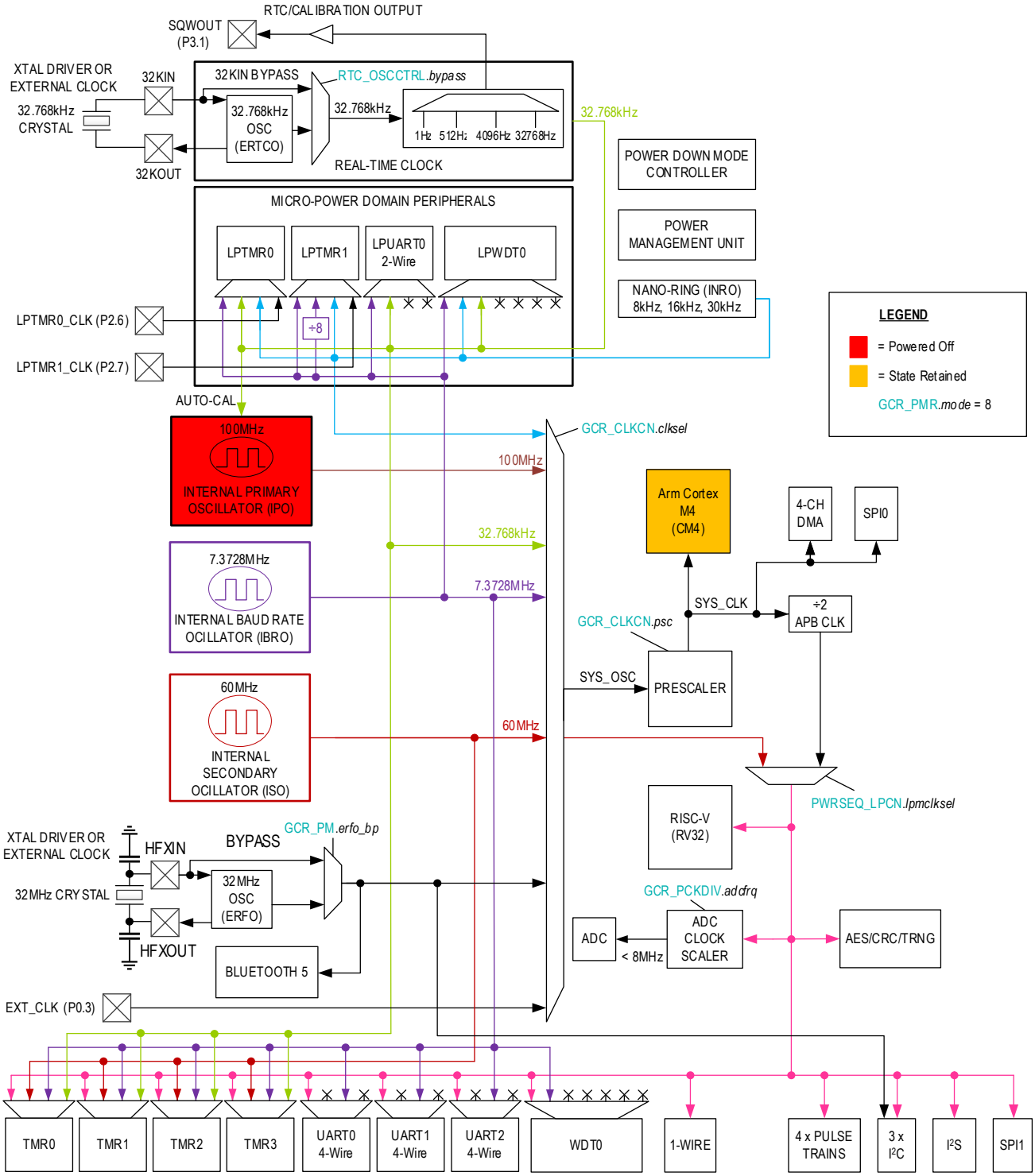
```
SCR.sleepdeep = 1; // DEEPSLEEP mode enabled

WFI (or WFE);      // Enter DEEPSLEEP mode
```

If the RV32 requests the CM4 to enter *LPM* mode through *Multiprocessor Communications* and the CM4 enters *SLEEP* instead, by setting SCR.*sleepdeep* to 0 and performing a WFI or WFE instruction, the RV32 can put the device into *LPM* by directly setting the *GCR_PM*.*mode* field to *LPM* (8).

Note: The device immediately enters LPM when the GCR_PM.mode field is set to LPM, if the CM4 is not in a known state, issues may occur when exiting LPM.

Figure 3-4: Low-Power Mode (LPM) Clock and State Retention Diagram



3.3.4 Micro Power Mode (UPM)

This mode is used for extremely low-power consumption while using a minimal set of peripherals to provide wakeup capability. The device status during *UPM* is:

- Both CM4 and RV32 are state retained.
- System state and all system RAM are retained.
- The GPIO pins retain their state.
- All non-Micro Power peripherals are state retained.
- The GPIO pins retain their state.
- All non-*UPM* peripherals are state retained.
- The following oscillators are powered down:
  - IPO
  - ISO
  - ERFO
- The following oscillators are enabled:
  - IBRO
  - ERTCO
  - INRO
- The following *UPM* peripherals are available for use to wakeup the device:
  - LPUART0 (UART3)
  - LPTMR0 (TMR4)
  - LPTMR1 (TMR5)
  - LPWDT0 (WDT1)
  - COMP0 and LPCOMP1-LPCOMP3
  - GPIO

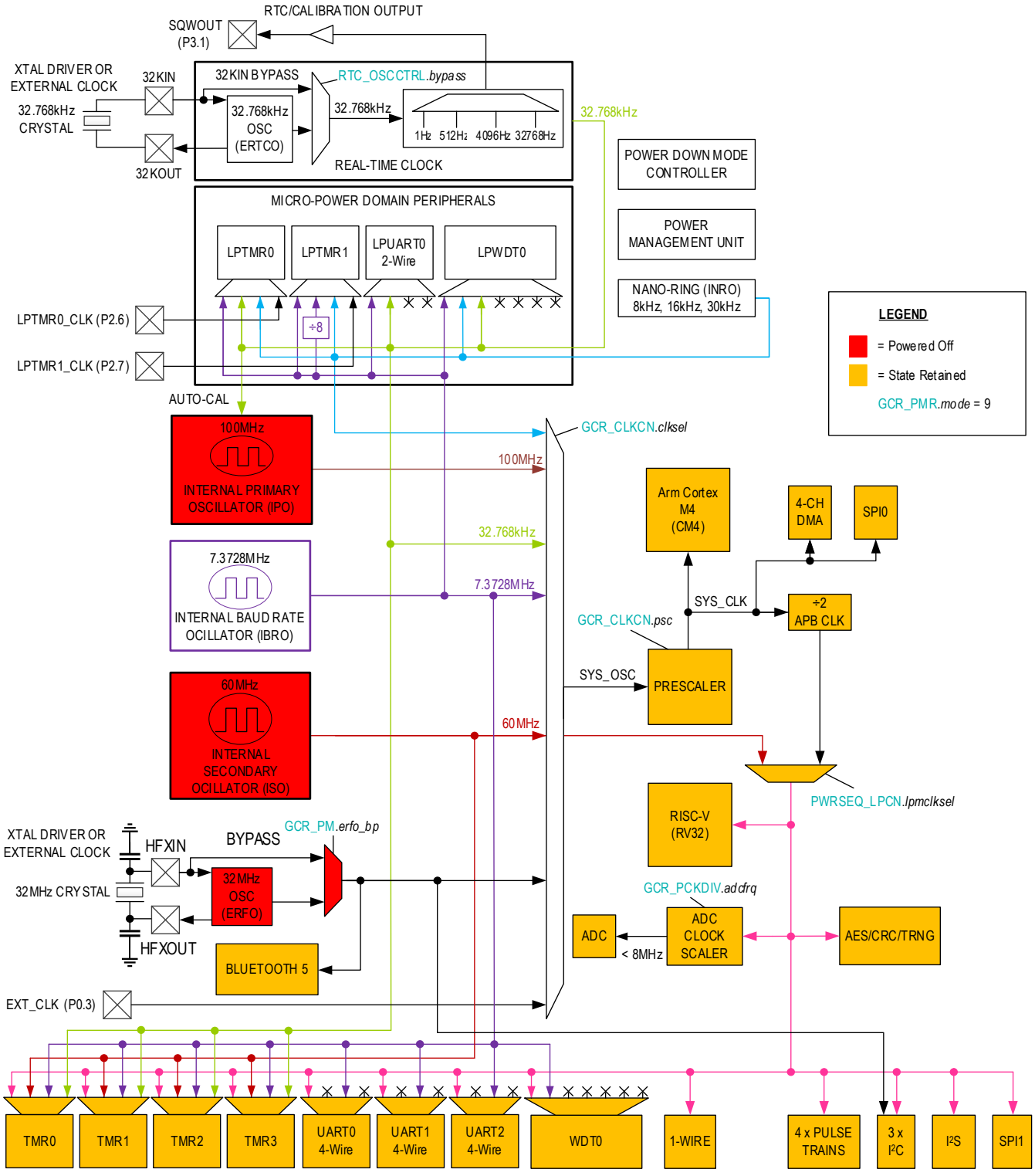
3.3.4.1 Entering UPM

Entering *UPM* mode requires both the CM4 and RV32 to cooperate. Synchronization is necessary for deterministic entry into *UPM*. Two methods are described below, allowing either core to request entry into *UPM* and ensuring deterministic entry. Both methods use the Semaphore peripheral interrupt to communicate between the cores.

If the RV32 is driving entry to *UPM*, the RV32 notifies the CM4 of a request to enter *UPM* using *Multiprocessor Communications*. The CM4 receives the notification and then sends a confirmation through the Semaphore peripheral to the RV32. The CM4 should then enter *SLEEP* by setting SCR.*sleepdeep* to 0 and performing a WFI or WFE instruction. The RV32 sets the *GCR_PM*.*mode* to *UPM*, followed by two NOP instructions and the device immediately enters into *UPM*.

Alternatively, the CM4 can initiate the request to enter *UPM* by sending the request to the RV32 using *Multiprocessor Communications*. The RV32 confirms the request through *Multiprocessor Communications* and performs a WFI instruction followed by two NOP instructions. The CM4 the sets the *GCR_PM*.*mode* to *UPM* and the device immediately enters *UPM*.

Figure 3-5: Micro Power Mode (UPM) Clock and State Retention Block Diagram



3.3.5 STANDBY Mode

This mode maintains the system operation while keeping time with the RTC. The device status in *STANDBY* is as follows:

- Both the CM4 and the RV32 are state retained.
- System state and all saystem RAM are retained.
- GPIO pins retain their state.
- All peripherals retain state.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- The following oscillators are powered down:
  - IPO
  - ISO
  - ERFO
- The following oscillators are enabled:
  - ERTCO
  - INRO
  - IBRO

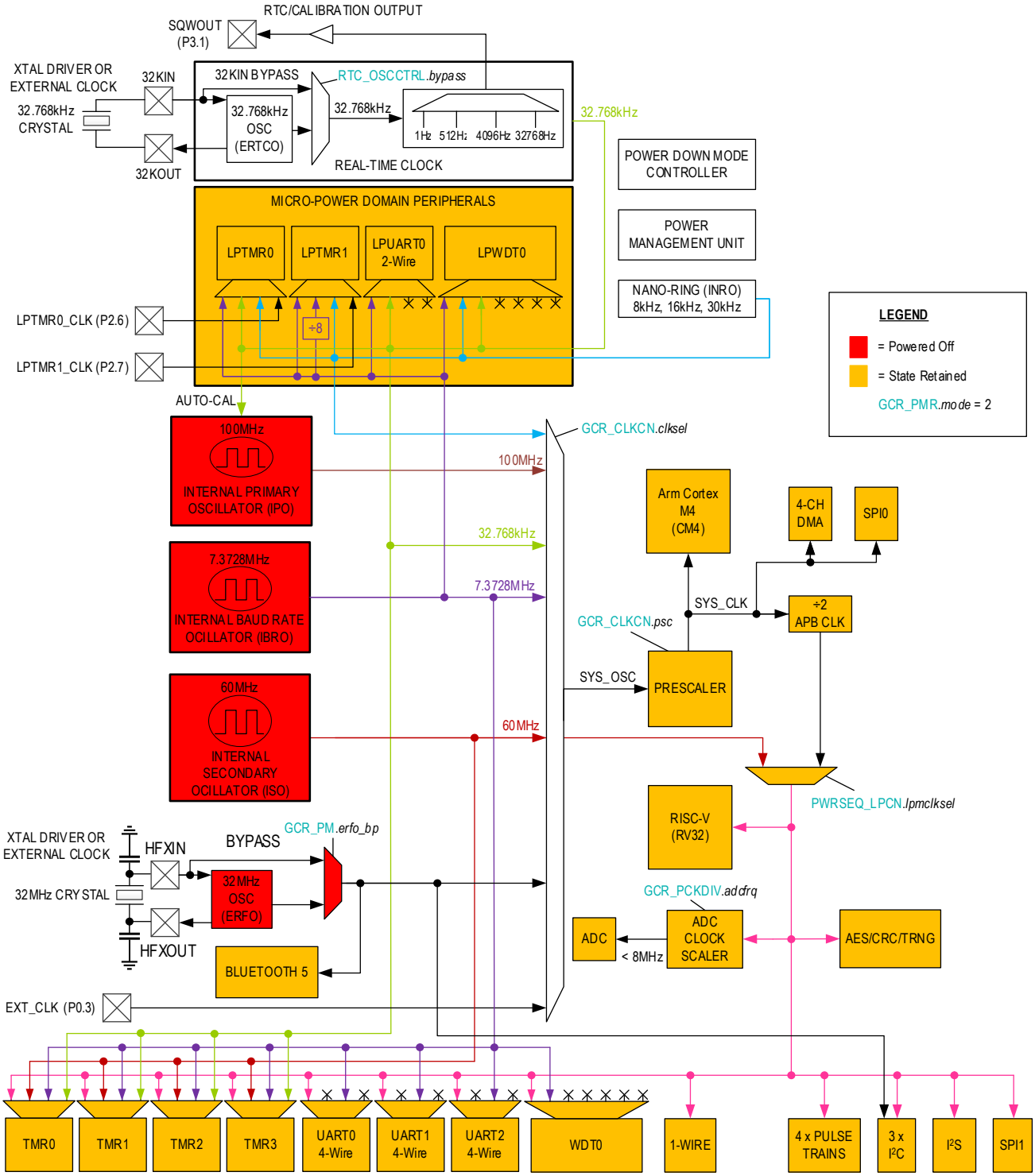
3.3.5.1 Entering STANDBY

Entering *STANDBY* mode requires both the CM4 and RV32 to enter *STANDBY* mode. Synchronization is necessary for deterministic entry into *STANDBY*. Two methods are described below, allowing either core to request entry into *STANDBY* and ensuring deterministic entry. Both methods use the Semaphore peripheral interrupt to communicate between the cores.

If the RV32 is driving entry to *STANDBY*, the RV32 notifies the CM4 of a request to enter *STANDBY* using *Multiprocessor Communications*. The CM4 receives the notification and then sends a confirmation through the Semaphore peripheral to the RV32. The CM4 should then enter *SLEEP* by setting SCR.*sleepdeep* to 0 and performing a WFI or WFE instruction. The RV32 sets the *GCR_PM*.*mode* to *STANDBY*, followed by two NOP instructions and the device immediately enters into *STANDBY*.

Alternatively, the CM4 can initiate the request to enter *STANDBY* by sending the request to the RV32 using *Multiprocessor Communications*. The RV32 confirms the request through *Multiprocessor Communications* and performs a WFI instruction followed by two NOP instructions. The CM4 then sets the *GCR_PM*.*mode* to *STANDBY* and the device immediately enters *STANDBY*.

Figure 3-6: STANDBY Mode Clock and State Retention Block Diagram



3.3.6 BACKUP Mode

This mode maintains the system RAM contents. The device status in *BACKUP* is as follows:

- CM4 and RV32 are powered off.
- *sysram0*, *sysram1*, *sysram2*, and *sysram3* can be independently configured to be state retained as shown in *Table 3-4*.
- All peripherals are powered off.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
  - INRO
  - ERFO
- The following oscillators are enabled:
  - ERTCO (The RTC peripheral can be turned off, but not the oscillator.)

Table 3-4 System RAM Retention in BACKUP Mode

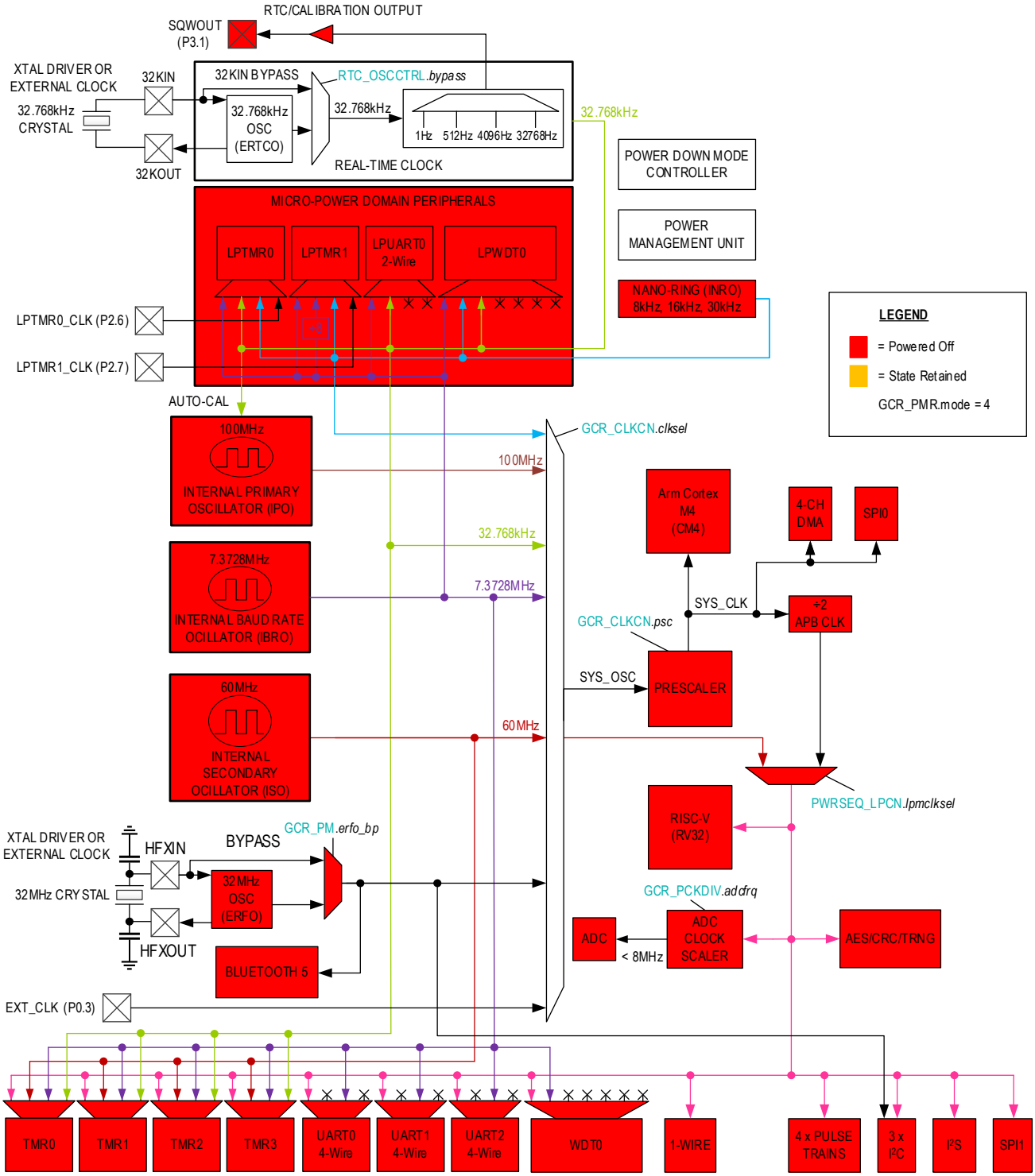
RAM Block #	Size	State Retention Control
sysram0	*32KB + ECC if enabled*	*PWRSEQ_LPCN.ram0*
sysram1	*32KB*	*PWRSEQ_LPCN.ram1*
sysram2	*48KB*	*PWRSEQ_LPCN.ram2*
sysram3	*16KB*	*PWRSEQ_LPCN.ram3*

3.3.6.1 Entering BACKUP

Entering *BACKUP* mode does not require synchronization between the RV32 and CM4 cores. However, it is recommended that *Multiprocessor Communications* is used to ensure both cores are aware of entry into BACKUP mode and complete any memory transactions prior to entry.

Either core can set *GCR_PM.mode* to *BACKUP* and the device immediately enters *BACKUP*.

Figure 3-7: BACKUP Mode Clock and State Retention Block Diagram



3.3.7 Power Down Mode (PDM)

This mode is used during a customer's product level distribution and storage. The device status in *PDM* is as follows:

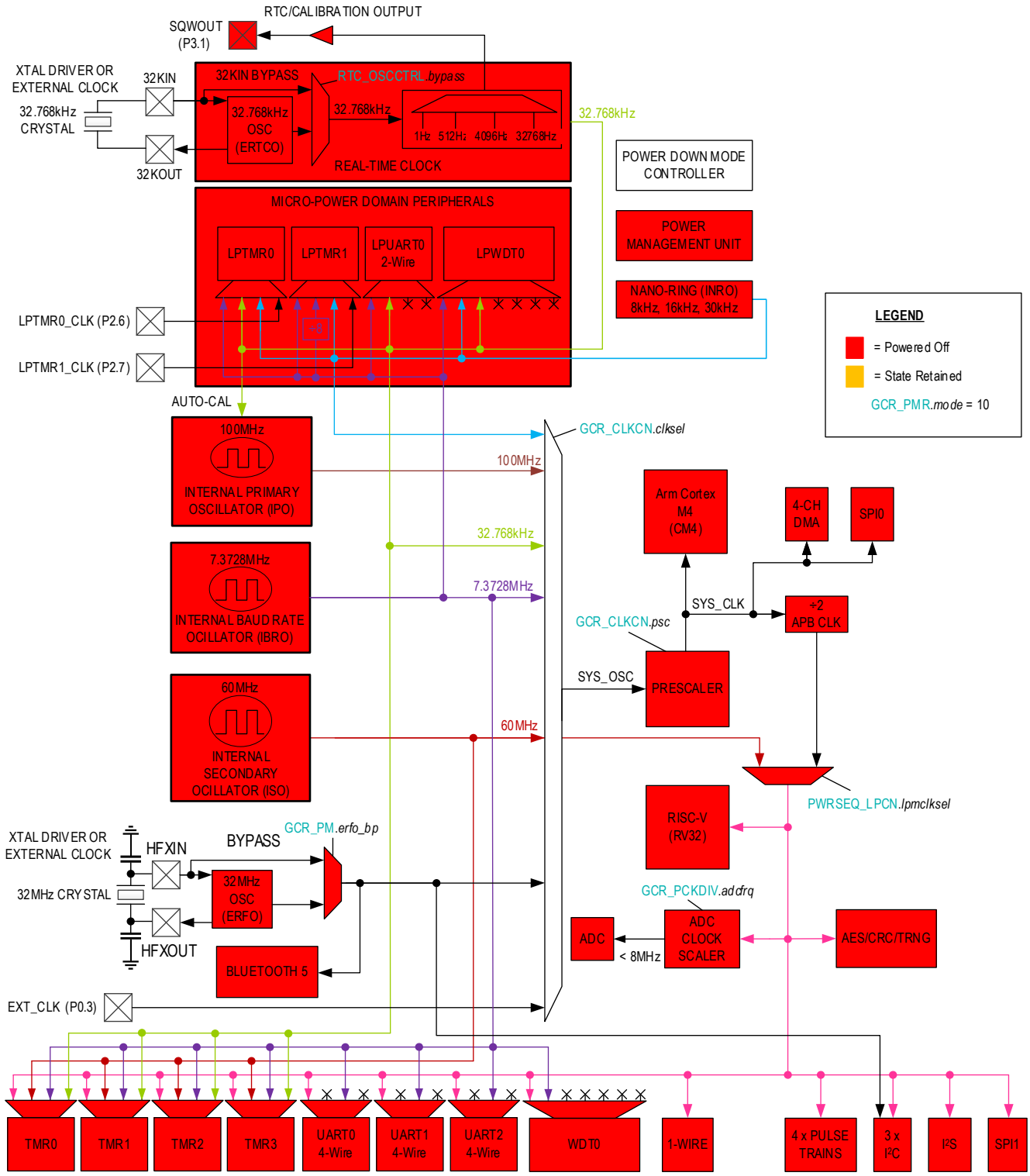
- The CM4 and RV32 are powered off.
- All peripherals and system RAM are powered down.
- All oscillators are powered down.
- There is no data retention in this mode, but values in the flash are preserved.
- V_{REGI} POR voltage monitor is operational.
- Exit from PDM is possible through an external reset (RSTN) or a wakeup event on using either P3.0 or P3.1 if configured.

3.3.7.1 Entering PDM

Entering *PDM* does not require synchronization between the RV32 and CM4 cores. However, it is recommended that *Multiprocessor Communications* is used to ensure both cores are aware of entry into *PDM* and complete any flash memory transactions.

Either core can set *GCR_PM*.*mode* to *PDM* and the device immediately enters *PDM*.

Figure 3-8: Power Down Mode (PDM) Clock and State Retention Block Diagram



3.3.8 Operating Modes Wakeup Sources

In all operating modes other than *ACTIVE*, wakeup sources are required to resume *ACTIVE* operation. *Table 3-5* shows available wakeup sources for each operating mode of the MAX32655.

Note: Each wakeup source must be enabled individually except for the External Reset (RSTN), which is hardware controlled.

Table 3-5: Wakeup Sources for Each Operating Mode in the MAX32655

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	BTLE	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCOMP3	LPCOMP2	LPCOMP1	COMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓									✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																								✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																								✓	✓	✓	✓	✓	✓	✓
PDM		✓																											✓			

3.4 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset
- Power-On Reset

On completion of any of the four reset cycles, all peripherals are reset. On completion of any reset cycle, HCLK and PCLK are operational, the CPU core receives clocks and power, and the device is in *ACTIVE* mode. Program execution begins at the reset vector address.

Contents of the Always-On Domain (AoD) are reset only on power-cycling V_{COREA}, V_{COREB}, V_{DDA}, V_{DDIOH}, or V_{REGI}.

Each of the on-chip peripherals can also be reset to their POR default state using the two reset registers *GCR_RST0* and *GCR_RST1*.

Table 3-6 shows the effects of the four reset types and seven power modes.

Table 3-6: Reset and Low-Power Mode Effects

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	ACTIVE	SLEEP	LPM	UPM	BACKUP[3]	PDM
IPO	-	-	Off	Off	R	-	Off	Off	Off	Off
ISO	-	-	On	Off	R	-	-		-	-
ERTCO	-	-	-	Off	FW	FW	FW		FW	FW
IBRO	-	-	Off	Off	R	-	Off		Off	Off
ERFO	-	-	Off	Off	R	-	Off		Off	Off

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	ACTIVE	SLEEP	LPM	UPM	BACKUP[3]	PDM
INRO	On	On	On	On	On	On	On		On	On
SYS_CLK	On	On	On[2]	On[2]	On	On	Off		Off	Off
CPU Clock	On	On	On	On	On	Off	Off		Off	Off
RTC				Reset	FW	FW	FW		FW	FW
WDT0,WDT1	-	Reset	Reset	Reset	FW	Off	Off		Off	Off
GPIO0-GPIO3	-	Reset	Reset	Reset	R	-	-		-	-
All Other Peripherals	Reset	Reset	Reset	Reset	R	-	Off		Off	Off
Always-On Domain	-	-	-	Reset	-	-	-		-	-
RAM Retention	-	-	-	Reset	-	-	On		FW	Off

Table key:
FW = Controlled by firmware
On = Enabled by hardware (Cannot be disabled)
Off = Disabled by hardware (Cannot be enabled)
- = No Effect
R = Restored to previous *ACTIVE* mode setting when exiting *LPM*, restored to system reset state when exiting *BACKUP* or *STORAGE*.

1: The always-on domain (AoD) is only reset on power-cycling V_{COREA}, V_{COREB}, V_{DDA}, V_{DDIOH}, or V_{REGI}.
2: On a system reset or POR, the ISO is automatically set as the SYS_OSC.
3: A system reset occurs when returning from *BACKUP* or *PDM* low-power mode.
4: Peripheral, soft and system resets are initiated by software though the *GCR_RST0* register. System Reset can also be triggered by the RSTN device pin or a Watchdog reset.

3.4.1 Peripheral Reset

This resets all peripherals. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, and general control registers (GCR), including the clock configuration, are unaffected.

To start a peripheral reset, set *GCR_RST0*.*periph* = 1. The reset is completed immediately upon setting *GCR_RST0*.*periph* = 1.

3.4.2 Soft Reset

This is the same as a peripheral reset except that it also resets the GPIO to its POR state.

To start a soft reset, set *GCR_RST0*.*soft* = 1. The reset is completed immediately upon setting *GCR_RST0*.*soft* = 1.

3.4.3 System Reset

This is the same as soft reset except it also resets all GCR, resetting the clocks to their default state. The CPU state is reset as well as the watchdog timers. The AoD and RAM are unaffected.

A watchdog timer reset event initiates a system reset. To start a system reset from software, set *GCR_RST0*.*sys* = 1.

3.4.4 Power-On Reset

A POR resets everything in the device to its default state.

3.5 Unified Internal Cache Controllers (ICC)

The MAX32655 includes two unified internal cache controllers. ICC0 is the cache controller used for the CM4. ICC1, if enabled as a cache controller, is dedicated to RV32 core. ICC1 uses *sysram3* as the cache memory. If ICC1 is enabled, *sysram3* is not accessible as data RAM (address range 0x2001 C000 to 0x2001 FFFF).

Both caches, ICC0 and ICC1, include a line buffer, tag RAM and a 16KB 2-way set associative RAM when enabled.

3.5.1 Enabling the Instruction Cache Controller

Enabling ICC1 for use as the cache controller for the RV32 requires using *sysram3* as the cache memory.

Note: The contents of sysram3 are lost when ICC1 is enabled and sysram3 is not accessible for data reads or writes as part of the memory map.

Perform the following steps to enable ICCn:

1. Set the *ICCn_CTRL.en* to 0, ensuring the cache is invalidated when enabled.
2. Set *ICCn_CTRL.en* to 1.
3. Read *ICCn_CTRL.rdy* until it returns 1.

3.5.2 Disabling the ICC

Disable an ICC instance by setting *ICCn_CTRL.en* to 0.

To use *sysram3* as data RAM, first disable the ICC1 instance as described above. When ICC1 is disabled, *sysram3* is accessible as data RAM by both the CM4 and RV32 controllers unless *sysram3* is configured for exclusive access by the RV32 core only.

3.5.3 Invalidating the ICC Cache and Tag RAM

The system configuration register (*GCR_SYSCTRL*) includes a field for flushing the ICC0 cache. Setting *GCR_SYSCTRL.icc0_flush* to 1 flushes the ICC0 16KB cache and the tag RAM.

Invalidate the contents of a specific ICC instance by setting the *ICCn_INVALIDATE* register to 1. Once invalidated, the system flushes the cache. Read the *ICCn_CTRL.rdy* field until it returns 1 to determine when the flush is completed.

3.5.4 Flushing the ICC

Flush ICC0 using the *GCR_SYSCTRL* register. Set *GCR_SYSCTRL.icc0_flush* to 1 to immediately flush the contents of the 16KB cache and tag RAM.

Flush ICC1 using the RV32 control register (*FCR_URVCTRL*). Set *FCR_URVCTRL.icc1_flush* to 1 to immediately flush the contents of the 16KB cache and tag RAM.

3.5.5 Internal Cache Control Registers (ICC)

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-7: Internal Cache Controller Register Summary

Offset	Register	Name
[0x0000]	*ICCn_INFO*	*Cache ID Register*
[0x0004]	*ICCn_SZ*	*Cache Memory Size Register*
[0x0100]	*ICCn_CTRL*	*Instruction Cache Control Register*
[0x0700]	*ICCn_INVALIDATE*	*Instruction Cache Controller Invalidate Register*

3.5.6 ICC0 Register Details

Table 3-8: ICC0 Cache Information Register

ICC0 Cache Information			ICCn_INFO	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:10	id	R	-	**Cache ID** Returns the ID for this cache instance.
9:6	partnum	R	-	**Cache Part Number** Returns the part number indicator for this cache instance.
5:0	relnum	R	-	**Cache Release Number** Returns the release number for this cache instance.

Table 3-9: ICC0 Memory Size Register

ICC0 Memory Size			ICCn_SZ	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	mem	R	-	**Addressable Memory Size** Indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cch	R	-	**Cache Size** Returns the size of the cache RAM memory in 1KB units. 16: 16KB Cache RAM

Table 3-10: ICC0 Cache Control Register

ICC0 Cache Control			ICCn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved**
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). The hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready. 0: Cache invalidation in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.*
15:1	-	R/W	-	**Reserved**
0	en	R/W	0	**Cache Enable** Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents and reads are handled by the line fill buffer. 0: Disable 1: Enable

Table 3-11: ICC0 Invalidate Register

ICC0 Invalidate			ICCn_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	invalid	W	-	**Invalidate** Writing any value to this register invalidates the cache.

3.6 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, instruction, and data cache (ICC0) for the CM4, and instruction and data cache (ICC1) for the RV32, and peripheral FIFOs.

3.6.1 On-Chip Cache Management

The MAX32655 includes two instruction cache controllers for code fetches from the flash memory. The caches can be enabled, disabled, and zeroized or flushed. See *Low-Power General Control Registers Details* for details.

3.6.2 RAM Zeroization

The GCR memory zeroize register, *GCR_MEMZ*, clears memory for software or security reasons. Zeroization writes all zeros to the specified memory.

The following SRAM memories can be zeroized:

- Each of the system RAMs can be individually zeroized by setting the respective *GCR_MEMZ* bit.
  - *GCR_MEMZ.ram0*
  - *GCR_MEMZ.ram0ecc*
  - *GCR_MEMZ.ram1*
  - *GCR_MEMZ.ram2*
  - *GCR_MEMZ.ram3*
- ICC0 16KB Cache
  - Enabling ICC0 from a disabled state zeroizes the cache RAM.
- ICC1 16KB Cache, if enabled
  - Write 1 to *GCR_MEMZ.icc1*

The system SRAM banks are shown with corresponding bank sizes, base address and ending addresses in *Table 3-12*.

Table 3-12 RAM Block Size and Base Address

System RAM Block #	Size	Base Address	End Address
sysram0	*32KB*	*2000 0000*	*2000 7FFF*
sysram1	*32KB*	*2000 8000*	*2000 FFFF*
sysram2	*48KB*	*2001 0000*	*2001 BFFF*
sysram3	*16KB*	*2001 C000*	*2001 FFFF*

3.7 Miscellaneous Control Registers (MCR)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-13: Miscellaneous Control Register Summary

Offset	Register Name	Access	Description
[0x0000]	*MCR_ECCEN*	R/W	Error Correction Coding Enable Register
[0x0004]	*MCR_IPO_MTRIM*	R/W	IPO Manual Trim Register
[0x0008]	*MCR_OUTEN*	R/W	Miscellaneous Output Enable Register
[0x000C]	*MCR_CMP0_CTRL*	R/W	Comparator 0 Control Register
[0x0010]	*MCR_CTRL*	R/W	Miscellaneous Control Register
[0x0020]	*MCR_GPIO3_CTRL*	R/W	GPIO3 Pin Control Register

3.7.1 Miscellaneous Control Register Details

Table 3-14: Error Correction Coding Enable Register

Error Correction Coding Enable			MCR_ECCEN	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	ram0	R/W	0	**System RAM 0 ECC Enable** Set this field to 1 to enable ECC for *sysram0*. 0: Disabled 1: Enabled

Table 3-15: IPO Manual Register

IPO Manual Trim			MCR_IPO_MTRIM	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	trim_range	R/W	0	**Trim Range Select** If this bit is set to 1, the value loaded into *MCR_IPO_MTRIM.mtrim* must be greater than the trim setting in *TRIMSIR_IPOLO.ipo_limitlo*. If this bit is set to 0, the value loaded into *MCR_IPO_MTRIM.mtrim* must be less than the trim setting in *TRIMSIR_CTRL.ipo_limithi*. 0: *MCR_IPO_MTRIM.mtrim* < *TRIMSIR_IPOLO.ipo_limitlo* 1: *MCR_IPO_MTRIM.mtrim* > *TRIMSIR_CTRL.ipo_limithi*
7:0	mtrim	R/W	4	**Manual Trim Value** Set this value to the desired manual trim based on the value set in *MCR_IPO_MTRIM.trim_range*. If *MCR_IPO_MTRIM.trim_range* is 0, the value in this field must be less than the value in *TRIMSIR_IPOLO.ipo_limitlo*. If *MCR_IPO_MTRIM.trim_range* is 1, the value in this field must be greater than the value in *TRIMSIR_CTRL.ipo_limithi*.

Table 3-16: Output Enable Register

Output Enable			MCR_OUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	pdown_out	R/W	0	**Power Down Output Enable on P3.0** Set this field to 1 to enable the power down output, P3.0 AF1 (PDOWN). PDOWN is active in *BACKUP* and *STANDBY*. 0: PDOWN output not enabled on P3.0 1: PDOWN output is enabled on P3.0
0	sqwout	R/W	0	**Square Wave Output Enable on P3.1 (SQWOUT)** Set this field to 1 to enable the square wave output on P3.1 AF1 (SQWOUT). 0: Square wave output not enabled on P3.1 1: Square wave output enabled on P3.1

Table 3-17: Comparator 0 Control Register

Comparator 0 Control			MCR_CMP0_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

Comparator 0 Control			MCR_CMP0_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	if	R/W1C	0	**Comparator 0 Interrupt Flag** This field is set to 1 by hardware when the comparator output changes to the active state as set using the *MCR_CMP0_CTRL.pol* field. Write 1 to clear this flag. 0: No interrupt 1: Interrupt occurred
14	out	RO	*	**Comparator 0 Output** This field is the comparator output state. 0: Output low 1: Output high
13:7	-	RO	0	**Reserved**
6	int_en	R/W	0	**Comparator 0 Interrupt Enable** Set this field to 1 to enable the IRQ for comparator 3. 0: Interrupt disabled 1: Interrupt enabled
5	pol	R/W	0	**Comparator 0 Interrupt Polarity Select** Set this field to select the polarity of the output change that generates a comparator 3 interrupt. 0: Interrupt occurs from a transition from low to high. 1: Interrupt occurs from a transition from high to low.
4:1	-	RO	0	**Reserved**
0	en	R/W	0	**Comparator 0 Enable** Set this field to 1 to enable the comparator 0: Comparator disabled 1: Comparator enable

Table 3-18: Miscellaneous Control Register

Miscellaneous Control			MCR_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	simo_rstd	R/W	0	**SIMO System Reset Disable** If this field is set, the SIMO is only reset by a Power-On Reset. When this bit is set, the VSET* stay's unchanged when exiting all low-power modes. 0: The SIMO is reset by all system resets. 1: The SIMO is only reset by a Power-On Reset.
8	simo_clkscl	R/W	0	**SIMO Clock Scaling Enable** Set this field to 1 to enable dynamic clock scaling to the SIMO based on load current. When enabled, the SIMO clock slows down in low-power modes, reducing current consumption. 0: SIMO clock scaling disabled 1: SIMO clock scaling enabled
7:4	-	DNM	1	**Reserved, Do Not Modify**
3	ertco	R/W	0	**ERTCO Enable Low-Power Modes** Set this field to 1 to enable the ERTCO in *UPM*, *STANDBY* and *BACKUP*. 0: ERTCO disabled in *UPM*, *STANDBY* and *BACKUP* 1: ERTCO enabled in *UPM*, *STANDBY* and *BACKUP*

Miscellaneous Control			MCR_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
2	inro	R/W	0	**INRO Enable** Set this field to 1 to enable the INRO. 0: INRO disabled 1: INRO enabled
1:0	-	RO	0	**Reserved**

Table 3-19: GPIO3 Pin Control Register

GPIO3 Pin Control			MCR_GPIO3_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	p31_in	RO	See Description	**GPIO3 Pin 1 Input Status** Read this field to determine the input status of P3.1. 0: Input low 1: Input high
6	p31_pe	R/W	0	**GPIO3 Pin 1 Pullup Enable** Set this bit to 1 to enable the pullup resistor for P3.1 0: Disabled 1: Enabled
5	p31_oe	R/W	0	**GPIO3 Pin 1 Output Enable** Set this bit to 1 to enable P3.1 for output mode. 0: Input mode. 1: Output mode.
4	p31_do	R/W	0	**GPIO3 Pin 1 Data Output** If *p31_oe* is set to 1, this field is used to control the output state of P3.1. 0: Output low if *p31_oe* is 1. 1: Output high if *p31_oe* is 1.
3	p30_in	RO	See Description	**GPIO3 Pin 0 Input Status** Read this field to determine the input status of P3.0. 0: Input low 1: Input high
2	p30_pe	R/W	0	**GPIO3 Pin 0 Pullup Enable** Set this bit to 1 to enable the pullup resistor for P3.0 0: Disabled. 1: Enabled.
1	p30_oe	R/W	0	**GPIO3 Pin 0 Output Enable** Set this bit to 1 to enable P3.0 for output mode. 0: Input mode. 1: Output mode enabled.
0	p30_do	R/W	0	**GPIO3 Pin 0 Data Output** If *MCR_GPIO3_CTRL.p30_oe* is set to 1, this field is used to control the output state of P3.0. 0: Output low if the pin is enabled (*MCR_GPIO3_CTRL.p30_oe* is 1). 1: Output high if the pin is enabled (*MCR_GPIO3_CTRL.p30_oe* is 1).

3.8 Single Inductor Multiple Output Power Supply (SIMO)

The single inductor multiple output (SIMO) switch-mode power supply allows the device to operate autonomously from a single lithium cell. The SIMO provides four buck switching regulators (V_{REGO_A} thru V_{REGO_D}). Each of the four regulator

voltages can be controlled by the CPU individually. For the SIMO to operate properly, the four buck regulator outputs must drive the power supply pins of the device as follows in *Table 3-20*.

3.8.1 Power Supply Monitor

The system also provides a power monitor that monitors the external power supplies relative to the on-chip bandgap voltage. The following power supplies are monitored:

- V_{COREA} Digital Core Supply Voltage A for the Always-On Domain
- V_{COREB} Digital Core Supply Voltage B
- V_{DDIO} GPIO Supply Voltage
- V_{DDIOH} GPIO High Supply Voltage
- V_{DDA} Always-On Domain Analog Supply Voltage
- V_{REGI} Input Supply Voltage, Battery
- BLE_LDO_IN Bluetooth Transceiver Supply Voltage Input

Each of the power supply monitors' settings are found in the low-power control register *PWRSEQ_LPCN*. When the corresponding power monitor is enabled, the input voltage pin is constantly monitored. If the voltage drops below the trigger threshold, all registers and peripherals in that power domain are reset. This improves reliability and safety by guarding against a low-voltage condition corrupting the contents of the registers and device state.

Refer to the data sheet electrical characteristics for the trigger threshold values and power fail reset voltages.

Table 3-20: SIMO Power Supply Device Pin Connectivity

SIMO Supply Output Pin	Connection	Device Power Supply Input Pin	Supply Monitor Reset Action
V_{REGO_A}	→	V_{DDA}	Domain reset
V_{REGO_B}	→	V_{COREB}	Domain Reset
V_{REGO_C}	→	V_{COREA}	Domain Reset
V_{REGO_D}	→	BLE_LDO_IN	Bluetooth POR
-	-	V_{REGI}	Domain Reset
-	-	V_{DDIO} Power On	GPIO pad held in reset until the voltage rises above threshold.
-	-	V_{DDIOH} Power On	GPIO pad held in reset until the voltage rises above threshold.
-	-	V_{DDIO}	GPIO pad logic enters POR.
-	-	V_{DDIOH}	GPIO pad logic enters POR.

3.8.2 Single Inductor Multiple Output Registers (SIMO)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-21: SIMO Controller Register Summary

Offset	Register	Access	Name
[0x0004]	*SIMO_VREGO_A*	*R/W*	*Buck Voltage Regulator A Control Register*
[0x0008]	*SIMO_VREGO_B*	*R/W*	*Buck Voltage Regulator B Control Register*
[0x000C]	*SIMO_VREGO_C*	*R/W*	*Buck Voltage Regulator C Control Register*
[0x0010]	*SIMO_VREGO_D*	*R/W*	*Buck Voltage Regulator D Control Register*
[0x0014]	*SIMO_IPKA*	*RO*	*Reserved. Do not modify this register.*
[0x0018]	*SIMO_IPKB*	*RO*	*Reserved. Do not modify this register.*
[0x001C]	*SIMO_MAXTON*	*RO*	*Reserved. Do not modify this register.*

Offset	Register	Access	Name
[0x0020]	*SIMO_ILOAD_A*	*RO*	*Reserved. Do not modify this register.*
[0x0024]	*SIMO_ILOAD_B*	*RO*	*Reserved. Do not modify this register.*
[0x0028]	*SIMO_ILOAD_C*	*RO*	*Reserved. Do not modify this register.*
[0x002C]	*SIMO_ILOAD_D*	*RO*	*Reserved. Do not modify this register.*
[0x0030]	*SIMO_BUCK_ALERT_THR_A*	*RO*	*Reserved. Do not modify this register.*
[0x0034]	*SIMO_BUCK_ALERT_THR_B*	*RO*	*Reserved. Do not modify this register.*
[0x0038]	*SIMO_BUCK_ALERT_THR_C*	*RO*	*Reserved. Do not modify this register.*
[0x003C]	*SIMO_BUCK_ALERT_THR_D*	*RO*	*Reserved. Do not modify this register.*
[0x0040]	*SIMO_BUCK_OUT_READY*	*RO*	*Buck Regulator Output Ready Register*
[0x0044]	*SIMO_ZERO_CROSS_CAL_A*	*RO*	*Reserved. Do not modify this register.*
[0x0048]	*SIMO_ZERO_CROSS_CAL_B*	*RO*	*Reserved. Do not modify this register.*
[0x004C]	*SIMO_ZERO_CROSS_CAL_C*	*RO*	*Reserved. Do not modify this register.*
[0x0050]	*SIMO_ZERO_CROSS_CAL_D*	*RO*	*Reserved. Do not modify this register.*

3.8.3 Single Inductor Multiple Output (SIMO) Registers Details

Table 3-22: SIMO Buck Voltage Regulator A Control Register

SIMO Buck Voltage Regulator A Control		SIMO_VREGO_A		[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved**
7	rangea	R/W	1	**Regulator Output A Range** Selects the Regulator output range for V_{REGO_A}. 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vseta	R/W	0x78h	**Regulator Output A Voltage** Each bit increment in this field represents 10mV allowing output voltage settings from the minimum to the maximum of the *SIMO_VREGO_A.rangea* selected. *SIMO_VREGO_A.rangea* = 1: $\text{Output Voltage} = 0.6\text{V} + (10\text{mV} \times \text{vseta})$ *SIMO_VREGO_A.rangea* = 0: $\text{Output Voltage} = 0.5\text{V} + (10\text{mV} \times \text{vseta})$ Default: 0x78 = *SIMO_VREGO_A.rangea* = 0, *Output Voltage* = 1.7V; *SIMO_VREGO_A.rangea* = 1, *Output Voltage* = 1.8V ***Warning**: When this regulator is connected as shown in Table 3-20, the following apply:* 1. The maximum setting for this regulator must be followed for V_{DDA} as indicated in the device data sheet. 2. Setting the regulator to a voltage below the power-fail reset voltage for V_{DDA} initiates the power monitor reset action.

Table 3-23: SIMO Buck Voltage Regulator B Control Register

SIMO Buck Voltage Regulator B Control		SIMO_VREGO_B		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved**
7	rangeb	R/W	1	**Regulator Output B Range** Selects the Regulator output range for V_{REGO_B}. 0: 0.5V to 1.77 1: 0.6V to 1.87V

SIMO Buck Voltage Regulator B Control			SIMO_VREGO_B	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
6:0	vsetb	R/W	0x32	**Regulator Output Voltage** Each bit increment in this field represents 10mV allowing output voltage settings from the minimum to the maximum of the *SIMO_VREGO_B.rangeb* selected. *SIMO_VREGO_B.rangeb* = 1; $Output\ Voltage = 0.6V + (10mV \times vsetb)$ *SIMO_VREGO_B.rangeb* = 0; $Output\ Voltage = 0.5V + (10mV \times vsetb)$ Setting this field to 0x7F results in the maximum output voltage per the *SIMO_VREGO_B.rangeb* selected (1.77V or 1.87V) Default: 0x32 = *SIMO_VREGO_B.rangeb* = 0, *Output Voltage* = 1.0V; *SIMO_VREGO_B.rangeb* = 1, *Output Voltage* = 1.1V ***Warning**: When this regulator is connected as shown in Table 3-20, the following apply:* 1. The maximum setting for this regulator must be followed for V_{COREB} as indicated in the device data sheet. 2. Setting the regulator to a voltage below the Power-Fail Reset Voltage for V_{COREB} initiates the power monitor reset action.

Table 3-24: SIMO Buck Voltage Regulator C Control Register

SIMO Buck Voltage Regulator C Control			SIMO_VREGO_C	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved**
7	rangec	R/W	1	**Regulator Output Range** Selects the Regulator output range for V_{REGO_C}. 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vsetc	R/W	0x32	**Regulator Output Voltage** Each increment in the register represents 10mV. *SIMO_VREGO_C.rangec* = 1; $Output\ Voltage = 0.6V + (10mV \times vsetc)$ *SIMO_VREGO_C.rangec* = 0; $Output\ Voltage = 0.5V + (10mV \times vsetc)$ Setting this field to 0x7F results in the maximum output voltage per the *SIMO_VREGO_C.rangec* selected (1.77V or 1.87V) Default: 0x32 = *SIMO_VREGO_C.rangec* = 0, *Output Voltage* = 1.0V; *SIMO_VREGO_C.rangec* = 1, *Output Voltage* = 1.1V ***Warning**: When this regulator is connected as shown in Table 3-20, the following apply:* 1. The maximum setting for this regulator must be followed for V_{COREA} as indicated in the device data sheet. 2. Setting the regulator to a voltage below the power-fail reset voltage for V_{COREA} initiates the power monitor reset action.

Table 3-25: SIMO Buck Voltage Regulator D Control Register

SIMO Buck Voltage Regulator D Control			SIMO_VREGO_D	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved**

SIMO Buck Voltage Regulator D Control			SIMO_VREGO_D	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
7	ranged	R/W	1	**Regulator Output Range** Selects the Regulator output range for V_{REGO_D}. 0: 0.5V to 1.77V 1: 0.6V to 1.87V
6:0	vsetd	R/W	0x32	**Regulator Output Voltage** Each increment in the register represents 10mV. *SIMO_VREGO_D.ranged* = 1; $Output\ Voltage = 0.6V + (10mV \times vsetd)$ *SIMO_VREGO_D.ranged* = 0; $Output\ Voltage = 0.5V + (10mV \times vsetd)$ Setting this field to 0x7F results in the maximum output voltage per the *SIMO_VREGO_D.ranged* selected (1.77V or 1.87V) Default: 0x32 = *SIMO_VREGO_D.ranged* = 0, *Output Voltage* = 1.0V; *SIMO_VREGO_D.ranged* = 1, *Output Voltage* = 1.1V ***Warning**: When this regulator is connected as shown inTable 3-20, the following apply:* 1. The maximum setting for this regulator must be followed for BLE_LDO_IN as indicated in the device data sheet. 2. Setting the regulator to a voltage below the power-fail reset voltage for the BLE_LDO_IN will initiate the power monitor reset action.

Table 3-26: SIMO High Side FET Peak Current V_{REGO_A} V_{REGO_B} Register

SIMO High Side FET Peak Current V_{REGO_A} V_{REGO_B}			SIMO_IPKA	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:4	ipksetb	DNM	8	**Reserved** Reserved. Do not modify this field.
3:0	ipkseta	DNM	8	**Reserved** Reserved. Do not modify this field.

Table 3-27: SIMO High Side FET Peak Current V_{REGO_C} V_{REGO_D} Register

SIMO High Side FET Peak Current V_{REGO_C} V_{REGO_D}			SIMO_IPKB	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:4	ipksetd	DNM	8	**Reserved** Reserved. Do not modify this field.
3:0	ipksetc	DNM	8	**Reserved** Reserved. Do not modify this field.

Table 3-28: SIMO Maximum High Side FET Time On Register

SIMO Maximum High Side FET Time On			SIMO_MAXTON	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	-	**Reserved**
3:0	tonset	DNM	8	**Reserved, Do Not Modify**

Table 3-29: SIMO Buck Cycle Count V_{REGO_A} Register

SIMO Buck Cycle Count V_{REGO_A}			SIMO_ILOAD_A	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	iloada	DNM	0	**Reserved** Do not modify this field.

Table 3-30: SIMO Buck Cycle Count V_{REGO_B} Register

SIMO Buck Cycle Count V_{REGO_B}			SIMO_ILOAD_B	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	iloadb	DNM	0	**Reserved** Do not modify this field.

Table 3-31: SIMO Buck Cycle Count V_{REGO_C} Register

SIMO Buck Cycle Count V_{REGO_C}			SIMO_ILOAD_C	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	iloadc	DNM	0	**Reserved** Do not modify this field.

Table 3-32: SIMO Buck Cycle Count V_{REGO_D} Register

SIMO Buck Cycle Count V_{REGO_D}			SIMO_ILOAD_D	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	iloadd	DNM	0	**Reserved** Do not modify this field.

Table 3-33: SIMO Buck Cycle Count Alert V_{REGO_A} Register

SIMO Buck Cycle Count Alert V_{REGO_A}			SIMO_BUCK_ALERT_THR_A	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	buckthra	R/W	0	**Reserved** Do not modify this field.

Table 3-34: SIMO Buck Cycle Count Alert V_{REGO_B} Register

SIMO Buck Cycle Count Alert V_{REGO_B}			SIMO_BUCK_ALERT_THR_B	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	buckthrb	R/W	0	**Reserved** Do not modify this field.

Table 3-35: SIMO Buck Cycle Count Alert V_{REGO_C} Register

SIMO Buck Cycle Count Alert V_{REGO_C}			SIMO_BUCK_ALERT_THR_C	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	buckthrc	DNM	0	**Reserved** Do not modify this field.

Table 3-36: SIMO Buck Cycle Count Alert V_{REGO_D} Register

SIMO Buck Cycle Count Alert V_{REGO_D}			SIMO_BUCK_ALERT_THR_D	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	buckthrd	DNM	0	**Reserved** Do not modify this field.

Table 3-37: SIMO Buck Regulator Output Ready Register

SIMO Buck Regulator Output Ready			SIMO_BUCK_OUT_READY	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	-	**Reserved**
3	buckoutrdya	RO	0	**V_{REGO_A} Output Ready** When *SIMO_VREGO_A.vseta* changes, this bit is set when the output voltage has reached its regulated value. It is not cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
2	buckoutrdyb	RO	0	**V_{REGO_B} Output Ready** When *SIMO_VREGO_B.vsetb* changes, this bit is set when the output voltage has reached its regulated value. It is not cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
1	buckoutrdyc	RO	0	**V_{REGO_C} Output Ready** When *SIMO_VREGO_C.vsetc* changes, this bit is set when the output voltage has reached its regulated value. It is not cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
0	buckoutrdyd	RO	0	**V_{REGO_D} Output Ready** When *SIMO_VREGO_D.vsetd* changes, this bit is set when the output voltage has reached its regulated value. It is not cleared if the output voltage drops below its set value. 0: Not ready 1: Ready

Table 3-38: SIMO Zero Cross Calibration VREGO_A Register

SIMO Zero Cross Calibration VREGO_A			SIMO_ZERO_CROSS_CAL_A	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved**
4:0	zxcla	RO	0	**Reserved** Do not modify this field.

Table 3-39: SIMO Zero Cross Calibration VREGO_B Register

SIMO Zero Cross Calibration VREGO_B			SIMO_ZERO_CROSS_CAL_B	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved**
4:0	zxclb	RO	0	**Reserved** Do not modify this field.

Table 3-40: SIMO Zero Cross Calibration VREGO_C Register

SIMO Zero Cross Calibration VREGO_C			SIMO_ZERO_CROSS_CAL_C	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved**
4:0	zxclc	RO	0	**Reserved** Do not modify this field.

Table 3-41: SIMO Zero Cross Calibration VREGO_D Register

SIMO Zero Cross Calibration VREGO_D			SIMO_ZERO_CROSS_CAL_D	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved**
4:0	zxcld	RO	0	**Reserved** Do not modify this field.

3.9 Low-Power General Control Registers (LPGCR)

This set of general control registers provides reset and clock control for the low-power peripherals including:

- LPUART0 (UART3)
- LPTMR0 (TMR4)
- LPTMR1 (TMR5)
- LPWDT0 (WDT1)
- LPCOMP1, LPCOMP2 and LPCOMP3
- GPIO2

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-42 Low-Power Control Register Summary

Offset	Register	Name
[0x0008]	*LPGCR_RST*	*Reset Control Register*
[0x000C]	*LPGCR_PCLKDIS*	*Clock Control Register*

3.9.1 Low-Power General Control Registers Details

Table 3-43: Reset Control Register

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved** Do not modify

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
6	lpcomp	W1	0	**Low-Power Comparator Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
5	-	RO	0	**Reserved**
4	uart3	W1O	0	**UART3 (LPUART0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
3	tmr5	W1O	0	**TMR5 (LPTMR1) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
2	tmr4	W1O	0	**TMR4 (LPTMR0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
1	wdt1	W1O	0	**WDT1 (LPWDT0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
0	gpio2	W1O	0	**GPIO2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.

Table 3-44: Clock Disable Register

Clock Disable Register			LPGCR_PCLKDIS	[0x008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	lpcomp	R/W	0	**Low-Power Comparator Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
5	-	RO	0	**Reserved**
4	uart3	R/W	0	**UART3 (LPUART0) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
3	tmr5	R/W	0	**TMR5 (LPTMR1) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
2	tmr4	R/W	0	**TMR4 (LPTMR0) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled

Clock Disable Register			LPGCR_PCLKDIS	[0x008]
Bits	**Field**	**Access**	**Reset**	**Description**
1	wdt1	R/W	0	**WDT1 (LPWDT0) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
0	gpio2	R/W	0	**GPIO2 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled

3.10 Power Sequencer Registers (PWRSEQ)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Note: The PWSEQ registers are reset only on a POR. System reset, soft reset, and peripheral reset do not affect the PWRSEQ register values.

Table 3-45: Power Sequencer Register Summary

Offset	Register	Name
[0x0000]	*PWRSEQ_LPCN*	*Low-Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*Low-Power GPIO0 Wakeup Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*Low-Power GPIO0 Wakeup Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*Low-Power GPIO1 Wakeup Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*Low-Power GPIO1 Wakeup Enable Register*
[0x0014]	*PWRSEQ_LPWKST2*	*Low-Power GPIO2 Wakeup Status Flags*
[0x0018]	*PWRSEQ_LPWKEN2*	*Low-Power GPIO2 Wakeup Enable Register*
[0x001C]	*PWRSEQ_LPWKST3*	*Low-Power GPIO3 Wakeup Status Flags*
[0x0020]	*PWRSEQ_LPWKEN3*	*Low-Power GPIO3 Wakeup Enable Register*
[0x0030]	*PWRSEQ_LPPWST*	*Low-Power Peripheral Wakeup Status Register*
[0x0034]	*PWRSEQ_LPPWEN*	*Low-Power Peripheral Wakeup Enable Register*
[0x0044]	*PWRSEQ_VBTLEPD*	*VBTLE Power Down Register*
[0x0048]	*PWRSEQ_GP0*	*General Purpose Register 0*
[0x004C]	*PWRSEQ_GP1*	*General Purpose Register 1*

3.10.1 Power Sequencer Register Details

Table 3-46: Low-Power Control Register

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31	lpwkst_clr	R/W1	0	**Low-Power Wakeup Status Register Clear** Write 1 to this field to clear the low-power wakeup status registers: • *PWRSEQ_LPWKST0* • *PWRSEQ_LPWKST1* • *PWRSEQ_LPWKST2* • *PWRSEQ_LPWKST3* • *PWRSEQ_LPPWST* 1: Write 1 to initiate a clear of all the low-power wakeup status registers. The hardware automatically clears this field when the registers are cleared.
30:12	-	DNM	0	**Reserved, Do Not Modify.** *Note: This field must always be set to 0 to maintain future compatibility.*
11	bg_dis	R/W	1	**Band Gap Disable for *LPM* and *BACKUP* Mode** Setting this field to 1 (default) disables the bandgap during *LPM* and *BACKUP* mode. 0: System bandgap is on in *LPM* and *BACKUP* modes 1: System bandgap is off in *LPM* and *BACKUP* modes
10	-	R/W	0	**Reserved**
9	lpmfast	R/W	0	**Low-Power Mode Entry Clock Select** If the ISO is selected (default), fast *LPM* entry is enabled. Setting the clock to INRO disables fast *LPM* entry. 0: ISO used for entering *LPM* (fast mode enabled). 1: INRO used for *LPM* entry (fast mode disabled).
8	lpmclksel	R/W	1	**Low-Power Mode APB Clock Select** This field selects the clock source to use for the RV32 (CPU1) and other APB periphrals during *LPM*. 0: PCLK is used as the RV32 (CPU1) and APB system clock during *LPM*. 1: ISO is used as the RV32 (CPU1) and APB system clock during *LPM*.
7:4	-	DNM	0	**Reserved, Do Not Modify** *Note: This field must be set to 0 to maintain future compatibility.*
3	ramret3	R/W	0	**System RAM 3 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable data retention for *sysram3*. See *Table 3-12* for system RAM configuration. 0: Disabled 1: Enabled
2	ramret2	R/W	0	**System RAM 2 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable data retention for *sysram2*. See *Table 3-12* for system RAM configuration. 0: Disabled 1: Enabled
1	ramret1	R/W	0	**System RAM 1 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable data retention for *sysram1*. See *Table 3-12* for system RAM configuration. 0: Disabled 1: Enabled

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
0	ramret0	R/W	0	**System RAM 0 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable data retention for *sysram0*. See *Table 3-12* for system RAM configuration. 0: Disabled 1: Enabled

Table 3-47: GPIO0 Low-Power Wakeup Status Flags

GPIO0 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	st	R/W1C	0	**GPIO0 Pin Wakeup Status Flag** Whenever a GPIO0 pin, in any power mode, transitions from low-to-high or high-to-low, the pin's corresponding bit in this register is set. The device transitions from a low-power mode to *ACTIVE* mode if the corresponding GPIO pin's interrupt enable bit is set in the *PWRSEQ_LPWKEN0* register. *Note: Clear this register before entering any low-power mode.*

Table 3-48: GPIO0 Low-Power Wakeup Enable Registers

GPIO0 Low-Power Wakeup Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a GPIO0 pin's bit in this register causes an interrupt to be generated that wakes up the device from any low-power mode to *ACTIVE* mode. A wakeup event sets the corresponding GPIO0's bit in the *PWRSEQ_LPWKST0* register, enabling determination of which GPIO0 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32655 to wakeup from a low-power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-49: GPIO1 Low-Power Wakeup Status Flags

GPIO1 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST1	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
9:0	st	R/W1C	0	**GPIO1 Pin Wakeup Status Flag** Whenever a GPIO1 pin, in any power mode, transitions from low-to-high or high-to-low, the pin's corresponding bit in this register is set. The device transitions from a low-power mode to *ACTIVE* mode if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN1*. *Note: Clear this register before entering any low-power mode.*

Table 3-50: GPIO1 Low-Power Wakeup Enable Registers

GPIO1 Low-Power Wakeup Enable			PWRSEQ_LPWKEN1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10		RO	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
9:0	en	R/W	0	**GPIO1 Pin Wakeup Interrupt Enable** Setting a GPIO1 pin's bit in this register causes an interrupt to be generated that wakes up the device from any low-power mode to *ACTIVE* mode. A wakeup event sets the corresponding GPIO1's bit in the *PWRSEQ_LPWKST1* register, enabling determination of which GPIO1 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32655 to wakeup from a low-power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-51: GPIO2 Low-Power Wakeup Status Flags

GPIO2 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST2	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8		RO	0	**Reserved** *Note: Bits corresponding to unimplemented GPIO are ignored.*
7:0	st	R/W1C	0	**GPIO2 Pin Wakeup Status Flag** Whenever a GPIO2 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: The device transitions from a low-power mode to ACTIVE mode if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN. This register should be cleared before entering any low-power mode.*

Table 3-52: GPIO2 Low-Power Wakeup Enable Registers

GPIO2 Low-Power Wakeup Enable			PWRSEQ_LPWKEN2	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8		RO	0	**Reserved** *Note: Bits corresponding to unimplemented GPIO are ignored.*
7:0	en	R/W	0	**GPIO2 Pin Wakeup Interrupt Enable** Setting a GPIO2 pin's bit in this register causes an interrupt to be generated that wakes up the device from any low-power mode to *ACTIVE* mode. A wakeup event sets the corresponding GPIO2's bit in the *PWRSEQ_LPWKST2* register, enabling determination of which GPIO2 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32655 to wakeup from a low-power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-53: GPIO3 Low-Power Wakeup Status Flags

GPIO3 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved** *Note: Bits corresponding to unimplemented GPIO are ignored.*

GPIO3 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
1:0	st	R/W1C	0	**GPIO3 Pin Wakeup Status Flag** Whenever a GPIO3 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: The device transitions from a low-power mode to ACTIVE mode if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN. This register should be cleared before entering any low-power mode.*

Table 3-54: GPIO3 Low-Power Wakeup Enable Registers

GPIO3 Low-Power Wakeup Enable			PWRSEQ_LPWKEN3	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved**
1:0	en	R/W	0	**GPIO3 Pin Wakeup Interrupt Enable** Setting a GPIO3 pin's bit in this register causes an interrupt to be generated that wakes up the device from any low-power mode to *ACTIVE* mode. A wakeup event sets the corresponding GPIO3's bit in the *PWRSEQ_LPWKST3* register, enabling determination of which GPIO3 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32655 to wake up from a low-power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-55: Low-Power Peripheral Wakeup Status Flags

Low-Power Peripheral Wakeup Status Flags			PWRSEQ_LPPWST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18		RO	0	**Reserved**
17	reset	R/W1C	0	**Reset Detected Wakeup Flag** This field is set when an external reset causes the wakeup event.
16	backup	R/W1C	0	***BACKUP* Wakeup Flag** This field is set when the device wakes up from *BACKUP*.
15:5	-	RO	0	**Reserved**
4	comp0	R/W1C	0	**Comparator 0 Wakeup Flag** This field is set if the wakeup event was the result of an input comparator trigger event.
3:0	-	RO	0	**Reserved**

Table 3-56: Low-Power Peripheral Wakeup Enable Registers

Low-Power Peripheral Wakeup Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27		RO	0	**Reserved**
26	lpcomp	R/W	0	**LPCOMP Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the LPCOMP interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.

Low-Power Peripheral Wakeup Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
25	spi1	R/W	0	**SPI1 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the SPI0 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
24	i2s	R/W	0	**I²S Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the I²S interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
23	i2c2	R/W	0	**I2C2 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C2 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
22	i2c1	R/W	0	**I2C1 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C1 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
21	i2c0	R/W	0	**I2C0 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C0 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
20	uart3	R/W	0	**LPUART0 (UART3) Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from LPUART0 (UART3) interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
19	uart2	R/W	0	**UART2 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the UART2 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
18	uart1	R/W	0	**UART1 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the UART1 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
17	uart0	R/W	0	**UART0 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the UART0 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
16	tmr5	R/W	0	**LPTMR1 (TMR5) Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the LPTMR1 (TMR5) interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
15	tmr4	R/W	0	**LPTMR0 (TMR4) Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the LPTMR0 (TMR4) interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
14	tmr3	R/W	0	**TMR3 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR3 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.

Low-Power Peripheral Wakeup Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
13	tmr2	R/W	0	**TMR2 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR2 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
12	tmr1	R/W	0	**TMR1 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR1 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
11	tmr0	R/W	0	**TMR0 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR0 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
10	cpu1	R/W	0	**RV32 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the RV32 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
9	wdt1	R/W	0	**WDT1 (LPWDT0) Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the WDT1 (LPWDT0) interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
8	wdt0	R/W	0	**WDT0 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from the WDT0 interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
7:5	-	RO	0	**Reserved**
4	comp0	R/W	0	**Comparator 0 Interrupt Wakeup Enable** Set this field to 1 to enable wakeup events from Comparator 0's interrupt. 0: Disable wakeup on interrupt. 1: Enable wakeup on interrupt.
3:0	-	RO	0	**Reserved**

Table 3-57: Low-Power VBTLE Power Down Register

Low-Power VBTLE Power Down			PWRSEQ_VBTLEPD	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	btle	R/W	0	**Power Down SIMO VREGO_D (V_{BTLE})** Set this field to 1 to power down the V_{BTLE} power domain. 0: V_{BTLE} Powered On 1: V_{BTLE} Powered Down
0	-	RO	0	**Reserved**

Table 3-58: Low-Power General Purpose Register 0

Low-Power General Purpose Register 0			PWRSEQ_GP0	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software and retains the contents during *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP* modes.

Table 3-59: Low-Power General Purpose Register 1

Low-Power General Purpose Register 1			PWRSEQ_GP1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software and retains the contents during *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP* modes.

3.11 Trim System Initialization Registers (TRIMSIR)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Note: The TRIMSIR registers are reset only on a POR. System reset, soft reset, and peripheral reset do not affect the TRIMSIR register values.

Table 3-60: Trim System Initialization Register Summary

Offset	Register Name	Access	Description
[0x0008]	*TRIMSIR_RTC*	R/W	*RTC Trim System Initialization Register*
[0x0034]	*TRIMSIR_SIMO*	R/W	*System Initialization Register*
[0x003C]	*TRIMSIR_IPOLO*	R/W	*System initialization Function Status Register*
[0x0040]	*TRIMSIR_CTRL*	R/W	Control Trim System Initialization Register
[0x0044]	*TRIMSIR_INRO*	R/W	*INRO Trim System Initialization Register*

3.11.1 Trim System Initialization Register Details

Table 3-61: RTC Trim System Initialization Register

RTC Trim System Initialization			TRIMSIR_RTC	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31	lock	RO	0	**Lock** If this field is set to 1, this register is read-only and the *TRIMSIR_RTC.x1trim* and *TRIMSIR_RTC.x2trim* fields cannot be modified.
30:26	-	RO	0	**Reserved**
25:21	x2trim	R/W*	0	**RTC X2 Trim** The X2 trim setting for the RTC. *Note: If TRIMSIR_RTC.lock is set to 1, this field is read-only.*
20:16	x1trim	R/W*	0	**RTC X1 Trim** The X1 trim setting for the RTC. *Note: If TRIMSIR_RTC.lock is set to 1, this field is read-only.*
15:0	-	RO	0	**Reserved**

Table 3-62: SIMO Trim System Initialization Register

SIMO System Initialization			TRIMSIR_SIMO	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

SIMO System Initialization			TRIMSIR_SIMO	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	simoclk_div	R/W	1	**SIMO Clock Divide** This field selects the SIMO clock divisor. The SIMO uses the INRO as its input clock. 0: $\frac{INRO}{1}$ 1: $\frac{INRO}{16}$ 2: $\frac{INRO}{1}$ 3: $\frac{INRO}{32}$ 4: $\frac{INRO}{1}$ 5: $\frac{INRO}{64}$ 6: $\frac{INRO}{1}$ 7: $\frac{INRO}{128}$

Table 3-63: IPO Low Trim System Initialization Register

IPO Trim Low System Initialization			TRIMSIR_IPOLO	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	ipo_limitlo	RO	See Description	**IPO Low Trim Limit** This field contains the low trim limit for the IPO.

Table 3-64: Control Trim System Initialization Register

Control System Initialization			TRIMSIR_CTRL	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:29	trim8kz	R/W	See Description	**INRO Clock Trim for 8KHz** This field contains the trim for the INRO when set to 8KHz.
28:26	-	RO	0	**Reserved**
25:24	inro_sel	R/W	2	**INRO Clock Select** Selects the INRO frequency. 0: 8KHz 1: 16KHz 2: 30KHz (Power-On Reset default) 3: Reserved for Future Use
23:15	ipo_limithi	R/W	0x1FF	**IPO High Trim Limit** This field contains the high limit for the IPO.
14:8	vdda_limithi	R/W	0x78	**V_{DDA} High Trim Limit** High trim limit for V_{DDA}.
7	-	RO	0	**Reserved**
6:0	vdda_limitlo	R/W	0x64	**V_{DDA} Low Trim Limit** Low trim limit for V_{DDA}

Table 3-65: INRO Trim System Initialization Register

INRO System Initialization			TRIMSIR_INRO	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

INRO System Initialization			TRIMSIR_INRO	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
7:6	lpclksel	R/W	2	**INRO Low-Power Mode Clock Select** Selects the INRO clock frequency for *LPM* operation. 0: 8KHz 1: 16KHz 2: 30KHz (POR default) 3: Reserved for Future Use
5:3	trim30k	R/W	0	**INRO 30KHz Trim** This field contains the trim for INRO when set to 30KHz.
2:0	trim16k	R/W	0	**INRO 16KHz Trim** This field contains the trim for INRO when set to 16KHz.

3.12 Global Control Registers (GCR)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Note: The General Control Registers are only reset on a System Reset or Power-On Reset. A Soft Reset or Peripheral Reset does not affect these registers.

Table 3-66: Global Control Register Summary

Offset	Register	Description
[0x0000]	*GCR_SYSCTRL*	*System Control Register*
[0x0004]	*GCR_RST0*	*Reset Register 0*
[0x0008]	*GCR_CLKCTRL*	*Clock Control Register*
[0x000C]	*GCR_PM*	*Power Management Register*
[0x0018]	*GCR_PCLKDIV*	*Peripheral Clocks Divisor*
[0x0024]	*GCR_PCLKDIS0*	*Peripheral Clocks Disable 0*
[0x0028]	*GCR_MEMCTRL*	*Memory Clock Control*
[0x002C]	*GCR_MEMZ*	*Memory Zeroize Register*
[0x0040]	*GCR_SYSST*	*System Status Flags*
[0x0044]	*GCR_RST1*	*Reset Register 1*
[0x0048]	*GCR_PCLKDIS1*	*Peripheral Clocks Disable 1*
[0x004C]	*GCR_EVENTEN*	*Event Enable Register*
[0x0050]	*GCR_REVISION*	*Revision Register*
[0x0054]	*GCR_SYSIE*	*System Status Interrupt Enable*
[0x0064]	*GCR_ECCERR*	*Error Correction Coding Error Register*
[0x0068]	*GCR_ECCCED*	*Error Correction Coding Correctable Error Detected*
[0x006C]	*GCR_ECCIE*	*Error Correction Coding Interrupt Enable Register*
[0x0070]	*GCR_ECCADDR*	*Error Correction Coding Error Address Register*
[0x0074]	*GCR_BTLELDOCTRL*	*BTLE LDO Control Register*
[0x0078]	*GCR_BTLELDODLY*	*BTLE LDO Delay Count Register*
[0x0080]	*GCR_GPR*	*General Purpose Register 0*

3.12.1 Global Control Register Details (GCR)

Table 3-67: System Control Register

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**
17:16	ovr	R/W	0b10	**Operating Voltage Range** Set this field to match the V_{CORE} voltage to enable the on-chip RAM to operate at the optimal timing range. 0b00: 0.9V ± 10% 0b01: 1.0V ± 10% 0b10: 1.1V ± 10% 0b11: Reserved for Future Use
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This is the result after setting bit *GCR_SYSCTRL.cchk*. This bit is only valid after the ROM checksum is complete and *GCR_SYSCTRL.cchk* is cleared. 0: Pass 1: Fail
14	sw_dis	R/W	0	**Serial Wire Debug Disable** This bit is used to disable the serial wire debug interface. 0: SWD disabled. 1: SWD enabled. *Note: This bit is only writeable if the flash is not factory locked or if the GCR_SYSST.icelock bit is 0 and the GCR_SYSCTRL.rom_done bit is 1.*
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation. 1: Start ROM checksum calculation.
12	romdone	R/W	0	**ROM Start Code Status** Used to disable SWD interface during system initialization procedure. 0: ROM start code not completed. 1: ROM start code not completed. *Note: If the flash is factory locked, software can only set this bit. Subsequently, this bit cannot be cleared by software.*
11:7	-	RO	0	**Reserved**
6	icc0_flush	R/W	0	**ICC0 Cache Flush** Write 1 to flush the code cache and the instruction buffer for the M4. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect and does not stop a cache flush in progress. 0: ICC0 flush complete. 1: Flush the contents of the ICC0 cache.
5	-	RO	0	**Reserved**

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
4	flash_page_flip	R/*	0	**Flash Page Flip Flag** Flips the bottom and top halves of Flash memory. This bit is controlled by hardware. *Note: Software should not change the state of this bit during normal operation. Any change to this bit flushes all caches.* 0: Physical layout matches logical layout. 1: Top and Bottom halves flipped.
3:1	-	RO	0b001	**Reserved**
0	bstapen	R/W	*	**Boundary Scan Tap Enable** This field's reset value matches *GCR_SYSST.icelock*. Do not modify this field.

Table 3-68: Reset Register 0

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	sys	R/W	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *System Reset* for additional information.
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: Watchdog timers, GPIO ports, the AoD, RAM retention and the GCR are unaffected. See* Peripheral Reset *for additional information.*
29	soft	R/W	0	**Soft Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Soft Reset* for additional information.
28	uart2	R/W	0	**UART2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27	-	R/W	0	**Reserved**
26	adc	R/W	0	**ADC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
25	-	RO	0	**Reserved** Do not modify
24	trng	R/W	0	**TRNG Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
23:19	-	RO	0	**Reserved**
18	btle	R/W	0	**Bluetooth Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
17	rtc	R/W	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
16	i2c0	R/W	0	**I2C0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15:14	-	RO	0	**Reserved**
13	spi1	R/W	0	**SPI1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Reset 0				GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
12	uart1	R/W	0	**UART1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
11	uart0	R/W	0	**UART0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
10:9	-	RO	0	**Reserved**	
8	tmr3	R/W	0	**TMR3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
7	tmr2	R/W	0	**TMR2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
6	tmr1	R/W	0	**TMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
5	tmr0	R/W	0	**TMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
4	-	RO	-	**Reserved** Do not modify	
3	gpio1	R/W	0	**GPIO1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
2	gpio0	R/W	0	**GPIO0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
1	wdt0	R/W	0	**Watchdog Timer 0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
0	dma	R/W	0	**DMA Access Block Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	

Table 3-69: Clock Control Register

Clock Control				GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:30	-	RO	0b10	**Reserved**	
29	inro_rdy	RO	0	**8kHz Internal Nano-Ring Oscillator (INRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
28	ibro_rdy	RO	0	**7.3728MHz Internal Baud Rate Oscillator (IBRO) Ready Status** 0: Not ready. 1: Oscillator ready.	
27	ipo_rdy	RO	0	**100MHz Internal Primary Oscillator (IPO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
26	iso_rdy	RO	0	**60MHz Internal Secondary Oscillator (ISO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
25	ertco_rdy	RO	0	**32.768kHz External RTC Oscillator (ERTCO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	

Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
24	erfo_rdy	RO	0	**32MHz External RF Oscillator (ERFO) Ready** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved**
21	ibro_vs	R/W	0	**IBRO Voltage Select** Set this field to 0 to use V_{COREA} as the IBRO. Setting this field to 1 uses a dedicated 1V regulated supply for the IBRO. 0: V_{CORE} selected as the IBRO supply 1: IBRO supplied using a 1V regulated supply.
20	ibro_en	RO	1	**7.3728MHz IBRO Enable** The IBRO is always enabled. 1: Enabled and ready when *GCR_CLKCTRL.ibro_rdy* = 1.
19	ipo_en	R/W	0	**100MHz IPO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.ipo_rdy* = 1.
18	iso_en	R/W	1	**60MHz ISO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.iso_rdy* = 1
17	ertco_en	R/W	0	**32.768kHz ERTCO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.ertco_rdy* = 1.
16	erfo_en	R/W	0	**32MHz ERFO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.erfo_rdy* = 1.
15:14	-	RO	0	**Reserved**
13	sysclk_rdy	RO	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying *GCR_CLKCTRL.sysclk_sel*, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is clock source selected in *GCR_CLKCTRL.sysclk_sel*.
12	-	RO	0	**Reserved**
11:9	sysclk_sel	R/W	0	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field immediately clears *GCR_CLKCTRL.sysclk_rdy*. 0: ISO 1: Reserved 2: ERFO 3: INRO 4: IPO 5: IBRO 6: ERTCO 7: External Clock, EXT_CLK, P0.3, AF1
8:6	sysclk_div	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC as shown in the following equation: $SYS\_CLK = \frac{SYS\_CLK}{2^{sysclk\_div}}$ *Note: Valid values are from 0 to 7 for sysclk_div.*

Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
5:0	-	RO	4	**Reserved**

Table 3-70: Power Management Register

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20	erfo_bp	R/W	0	**ERFO Bypass** This bit is set to 0 on a POR is not affected by other forms of reset. Set this field to 1 to use a square wave input as the clock source for the ERFO driving the HFXIN pin. 0: Clock source is the crystal oscillator, driving the crystal connected between HFXIN and HFXOUT pins. 1: Clock source is a square wave driven into the HFXIN pin.
19:18	-	RO	0	**Reserved**
17	ibro_pd	R/W	1	**IBRO Power Down** Set this field to 1 to power down the IBRO during *LPM* and *UPM*. 0: IBRO enabled during *LPM* and *UPM.* 1: IBRO powered down during *LPM* and *UPM.*
16	ipo_pd	R/W	1	**IPO Power Down** Set this field to 1 to power down the IPO during *LPM*. 0: IPO enabled during *LPM.* 1: IPO powered down during *LPM.*
15	iso_pd	R/W	1	**ISO Power Down** This bit selects the power state in *LPM* for the ISO oscillator. 0: IPO enabled during *LPM.* 1: IPO powered down in *LPM.*
14:10	-	DNM	0b11100	**Reserved, Do Not Modify**
9	aincomp_we	R/W	0	**Analog Input Comparator Wakeup Enable** This bit enables the Analog Input Comparator IRQ to wake the device from *SLEEP*, *LPM, UPM*, *STANDBY,* or *BACKUP*.
8	-	RO	0	**Reserved**
7	wut_we	R/W	0	**Wakeup Time Enable** Set this field to 1 to enable the wakeup timer as a wakeup source. The wakeup timer wakes the device from *SLEEP*, *LPM*, or *BACKUP*. 0: Disabled 1: Enabled
6	-	RO	0	**Reserved**
5	rtc_we	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm wakes the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*. 0: Disabled 1: Enabled
4	gpio_we	R/W	0	**GPIO Wakeup Enable** Set this field to 1 to enable all GPIO pins as potential wakeup sources. Any GPIO configured for wakeup wakes the device from *SLEEP* , *LPM*, or *BACKUP*. 0: Disabled 1: Enabled

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	mode	R/W	0	**Operating Mode** This field controls the operating state of the device. 0b0000: *ACTIVE* 0b0001: *SLEEP* 0b0010: *STANDBY* 0b0100: *BACKUP* 0b1000: *LPM* (CM4 *deepsleep*) 0b1001: *UPM* 0b1010: *PDM*

Table 3-71: Peripheral Clock Divisor Register

Peripheral Clocks Divisor			GCR_PCLKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	-	**Reserved**
13:10	adcfrq	R/W	0	**ADC Peripheral Clock Frequency Select** Configures the frequency of the ADC peripheral clock from the PCLK. 0: Reserved 1: Reserved 2 – 15: $f_{adc\_CLK} = {f_{PCLK}}/{adcfrq}$
9:0	-	RO	-	**Reserved**

Table 3-72: Peripheral Clock Disable Register 0

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	1	**Reserved**
29	pt	R/W	1	**Pulse Train Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
28	i2c1	R/W	1	**I2C1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
27:24	-	RO	1	**Reserved**
23	adc	R/W	1	**ADC Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
22:19	-	RO	1	**Reserved**
18	tmr3	R/W	1	**TMR3 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
17	tmr2	R/W	1	**TMR2 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
16	tmr1	R/W	1	**TMR1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
15	tmr0	R/W	1	**TMR0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
14	-	RO	1	**Reserved**
13	i2c0	R/W	1	**I2C0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
12:11	-	RO	1	**Reserved**
10	uart1	R/W	1	**UART1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
9	uart0	R/W	1	**UART0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
8	-	RO	1	**Reserved**
6	spi1	R/W	1	**SPI1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
5	dma	R/W	1	**DMA Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
4:2	-	RO	1	**Reserved**
1	gpio1	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
0	gpio0	R/W	1	**GPIO0 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled

Table 3-73: Memory Clock Control Register

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	sysram0ecc	R/W	0	**Sysram0 ECC Enable** Set this field to 1 to enable ECC for *sysram0*. 0: *sysram0* active, ECC disabled. 1: *sysram0* active, ECC enabled.
15:3	-	RO	0	**Reserved** Do not modify
2:0	fws	R/W	5	**Program Flash Wait States** Number of wait-state cycles per Flash code read access. 0: Invalid 1 – 7: Number of Flash code access wait states *Note: For the IPO and ISO clocks the minimum wait state is 2.* *Note: For all other clock sources the minimum wait state is 1.*

Table 3-74: Memory Zeroize Control Register

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	-	**Reserved**
6	icc1	R/W1O	0	**ICC1 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress
5	icc0	R/W1O	0	**ICC0 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress
4	sysram0ecc	R/W10	0	**sysram0 ECC Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress
3	ram3	R/W1O	0	***sysram3* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
2	ram2	R/W1O	0	***sysram2* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress
1	ram1	R/W1C	0	***sysram1* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress
0	ram0	R/W1C	0	***sysram0* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete 1: Operation in progress

Table 3-75: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	icelock	R	0	**Arm ICE Lock Status Flag** 0: Arm ICE is unlocked (enabled). 1: Arm ICE is locked (disabled).

Table 3-76: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	RO	0	**CPU1 (RV32) Reset** Write 1 to initiate the reset operation. 0: Operation complete. 1: Operation in progress.
30:26	-	RO	0	**Reserved**
25	simo	R/W	0	**Single Inductor Multiple Output Block Reset** Write 1 to initiate the reset operation. 0: Operation complete. 1: Operation in progress.
24	dvs	R/W	0	**Dynamic Voltage Scaling Controller Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
23:21	-	RO	0	**Reserved**
20	i2c2	R/W	0	**I^2C2 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
19	i2s	R/W	0	**Audio Interface Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
18:17	-	RO	0	**Reserved**
16	smphr	R/W	0	**Semaphore Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
15:12	-	RO	-	**Reserved**
11	spi0	R/W	0	**SPI0 AHB Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
10	aes	R/W	0	**AES Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
9	crc	R/W	0	**CRC Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
8	-	RO	0	**Reserved**
7	owm	R/W	0	**One-Wire Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
6:2	-	RO	0	**Reserved**
1	pt	R/W	0	**Pulse Train Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	i2c1	R/W	0	**I^2C1 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 3-77: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	R/W	1	**CPU1 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
30:28	-	RO	1	**Reserved**

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
27	wdt0	R/W	1	**Watchdog Timer 0 Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
26:25	-	RO	1	**Reserved**
24	i2c2	R/W	1	**I²C2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
23	i2s0	R/W	1	**I²S Audio Interface Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
22:17	-	RO	1	**Reserved**
16	spi0	R/W	1	**SPI0 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
15	aes	R/W	1	**AES Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
14	crc	R/W	1	**CRC Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
13	owm	R/W	1	**One-Wire Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
12:10	-	RO	1	**Reserved**
9	smphr	R/W	1	**Semaphore Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
8:3	-	RO	1	**Reserved**

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
2	trng	R/W	1	**TRNG Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
1	uart2	R/W	1	**UART2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled
0	btle	R/W	1	**Bluetooth Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled

Table 3-78: Event Enable Register

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	tx	R/W	0	**CPU0 (CM4) TXEV Event Enable** When this bit is set, the TXEV event wakes the CM4 from a low-power mode entered with a WFE instruction. 0: Disabled 1: Enabled
1	-	RO	0	**Reserved**
0	dma	R/W	0	**CPU0 (CM4) DMA CTZ Wakeup Enable** Enables a DMA CTZ event to generate an RXEV interrupt to wake the CM4 from a low-power mode entered with a WFE instruction. 0: Disabled 1: Enabled

Table 3-79: Revision Register

Revision			GCR_REVISION	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	*	**Device Revision** Returns the chip revision ID as packed BCD. For example, 0xA1 would indicate the device is revision A1.

Table 3-80: System Status Interrupt Enable Register

System Status Interrupt Enable			GCR_SYSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	-	**Reserved**

System Status Interrupt Enable			GCR_SYSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
0	iceunlock	R/W	0	**Arm ICE Unlocked Interrupt Enable** Set this field to generate an interrupt if the *GCR_SYSST.icelock* field is set. 0: Disabled 1: Enabled

Table 3-81: Error Correction Coding Error Register

ECC Error			GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	ram0	R/W1C	0	***sysram0* ECC Error** This flag is set if an ECC error occurs in *sysram0*. Write to 1 to clear the flag. 0: No error 1: Error

Table 3-82: Error Correction Coding Correctable Error Detected Register

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	ram0	R/W1C	0	***sysram0* Correctable ECC Error Detected** When this bit is set it indicates that there is a single correctable error in the *sysram0* block. Write to 1 to clear the flag. 0: No error or uncorrectable error if *GCR_ECCERR.ram0* is set to 1. 1: Correctable error detected.

Table 3-83: Error Correction Coding Interrupt Enable Register

ECC Interrupt Enable			GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	ram0	R/W	0	***sysram0* ECC Error Interrupt Enable** Set this field to 1 to generate an interrupt if an ECC error condition occurs for *sysram0*. 0: Disabled 1: Enabled

Table 3-84: Error Correction Coding Error Address Register

ECC Error Address			GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31	tagramerr	R	0	**ECC Error Address/Tag RAM Error** Data depends on which block has reported the error. If *sysram0*,then this bit represents the bit of the AMBA address of read which produced the error. If the error is the cache, then this bit is set as shown below: 0: No error. 1: Tag error. The error is in the tag RAM.

ECC Error Address			GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
30	tagrambank	R	0	**ECC Error Address/Tag RAM Error Bank** Data depends on which block has reported the error. If *sysram0*, then this bit represents the bit of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in tag RAM bank 0. 1: Error is in tag RAM bank 1.
29:16	tagramaddr	R	0	**ECC Error Address/Tag RAM Error Address** Data depends on which block has reported the error. If *sysram0*, these bits represents the bits of the AMBA address of read which produced the error. If the error is from the cache, then this field is set as shown below: [TAG ADDRESS]: Represents the tag RAM address.
15	dataramerr	R	0	**ECC Error Address/Data RAM Error Address** Data depends on which block has reported the error. If *sysram0*, then this bit represents the bit of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: No error. 1: *sysram0* error. The error is in the *sysram0.*
14	datarambank	R	0	**ECC Error Address/Data RAM Error Bank** Data depends on which block has reported the error. If *sysram0*, then this bit represents the bits of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in data RAM bank 0. 1: Error is in data RAM bank 1.
13:0	dataramaddr	R	0	**ECC Error Address/Tag RAM Error Address** Data depends on which block has reported the error. This field represents the bits of the AMBA address of read which produced the error. If the error is from the caches, then this field is set as shown below: [DATA ADDRESS]: Represents the data RAM error address

Table 3-85: Bluetooth LDO Control Register

Bluetooth LDO Control			GCR_BTLELDOCTRL	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ldotxbypenendly	R	0	**LDOTX Bypass Enable Delay** Not used
14	ldorxbypenendly	R	0	**LDORX Bypass Enable Delay** Not used.
13	ldorxendly	R	0	**LDORX Enable Delay** Not used.
12	ldotxendly	R	0	**LDOTX Enable Delay** Not used.
11	ldotxdisch	R/W	0	**LDOTX Discharge** This bit is used to discharge the LDOTX output using a strong pulldown. For use when switching between bypass mode and regulation mode. 0: No discharge 1: Discharge

Bluetooth LDO Control			GCR_BTLELDOCTRL	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
10	ldotxbyp	R/W	0	**LDOTX Bypass Enable** This bit enables the LDOTX bypass mode and charge the LDOTX output voltage to its input voltage level.
9	ldorxdisch	R/W	0	**LDORX Discharge** This bit is used to discharge the LDORX output using a strong pulldown. For use when switching between bypass mode and regulation mode. 0: No discharge 1: Discharge
8	ldorxbyp	R/W	0	**LDORX Bypass Enable** This bit enables the LDORX bypass mode and charges the LDORX output voltage to its input voltage level.
7:6	ldorxvsel	R/W	b'01	**LDORX Output Voltage Setting** 0b00: 0.85V 0b01: 0.90V 0b10: 1.0V 0b11: 1.1V
5	ldorxpulld	R/W	1	**LDORX Pulldown** This bit is used to provide a weak pulldown to the LDORX output. This bit is only valid if both *GCR_BTLELDOCTRL.ldorxen* and *GCR_BTLELDOCTRL.ldorxbyp* are set to 0. 0: Pulldown disabled. 1: Pulldown enabled.
4	ldorxen	R/W	0	**LDORX Enable** Enable the RX LDO and charge its output voltage to the setting in *GCR_BTLELDOCTRL.ldorxvsel*. 0: Disabled 1: Enabled
3:2	ldotxvsel	R/W	b'01	**LDOTX Output Voltage Setting** 0b00: 0.85V 0b01: 0.9V 0b10: 1.0V 0b11: 1.1V
1	ldotxpulld	R/W	1	**LDOTX Pull Down** Setting this bit enables a weak pulldown on the output of the TX LDO. 0: Pulldown disabled. 1: Pulldown enabled.
0	ldotxen	R/W	0	**LDOTX Enable** Enable the TX LDO and charge its output voltage to the setting in *GCR_BTLELDOCTRL.ldotxvsel*. 0: Disabled 1: Enabled

Table 3-86: Bluetooth LDO Delay Count Register

Bluetooth LDO Delay Count			GCR_BTLELDODLY	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:20	ldotxdlycnt	R/W	0x01B	**Bluetooth LDOTX Delay Count** Not used

Bluetooth LDO Delay Count			GCR_BTLELDODLY	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
19:17	-	RO	0	**Reserved**
16:8	ldorxdlycnt	R/W	0x01B	**Bluetooth LDORX Delay Count** Not used
7:0	bypdlycnt	R/W	0x28	**Bluetooth LDO Bypass Delay Count** Not used

Table 3-87: General Purpose Register

General Purpose			GCR_GPR	[0x0080]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	gpr	R/W	0	**General Purpose Register** General purpose register.

3.13 Error Correction Coding (ECC)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-88: Error Correction Coding Enable Register Summary

Offset	Register Name	Access	Description
[0x0000]	*ECC_EN*	R/W	*Error Correction Coding Enable*

3.13.1 Error Correction Coding Enable Register Details

Table 3-89: Error Correction Coding Enable Register

Error Correction Coding Enable			ECC_EN	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	ram0	R/W	0	***sysram0* ECC Enable** Set this field to 1 to enable ECC for *sysram0*. 0: Disabled 1: Enabled
7:0	-	RO	0	**Reserved**

3.14 System Initialization Registers (SIR)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-90: System Initialization Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SIR_STAT*	RO	*System Initialization Status Register*
[0x0004]	*SIR_ADDR*	RO	*System Initialization Address Error Register*
[0x0100]	*SIR_FSTAT*	RO	*System initialization Function Status Register*
[0x0104]	*SIR_SFSTAT*	RO	*System initialization Security Function Status Register*

3.14.1 System Initialization Register Details

Table 3-91: System Initialization Status Register

System Initialization Status			SIR_STAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	crcerr	RO	See Description	**Configuration Error Flag** This field is set by hardware during reset if an error in the device configuration is detected. 0: Configuration valid. 1: Configuration invalid. *Note: If this field reads 1, a device error has occurred. Please contact Maxim Integrated technical support for additional assistance providing the address contained in SIR_ADDR.addr.*
0	magic	RO	See Description	**Configuration Valid Flag** This field is set to 1 by hardware during reset if the device configuration is valid. 0: Configuration invalid. 1: Configuration valid. *Note: If this field reads 0, the device configuration is invalid, and a device error has occurred during system initialization. Please contact Maxim Integrated technical support for additional assistance.*

Table 3-92: System Initialization Address Error Register

System Initialization Status			SIR_ADDR	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	erraddr	RO	0	**Configuration Error Address** If the *SIR_STAT.crcerr* field is set to 1, the value in this register is the address of the configuration failure.

Table 3-93: System Initialization Function Status Register

System Initialization Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	btle	RO	See Description	**BTLE** This field indicates if the device includes the BTLE. 0: Block is not available. 1: Block is available.
8	-	RO	0	**Reserved**
7	smphr	RO	See Description	**Semaphore** This field indicates if the device includes the Semaphore. 0: Block is not available. 1: Block is available.
6:3	-	RO	0	**Reserved**
2	adc	RO	See Description	**ADC** This field indicates if the device includes the ADC. 0: Block is not available. 1: Block is available.
1	-	RO	0	**Reserved**

System Initialization Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
0	fpu	RO	See Description	**FPU** This field indicates if the device includes the FPU. 0: Block is not available. 1: Block is available.

Table 3-94: System Initialization Security Function Status Register

System Initialization Security Function Status			SIR_SFSTAT	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	aes	RO	See Description	**AES** This field indicates if the device includes the AES block. 0: Block is not available. 1: Block is available.
2	trng	RO	See Description	**TRNG** This field indicates if the device includes the TRNG block. 0: Block is not available. 1: Block is available.
1:0	-	RO	0	**Reserved**

3.15 Function Control Registers (FCR)

See Table 2-4 for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 3-95: Function Control Register Summary

Offset	Register	Access	Description
[0x0000]	*FCR_FCTRL0*	*R/W*	*Function Control 0 Register*
[0x0004]	*FCR_AUTOCAL0*	*R/W*	*Automatic Calibration 0 Register*
[0x0008]	*FCR_AUTOCAL1*	*R/W*	*Automatic Calibration 1 Register*
[0x000C]	*FCR_AUTOCAL2*	*R/W*	*Automatic Calibration 2 Register*
[0x0010]	*FCR_URVBOOTADDR*	*R/W*	*RV32 Boot Address Register*
[0x0014]	*FCR_URVCTRL*	*R/W*	*RV32 Control Register*
[0x0018]	*FCR_ERFOKS*	*R/W*	*ERFO Kick Start Register*

3.15.1 Function Control Register Details

Table 3-96: Function Control 0 Register

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	i2c2_scl_filter_en	R/W	0	**I^2C2 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
24	i2c2_sda_filter_en	R/W	0	**I^2C2 SDA Glitch Filter Enable** 0: Disabled 1: Enabled

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
23	i2c1_scl_filter_en	R/W	0	**I²C1 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
22	i2c1_sda_filter_en	R/W	0	**I²C1 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
21	i2c0_scl_filter_en	R/W	0	**I²C0 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
20	i2c0_sda_filter_en	R/W	0	**I²C0 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
19:0	-	RO	0	**Reserved**

Table 3-97: Automatic Calibration 0 Register

Automatic Calibration 0			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	trim	RO	0	**IPO Automatic Trim Value Output** This field contains the calculated trim output from an automatic calibration run.
22:20	-	RO		**Reserved**
19:8	gain	R/W	0	**IPO Trim Pulse Count** Load this field with the desired number of trim adjustment pulses required before the trim is updated during automatic calibration operation. The recommended value for this field is 4.
7:5	-	RO	0	**Reserved**
4	atomic	R/W1	0	**IPO Trim Atomic Start** Set this bit to start an atomic automatic calibration of the IPO. The calibration will run for *FCR_AUTOCAL2.runtime* milliseconds. This bit is automatically cleared by hardware when the calibration is complete.
3	invert	R/W	0	**IPO Trim Step Invert** Set this field to invert the up/down trim steps during calibration operations. 0: Trim steps are not inverted 1: Enable inverted trim steps for calibration.
2	load	R/*	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.initial*. This bit is cleared by hardware once the load is complete.
1	en	R/W	0	**IPO Automatic Calibration Run** 0: Not running 1: Running
0	sel	R/W	0	**IPO Automatic Calibration Enable** Selects the trim value to use for the IPO. The reset default for this field uses the factory trim for the IPO. Setting this field to 1 uses the automatic calibration trim output stored in *FCR_AUTOCAL0.trim*. 0: Use default trim from factory 1: Use automatic calibration trim value calculated and stored in *FCR_AUTOCAL0.trim*.

Table 3-98: Automatic Calibration 1 Register

Automatic Calibration 1			FCR_AUTOCAL1	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	initial	R/W	0	**IPO Trim Automatic Calibration Initial Trim** This field contains the initial trim setting for the IPO. Set this field to the desired initial trim value to use for an IPO automatic calibration operation. The closer this field is to the target trim value required, the faster the automatic trim operation completes.

Table 3-99: Automatic Calibration 2 Register

Automatic Calibration 2			FCR_AUTOCAL2	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20:8	div	R/W	0	**IPO Trim Automatic Calibration Divide Factor** The target frequency of the calibration is determined by dividing the IPO frequency by 32,768 before comparing it with the ERTCO frequency. For example, to achieve a target IPO frequency of 100MHz, load this field with 0xBEB (3,051). *Equation 3-2: IPO Divisor Equation* $div = \frac{f_{IPO\_TARGET}}{32768}$ *Note: Setting div to 0 is equivalent to setting div to 1.*
7:0	runtime	R/W	0	**IPO Trim Automatic Calibration Run Time** Set this field to the desired number of milliseconds for the atomic calibration run time for the IPO. $Automatic\ Calibration\ Run\ Time = runtime\ (milliseconds)$

Table 3-100: RV32 Boot Address Register

RV32 Boot Address			FCR_URVBOOTADDR	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0x2001 C000	**RV32 Boot Address** Set this field to the boot address for the RV32 core. The reset value for this register is 0x2001 C000.

Table 3-101: RV32 Control Register

RV32 Boot Address			FCR_URVCTRL	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	iflushen	R/W	0	**ICC1 Cache Flush** Write 1 to flush the cache and the instruction buffer for the RV32 core. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect and does not stop a cache flush in progress. 0: ICC1 flush complete. 1: Flush the contents of the ICC1 cache.

RV32 Boot Address			FCR_URVCTRL	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
0	memsel	R/W	0	**RV32 Memory Select** This field determines if *sysram2* and *sysram3* are shared between the CM4 and RV32 cores. Set this field to 1 to set the RV32 core as the exclusive master for *sysram2* and *sysram3*. 0: *sysram2* and *sysram3* are shared and accessible by both the CM4 and RV32 cores. 1: *sysram2* and *sysram3* are accessible by the RV32 core only. *Note: The application firmware must ensure that no accesses are occurring in either sysram2 or sysram3 prior to setting this field to 1. See* Semaphores *for information on multiprocessor communications.*

Table 3-102: ERFO Kick Start Register

ERFO Kick Start			FCR_ERFOKS	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	R/W	0	**Reserved**
13:12	ksclksel	R/W	0	**Kick Start Clock Select for ERFO** Set this field to the clock source to use for kick starting the ERFO. 0: No kick start clock 1: Reserved 2: ISO 3: IPO
11	kserfo2x	R/W	0	**Kick Start ERFO Double Pulse Length** Setting this field to 1 enables double pulse length for the kick start pulses. 0: Disabled 1: Enabled
10:8	kserfodriver	R/W	0	**Kick Start ERFO Drive Strength** This field controls the drive strength of the kick start pulses. 0: Disabled 1 – 7: Drive strength selection.
7	kserfo_en	R/W	0	**Kick Start ERFO Enable** Set this field to 1 to start the kick start of the ERFO. 0: Disabled 1: Enabled
6:0	kserfo_cnt	R/W	0	**Kick Start ERFO Counter** Set this value to the number of kick start counts required to improve the start time for the ERFO. See *ERFO Kick Start* for details.

4. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU Nested Vector Interrupt Controller (NVIC) or the RV32 interrupt controller. The NVIC manages the interrupts, exceptions, priorities, and masking. *Table 4-1* and *Table 4-2* details the MAX32655's interrupt vector tables for the CM4 and RV32 processors respectively, and describes each exception and interrupt.

4.1 CM4 Interrupt and Exception Features

- 8 programmable priority levels
- Nested exception and interrupt support
- Interrupt masking

4.2 CM4 Interrupt Vector Table

Table 4-1 lists the interrupt and exception table for the MAX32655's CM4 core. There are 119 interrupt entries for the MAX32655, including reserved for future use interrupt place holders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 134.

Table 4-1: MAX32655 CM4 Interrupt Vector Table

Exception (Interrupt) Number	Offset	Name	Description
1	[0x0004]	Reset_Handler	Reset
2	[0x0008]	NMI_Handler	Non-Maskable Interrupt
3	[0x000C]	HardFault_Handler	Hard Fault
4	[0x0010]	MemManage_Handler	Memory Management Fault
5	[0x0014]	BusFault_Handler	Bus Fault
6	[0x0018]	UsageFault_Handler	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVC_Handler	Supervisor Call Exception
12	[0x0030]	DebugMon_Handler	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_Handler	Request Pending for System Service
15	[0x003C]	SysTick_Handler	System Tick Timer
16	[0x0040]	PF_IRQHandler	Power Fail interrupt
17	[0x0044]	WDT0_IRQHandler	Windowed Watchdog Timer 0 Interrupt
18	[0x0048]	-	Reserved
19	[0x004C]	RTC_IRQHandler	Reserved
20	[0x0050]	TRNG_IRQHandler	True Random Number Generator Interrupt
21	[0x0054]	TMR0_IRQHandler	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQHandler	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQHandler	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQHandler	Timer 3 Interrupt
25	[0x0064]	LPTMR0_IRQHandler	Low-Power Timer 0 (TMR4) Interrupt
26	[0x0068]	LPTMR1_IRQHandler	Low-Power Timer 1 (TMR5) Interrupt
27:28	[0x006C]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQHandler	I^2C Port 0 Interrupt
30	[0x0078]	UART0_IRQHandler	UART Port 0 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
31	[0x007C]	UART1_IRQHandler	UART Port 1 Interrupt
32	[0x0080]	SPI1_IRQHandler	SPI Port 1 Interrupt
33:35	[0x0084]:[0x008C]	-	Reserved
36	[0x90]	ADC_IRQHandler	ADC Interrupt
37:38	[0x0094]:[0x0098]	-	Reserved
39	[0x009C]	FLC0_IRQHandler	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQHandler	GPIO Port 0 Interrupt
41	[0x00A4]	GPIO1_IRQHandler	GPIO Port 1 Interrupt
42	[0x00A8]	GPIO2_IRQHandler	GPIO Port 2 Interrupt
43	[0x00AC]	-	Reserved
44	[0x00B0]	DMA0_IRQHandler	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQHandler	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQHandler	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQHandler	DMA3 Interrupt
48:49	[0x00C0 : 0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQHandler	UART Port 2 Interrupt
51	[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQHandler	I^2C Port 1 Interrupt
53:54	[0x00D4]:[0x00D8]	-	Reserved
55	[0x00DC]	BTLE_TX_DONE_IRQHandler	Bluetooth Transmitter Done Interrupt
56	[0x00E0]	BTLE_RX_RCVD_IRQHandler	Bluetooth Receive Data Interrupt
57	[0x00E4]	BTLE_RX_ENG_DET_IRQHandler	Bluetooth Receive Energy Detected Interrupt
58	[0x00E8]	BTLE_SFD_DET_IRQHandler	BTLE SFD Detected
59	[0x00EC]	BTLE_SFD_TO_IRQHandler	BTLE SFD Timeout
60	[0x00F0]	BTLE_GP_EVENT_IRQHandler	BTLE Timestamp
61	[0x00F4]	BTLE_CFO_IRQHandler	BTLE CFO Done
62	[0x00F8]	BTLE_SIG_DET_IRQHandler	BTLE Signal Detected
63	[0x00FC]	BTLE_AGC_EVENT_IRQHandler	BTLE AGC Event
64	[0x0100]	BTLE_RFFE_SPIM_IRQHandler	BTLE RFFE SPIM Done
65	[0x0104]	BTLE_TX_AES_IRQHandler	BTLE TX AES Done
66	[0x0108]	BTLE_RX_AES_IRQHandler	BTLE RX AES Done
67	[0x010C]	BTLE_INV_APB_ADDR_IRQHandler	BTLE Invalid APB Address
68	[0x0110]	BTLE_IQ_DATA_VALID_IRQHandler	BTLE IQ Data Valid
69	[0x0114]	WUT_IRQHandler	Wakeup Timer Interrupt
70	[0x0118]	GPIOWAKE_IRQHandler	GPIO Wakeup Interrupt
71	[0x011C]	-	Reserved
72	[0x0120]	SPI0_IRQHandler	SPI Port 0 Interrupt
73	[0x0124]	LPWDT0_IRQHandler	Low-Power Watchdog Timer 0 (WDT1) Interrupt
74	[0x0128]	-	Reserved
75	[0x012C]	PT_IRQHandler	Pulse Train Interrupt
76:77	[0x0130]:[0x0134]	-	Reserved
78	[0x0138]	I2C2_IRQHandler	I^2C Port 2 Interrupt
79	[0x013C]	RISCV_IRQHandler	RV32 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
80:82	[0x0140]:[0x0148]	-	Reserved
83	[0x014C]	OWM_IRQHandler	1-Wire Master Interrupt
84:97	[0x0150]:[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQHandler	Error Correction Coding Block Interrupt
99	[0x018C]	DVS_IRQHandler	Digital Voltage Scaling Interrupt
100	[0x0190]	SIMO_IRQHandler	Single Input Multiple Output Interrupt
101:103	[0x0194]:[0x019C]	-	Reserved
104	[0x01A0}	LPUART0_IRQHandler	Low-Power UART 0 (UART3) Interrupt
105:106	[0x01A4]:[0x01A8]	-	Reserved
107:112	[0x01B0]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQHandler	AES Interrupt
114	[0x01C8]	-	Reserved
115	[0x01CC]	I2S_IRQHandler	I2S Interrupt
116:118	[0x01D0]:[0x01D8]	-	Reserved
119	[0x01DC]	LPCMP_IRQHandler	Low-Power Comparator Interrupt

4.3 RV32 Interrupt Vector Table

Table 4-2 lists the interrupt and exception table for the MAX32655's RV32 core.

Table 4-2: MAX32655 RV32 Interrupt Vector Table

Exception (Interrupt) Number	Name	Description
4	PF_IRQHandler	Power Fail interrupt
5	WDT0_IRQHandler	Windowed Watchdog Timer 0 Interrupt
6	GPIOWAKE_IRQHandler	GPIO Wakeup Interrupt
6	LPCMP_IRQHandler	Low-Power Comparator Interrupt
7	RTC_IRQHandler	RTC Interrupt
8	TMR0_IRQHandler	Timer 0 Interrupt
9	TMR1_IRQHandler	Timer 1 Interrupt
10	TMR2_IRQHandler	Timer 2 Interrupt
11	TMR3_IRQHandler	Timer 3 Interrupt
12	LPTMR0_IRQHandler	Low-Power Timer 0 (TMR4) Interrupt
13	LPTMR1_IRQHandler	Low-Power Timer 1 (TMR5) Interrupt
14	I2C0_IRQHandler	I^2C Port 0 Interrupt
15	UART0_IRQHandler	UART Port 0 Interrupt
16	-	Reserved
17	I2C1_IRQHandler	I^2C Port 1 Interrupt
18	UART1_IRQHandler	UART Port 1 Interrupt
19	UART2_IRQHandler	UART Port 2 Interrupt
20	I2C2_IRQHandler	I^2C Port 2 Interrupt
21	LPUART0_IRQHandler	Low-Power UART 0 (UART3) Interrupt
22	SPI1_IRQHandler	SPI Port 1 Interrupt
23	WUT_IRQHandler	Wakeup Timer Interrupt

Exception (Interrupt) Number	Name	Description
24	FLC0_IRQHandler	Flash Controller 0 Interrupt
25	GPIO0_IRQHandler	GPIO Port 0 Interrupt
26	GPIO1_IRQHandler	GPIO Port 1 Interrupt
27	GPIO2_IRQHandler	GPIO Port 2 Interrupt
28	DMA0_IRQHandler	DMA0 Interrupt
29	DMA1_IRQHandler	DMA1 Interrupt
30	DMA2_IRQHandler	DMA2 Interrupt
31	DMA3_IRQHandler	DMA3 Interrupt
32	BTLE_TX_DONE_IRQHandler	Bluetooth Transmitter Done Interrupt
33	BTLE_RX_RCVD_IRQHandler	Bluetooth Receive Data Interrupt
34	BTLE_RX_ENG_DET_IRQHandler	Bluetooth Receive Energy Detected Interrupt
35	BTLE_SFD_DET_IRQHandler	BTLE SFD Detected
36	BTLE_SFD_TO_IRQHandler	BTLE SFD Timeout
37	BTLE_GP_EVENT_IRQHandler	BTLE Timestamp
38	BTLE_CFO_IRQHandler	BTLE CFO Done
39	BTLE_SIG_DET_IRQHandler	BTLE Signal Detected
40	BTLE_AGC_EVENT_IRQHandler	BTLE AGC Event
41	BTLE_RFFE_SPIM_IRQHandler	BTLE RFFE SPIM Done
42	BTLE_TX_AES_IRQHandler	BTLE TX AES Done
43	BTLE_RX_AES_IRQHandler	BTLE RX AES Done
44	BTLE_INV_APB_ADDR_IRQHandler	BTLE Invalid APB Address
45	BTLE_IQ_DATA_VALID_IRQHandler	BTLE IQ Data Valid
46	AES_IRQHandler	AES Interrupt
47	TRNG_IRQHandler	TRNG Interrupt
48	LPWDT0_IRQHandler	Low-Power Watchdog Timer 0 (WDT1) Interrupt
49	DVS_IRQHandler	Digital Voltage Scaling Interrupt
50	SIMO_IRQHandler	Single Input Multiple Output Interrupt
51	-	Reserved
52	PT_IRQHandler	Pulse Train Interrupt
53	ADC_IRQHandler	ADC Interrupt
54	OWM_IRQHandler	1-Wire Master Interrupt
55	I2S_IRQHandler	I^2S Interrupt
56-59	-	Reserved

5. General-Purpose I/O and Alternate Function Pins (GPIO)

General-purpose I/O (GPIO) pins can be individually configured to operate in a digital I/O mode or in an alternate function (AF) mode that maps a signal associated with an enabled peripheral to that GPIO. The GPIO support dynamic switching between I/O mode and alternate function mode. Configuring a pin for an alternate function supersedes its use as a digital I/O, however the state of the GPIO can still be read through the *GPIOn_IN* register.

The electrical characteristics of a GPIO pin are identical whether the pin is configured as an I/O or as an alternate function, except where explicitly noted in the data sheet electrical characteristics tables.

GPIO are logically divided into ports of 32 pins. Package variants may not implement all pins of a specific 32-bit GPIO port.

Each pin of a port has an interrupt function that can be independently enabled and configured as a level- or edge-sensitive interrupt. All GPIOs of a given port share the same interrupt vector as detailed in the section *GPIO Interrupt Handling*.

Note: The register set used to control the GPIO are identical across multiple Maxim Integrated microcontrollers, however the behavior of several registers vary depending on the specific device. The behavior of the registers should not be assumed to be the same from one device to a different device. Specifically the registers GPIOn_PADCTRL0, GPIOn_PADCTRL1, GPIOn_HYSEN, GPIOn_SRSEL, GPIOn_DS0, GPIOn_DS1, and GPIOn_VSSEL are device dependent in their usage. GPIO3 is controlled differently and has different features than the other GPIO ports in the MAX32655. Details for using GPIO3 are covered in the system chapter.

The features for each GPIO pin include:

- Full CMOS outputs with configurable drive strength settings.
- Input modes/options:
  - High impedance
  - Weak pullup/pulldown
  - Strong pullup/pulldown
- Output data can be from *GPIOn_OUT* register or an enabled peripheral.
- Input data can be read from *GPIOn_IN* input register or the enabled peripheral.
- Bit set and clear registers for efficient bit-wise write access to the pins and configuration registers.
- Wake from low-power modes using edge triggered inputs.
- Selectable GPIO voltage supply for GPIO0, GPIO1, and GPIO2:
  - V_{DDIO}
  - V_{DDIOH}
- Selectable interrupt events:
  - Level triggered low
  - Level triggered high
  - Edge triggered rising edge
  - Edge triggered falling edge
  - Edge triggered rising and falling edge
- All GPIO pins default to input mode with weak-pullup during power-on-reset events.

5.1 Instances

Table 5-1 shows the number of GPIO available on each IC package. Some packages and part numbers do not implement all bits of a 32-bit GPIO port. Register fields corresponding to unimplemented GPIO contain indeterminate values and should not be modified.

Table 5-1: MAX32655 GPIO Pin Count

Package	GPIO	PINS
81-CTBGA	GPIO0[30:0]	31
	GPIO1[9:0]	10
	GPIO2[7:0]	8
	GPIO3[1:0]†	2

Note: See Power Sequencer Registers (PWRSEQ) for details on using GPIO3.

Note: Refer to the MAX32655 device data sheet for a description of alternate functions for each GPIO port pin.

5.2 Configuration

Each device pin can be individually configured as a GPIO or an alternate function. The correct alternate function setting must be selected for each pin of a given multi-pin peripheral for proper operation.

5.2.1 Power-On-Reset Configuration

During a POR event, all I/O default to GPIO mode as high impedance inputs except the SWDIO and SWDCLK pins. The SWD is enabled by default after POR with AF1 selected by hardware. See the *Secure Boot* chapter for exceptions.

Following a POR event, all GPIO except device pins that have the SWDIO and SWDCLK function, are configured with the following default settings:

- GPIO mode enabled
  - *GPIOn_EN0.en[pin]* = 1
  - *GPIOn_EN1.en[pin]* = 0
  - *GPIOn_EN2.en[pin]* = 0
- Pullup/Pulldown disabled, I/O in Hi-Z mode
  - *GPIOn_PADCTRL0.mode[pin]* = 0
  - *GPIOn_PADCTRL1.mode[pin]*
- Output mode disabled
  - *GPIOn_OUTEN.en[pin]* = 0
- Interrupt disabled
  - *GPIOn_INTEN.en[pin]* = 0

5.2.2 Serial Wire Debug Configuration

Perform the following steps to configure the SWDIO and SWDCLK device pins for SWD mode:

1. Set the device pin P0.28 for AF1 mode:
   a. *GPIOn_EN0.config[28]* = 0
   b. *GPIOn_EN1.config[28]* = 0
   c. *GPIOn_EN2.config[28]* = 0
2. Set device pin P0.29 for AF1 mode:
   a. *GPIOn_EN0.config[28]* = 0
   b. *GPIOn_EN1.config[29]* = 0
   c. *GPIOn_EN2.config[29]* = 0

*Note: To use the SWD pins in I/O mode, set the desired GPIO pins for SWD AF and set the SWD disable field to 1 (**GCR_**.swd_dis = 1).*

5.2.3 Pin Function Configuration

Table 5-2 depicts the bit settings for the GPIOn_EN0, GPIOn_EN1, and GPIOn_EN2 registers to configure the function of the GPIO port pins. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_EN0.config[25]*, *GPIO0_EN1.config[25]*, and *GPIO0_EN2.config[25]* all represent configuration for device pin P0.25. See Table 5-5 for a detailed example of how each of these bits applies to each of the GPIO device pins.

Table 5-2: MAX32655 GPIO Pin Function Configuration

MODE	GPIOn_EN0.config[pin]	GPIOn_EN1.config[pin]	GPIOn_EN2.config[pin]
AF1	0	0	0
AF2	0	1	0
I/O (transition to AF1)	1	0	0
I/O (transition to AF2)	1	1	0

5.2.4 Input Mode Configuration

Table 5-3 depicts the bit settings for the digital I/O input mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_PADCTRL1.config[25]*, *GPIO0_PADCTRL0.config[25]*, *GPIO0_PS.pull_sel[25]*, and *GPIO0_VSSEL.v_sel[25]* all represent configuration for device pin P0.25. See Table 5-8 for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the MAX32655 data sheet for details of specific electrical characteristics.

Table 5-3: MAX32655 Input Mode Configuration

Input Mode	Mode Select		Pullup/Pulldown Strength	Power Supply
	GPIOn_PADCTRL1.config[pin]	GPIOn_PADCTRL0.config[pin]	GPIOn_PS.pull_sel[pin]	GPIOn_VSSEL.v_sel[pin]
High-impedance	0	0	N/A	N/A
Weak Pullup to V_{DDIO} (1MΩ)	0	1	0	0
Strong Pullup to V_{DDIO} (25KΩ)	0	1	1	0
Weak Pulldown to V_{DDIOH} (1MΩ)	1	0	0	1
Strong Pulldown to V_{DDIOH} (25KΩ)	1	0	1	1
Reserved	1	1	N/A	N/A

5.2.5 Output Mode Configuration

Table 5-4 shows the configuration options for digital I/O in output mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO2_DS0.config[7], GPIO2_DS1.config[7], and GPIO2_VSSEL.v_sel[7]* all represent configuration for device pin P2.7. See Table 5-8 for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the MAX32655 data sheet for details of specific electrical characteristics.

Table 5-4: MAX32655 Output Mode Configuration

Input Mode	Drive Strength		Power Supply
	GPIOn_DS1.config[pin]	GPIOn_DS0.config[pin]	GPIOn_VSSEL.v_sel[pin]
Output Drive Strength 0, V_{DDIO} Supply	0	0	0
Output Drive Strength 1, V_{DDIO} Supply	0	1	0
Output Drive Strength 2, V_{DDIO} Supply	1	0	0

Input Mode	Drive Strength		Power Supply
	GPIOn_DS1.config[pin]	GPIOn_DS0.config[pin]	GPIOn_VSSEL.v_sel[pin]
Output Drive Strength 3, V_{DDIO} Supply	1	1	0
Output Drive Strength 0, V_{DDIOH} Supply	0	0	1
Output Drive Strength 1, V_{DDIOH} Supply	0	1	1
Output Drive Strength 2, V_{DDIOH} Supply	1	0	1
Output Drive Strength 3, V_{DDIOH} Supply	1	1	1

Each GPIO port is assigned a dedicated interrupt vector as shown in *Table 5-9*.

5.3 Reference Tables

The tables in this section provide example references for register bit assignment to configure a device's GPIO pins.

Table 5-5: MAX32655 GPIO Alternate Function Configuration Reference

Device Pin	Alternate Function Configuration Bits		
P0.0	GPIO0_EN0.config[0]	GPIO0_EN1.config[0]	GPIO0_EN2.config[0]
P0.1	GPIO0_EN0.config[1]	GPIO0_EN1.config[1]	GPIO0_EN2.config[1]
…	…	…	…
P0.30	GPIO0_EN0.config[30]	GPIO0_EN1.config[30]	GPIO0_EN2.config[30]
P0.31	GPIO0_EN0.config[31]	GPIO0_EN1.config[31]	GPIO0_EN2.config[31]

Table 5-6: MAX32655 GPIO Output/Input Configuration Reference

Device Pin	GPIO Output Enable	GPIO Output Write	GPIO Input Enable	GPIO Input Read
P0.0	GPIO0_OUTEN.en[0]	GPIO0_OUT.level[0]	GPIO0_INEN.en[0]	GPIO0_IN.level[0]
P0.1	GPIO0_OUTEN.en[1]	GPIO0_OUT.level[1]	GPIO0_INEN.en[1]	GPIO0_IN.level[1]
…	…	…	…	…
P0.30	GPIO0_OUTEN.en[30]	GPIO0_OUT.level[30]	GPIO0_INEN.en[30]	GPIO0_IN.level[30]
P0.31	GPIO0_OUTEN.en[31]	GPIO0_OUT.level[31]	GPIO0_INEN.en[31]	GPIO0_IN.level[31]

Table 5-7: MAX32655 GPIO Interrupt Configuration Reference

Device Pin	Enable	Status	Dual Edge	Polarity	Trigger	Wakeup
P0.0	GPIO0_INTEN.en[0]	GPIO0_INTFL.config[0]	GPIO0_DUALEDGE.dualedge[0]	GPIO0_INTPOL.pol[0]	GPIO0_INTMODE.gpio_intmode[0]	GPIO0_WKEN.en[0]
P0.1	GPIO0_INTEN.en[1]	GPIO0_INTFL.config[1]	GPIO0_DUALEDGE.config[1]	GPIO0_INTPOL.pol[1]	GPIO0_INTMODE.gpio_intmode[1]	GPIO0_WKEN.en[1]
…	…	…	…	…	…	…
P0.30	GPIO0_INTEN.en[30]	GPIO0_INTFL.int[30]	GPIO0_DUALEDGE.gpio_dualedge[30]	GPIO0_INTPOL.pol[30]	GPIO0_INTMODE.gpio_intmode[30]	GPIO0_WKEN.en[30]
P0.31	GPIO0_INTEN.en[31]	GPIO0_INTFL.int[31]	GPIO0_DUALEDGE.gpio_dualedge[31]	GPIO0_INTPOL.pol[31]	GPIO0_INTMODE.gpio_intmode[31]	GPIO0_WKEN.en[31]

Table 5-8: MAX32655 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration Reference

Device Pin	Pullup/Pulldown/Strength Select			Drive Strength		Voltage
P0.0	*GPIO0_PADCTRL0.config[0]*	*GPIO0_PADCTRL1.config[0]*	*GPIO0_PS.pull_sel[0]*	*GPIO0_DS0.config[0]*	*GPIO0_DS1.config[0]*	*GPIO0_VSSEL.v_sel[0]*
P0.1	*GPIO0_PADCTRL0.config[1]*	*GPIO0_PADCTRL1.config[1]*	*GPIO0_PS.pull_sel[1]*	*GPIO0_DS0.config[1]*	*GPIO0_DS1.config[1]*	*GPIO0_VSSEL.v_sel[1]*
…	…	…	…	…	…	…
P0.30	*GPIO0_PADCTRL0.config[30]*	*GPIO0_PADCTRL1.config[30]*	*GPIO0_PS.pull_sel[30]*	*GPIO0_DS0.config[30]*	*GPIO0_DS1.config[30]*	*GPIO0_VSSEL.v_sel[30]*
P0.31	*GPIO0_PADCTRL0.config[31]*	*GPIO0_PADCTRL1.config[31]*	*GPIO0_PS.pull_sel[31]*	*GPIO0_DS0.config[31]*	*GPIO0_DS1.config[31]*	*GPIO0_VSSEL.v_sel[31]*

5.4 Usage

5.4.1 Reset State

During a power-on-reset event, each GPIO is reset to the default input mode with the weak pullup resistor enabled as follows:

1. The GPIO configuration enable bits shown in *Table 5-2* are set to I/O (transition to AF1) mode.
2. Input mode is enabled (*GPIOn_INEN*.*en[pin]* = 1).
3. High impedance mode enabled (*GPIOn_PADCTRL1*.*config[pin]* = 0, *GPIOn_PADCTRL0*.*config[pin]* = 0), pullup and pulldown disabled.
4. Output mode disabled (*GPIOn_OUTEN*.*en[pin]* = 0)
5. Interrupt disabled (*GPIOn_INTEN*.*en[pin]* = 0)

5.4.2 Input Mode Configuration

Perform the following steps to configure one or more pins for input mode:

1. Set the GPIO configuration enable bits shown in *Table 5-2* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired as shown in *Table 5-3*.
3. Enable the input buffer connection to the GPIO pin by setting *GPIOn_INEN*.*en[pin]* to 1.
4. Read the input state of the pin using the *GPIOn_IN*.*level[pin]* field.

5.4.3 Output Mode Configuration

Perform the following steps to configure a pin for output mode:

1. Set the GPIO configuration eEnable bits shown in *Table 5-2* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired as shown in *Table 5-4*.
3. Set the output logic high or logic low using the *GPIOn_OUT*.*level[pin]* bit.
4. Enable the output buffer for the pin by setting *GPIOn_OUTEN*.*en[pin]* to 1.

5.4.4 Alternate Function Configuration

Most GPIO support one or more alternate functions selected with the GPIO configuration enable bits shown in *Table 5-2*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN0* = 1). The specific I/O

mode must match the desired AF. For example, if a transition to AF1 is desired, first select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO configuration enable bits shown in *Table 5-2* to the I/O mode that corresponds with the desired new AF setting. For example, select "I/O (transition to AF1)" if switching to AF1. Switching between different I/O mode settings does not affect the state or electrical characteristics of the pin.
2. Configure the electrical characteristics of the pin. See *Table 5-3* if the assigned alternate function uses the pin as an input. See *Table 5-4* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 5-2* to the desired alternate function.

5.5 Configuring GPIO (External) Interrupts

Each GPIO pin supports external interrupt events when the GPIO is configured for I/O mode and the input mode is enabled. If the GPIO is configured as a peripheral alternate function, the interrupts are peripheral-controlled.

GPIO interrupts can be individually enabled and configured as an edge or level triggered independently on a pin-by-pin basis. The edge trigger can be a rising, falling, or both transitions.

Each GPIO pin has a dedicated status bit in its corresponding *GPIOn_INTFL* register. A GPIO interrupt occurs when the status bit transitions from 0 to 1 if the corresponding bit is set in the corresponding *GPIOn_INTEN* register. Note that the interrupt status bit is always set when the current interrupt configuration event occurs, but an interrupt is only generated if explicitly enabled.

The following procedure details the steps for enabling ACTIVE mode interrupt events for a GPIO pin:

1. Disable interrupts by setting the *GPIOn_INTEN.en[pin]* field to 0. This prevents any new interrupts on the pin from triggering but does not clear previously triggered (pending) interrupts. The application can disable all interrupts for a GPIO port by writing 0 to the *GPIOn_INEN* register. To maintain previously enabled interrupts, read the *GPIOn_INEN* register and save the state prior to setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INTFL_CLR.clr[pin]* bit.
3. Configure the pin for the desired interrupt event
4. Set *GPIOn_INTMODE.mode[pin]* to select the desired interrupt.
5. For level triggered interrupts, the interrupt triggers on an input high (*GPIOn_INTPOL.pol[pin]* = 0) or input low level.
6. For edge triggered interrupts, the interrupt triggers on a transition from low to high(*GPIOn_INTPOL.pol[pin]* = 0) or high to low (*GPIOn_INTPOL.pol[pin]* = 1).
7. Optionally, set *GPIOn_DUALEDGE.de_en[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
   a. Set *GPIOn_INTEN.en[pin]* to 1 to enable the interrupt for the pin.

5.5.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector as shown in *Table 5-9*.

Table 5-9: MAX32655 GPIO Port Interrupt Vector Mapping

GPIO Interrupt Source	GPIO Interrupt Status Register	CM4 Interrupt Vector Number	RV32 Interrupt Vector Number	GPIO Interrupt Vector
GPIO0[31:0]	*GPIOn_INTFL*	40	25	GPIO0_IRQHandler
GPIO1[9:0]	*GPIOn_INTFL*	41	26	GPIO1_IRQHandler
GPIO2[7:0]	*GPIOn_INTFL*	42	27	GPIO2_IRQHandler

To handle GPIO interrupts in the interrupt vector handler, complete the following steps:

1. Read the *GPIOn_INTFL* register to determine the GPIO pin that triggered the interrupt.
2. Complete interrupt tasks associated with the interrupt source pin (application defined).
3. Clear the interrupt flag in the *GPIOn_INTFL* register by writing a 1 to the *GPIOn_INTFL_CLR* bit position that triggered the interrupt. This also clears and rearms the edge detectors for edge triggered interrupts.
4. Return from the interrupt vector handler.

5.5.2 Using GPIO for Wakeup from Low-Power Modes

Low-power modes support an asynchronous wakeup from edge triggered interrupts on the GPIO ports. Level triggered interrupts are not supported for wakeup because the system clock must be active to detect levels.

A single wakeup interrupt vector, GPIOWAKE_IRQHandler, is assigned for all pins of all GPIO ports. When the GPIO wakeup event occurs, the application software must interrogate each *GPIOn_INTFL* register to determine which external port pin caused the wakeup event.

Table 5-10: MAX32655 GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	CM4 Interrupt Vector Number	RV32 Interrupt Vector Number	GPIO Wakeup Interrupt Vector
GPIO0	*GPIO0_INTFL*	70	6	GPIOWAKE_IRQHandler
GPIO1	*GPIO1_INTFL*	70	6	GPIOWAKE_IRQHandler
GPIO2	*GPIO2_INTFL*	70	6	GPIOWAKE_IRQHandler

To enable low-power mode wakeup (*SLEEP*, *DEEPSLEEP*, *LPM*, *UPM*, and *BACKUP*) using an external GPIO interrupt, complete the following steps:

1. Clear pending interrupt flags by writing a logic 1 to *GPIOn_INTFL_CLR*.*clr[pin]*.
2. Activate the GPIO wakeup function by writing a logic 1 to *GPIOn_WKEN*.*we[pin]*.
3. Configure the power manager to use the GPIO as a wakeup source by ***GCR_PM***.*gpiowken* field to 1.

5.6 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 5-11*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 5-11: GPIO Register Summary

Offset	Register	Description
[0x0000]	*GPIOn_EN0*	*GPIO Port n Configuration Enable Bit 0 Register*
[0x0004]	*GPIOn_EN0_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 0 Register*
[0x0008]	*GPIOn_EN0_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 0 Register*
[0x000C]	*GPIOn_OUTEN*	*GPIO Port n Output Enable Register*
[0x0010]	*GPIOn_OUTEN_SET*	*GPIO Port n Output Enable Atomic Set Register*
[0x0014]	*GPIOn_OUTEN_CLR*	*GPIO Port n Output Enable Atomic Clear Register*
[0x0018]	*GPIOn_OUT*	*GPIO Port n Output Register*
[0x001C]	*GPIOn_OUT_SET*	*GPIO Port n Output Atomic Set Register*
[0x0020]	*GPIOn_OUT_CLR*	*GPIO Port n Output Atomic Clear Register*
[0x0024]	*GPIOn_IN*	*GPIO Port n Input Register*
[0x0028]	*GPIOn_INTMODE*	*GPIO Port n Interrupt Mode Register*
[0x002C]	*GPIOn_INTPOL*	*GPIO Port n Interrupt Polarity Register*
[0x0030]	*GPIOn_INEN*	*GPIO Port n Input Enable Register*
[0x0034]	*GPIOn_INTEN*	*GPIO Port n Interrupt Enable Register*
[0x0038]	*GPIOn_INTEN_SET*	*GPIO Port n Interrupt Enable Atomic Set Register*
[0x003C]	*GPIOn_INTEN_CLR*	*GPIO Port n Interrupt Enable Atomic Clear Register*
[0x0040]	*GPIOn_INTFL*	*GPIO Port n Interrupt Status Register*
[0x0048]	*GPIOn_INTFL_CLR*	*GPIO Port n Interrupt Clear Register*
[0x004C]	*GPIOn_WKEN*	*GPIO Port n Wakeup Enable Register*
[0x0050]	*GPIOn_WKEN_SET*	*GPIO Port n Wakeup Enable Atomic Set Register*
[0x0054]	*GPIOn_WKEN_CLR*	*GPIO Port n Wakeup Enable Atomic Clear Register*
[0x005C]	*GPIOn_DUALEDGE*	*GPIO Port n Interrupt Dual Edge Mode Register*
[0x0060]	*GPIOn_PADCTRL0*	*GPIO Port n Pad Configuration 1 Register*
[0x0064]	*GPIOn_PADCTRL1*	*GPIO Port n Pad Configuration 2 Register*
[0x0068]	*GPIOn_EN1*	*GPIO Port n Configuration Enable Bit 1 Register*
[0x006C]	*GPIOn_EN1_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 1 Register*
[0x0070]	*GPIOn_EN1_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 1 Register*
[0x0074]	*GPIOn_EN2*	*GPIO Port n Configuration Enable Bit 2 Register*
[0x0078]	*GPIOn_EN2_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 2 Register*
[0x007C]	*GPIOn_EN2_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 2 Register*
[0x00A8]	*GPIOn_HYSEN*	*GPIO Port n Hysteresis Enable Register*
[0x00AC]	*GPIOn_SRSEL*	*GPIO Port n Slew Rate Select Register*
[0x00B0]	*GPIOn_DS0*	*GPIO Port n Output Drive Strength Bit 0 Register*
[0x00B4]	*GPIOn_DS1*	*GPIO Port n Output Drive Strength Bit 1 Register*

Offset	Register	Description
[0x00B8]	GPIOn_PS	GPIO Port n Pulldown/Pullup Strength Select Register
[0x00C0]	GPIOn_VSSEL	GPIO Port n Voltage Select Register

5.6.1 GPIO Register Details

Table 5-12: GPIO Port n Configuration Enable Bit 0 Register

GPIO Port n Configuration Enable Bit 0			GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	1	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in Table 5-2 configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through GPIOn_EN0_SET or GPIOn_EN0_CLR. Table 5-5 depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

Table 5-13: GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_EN0_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 0** Writing 1 to one or more bits sets the corresponding bits in the GPIOn_EN0 register. 0: No effect. 1: Corresponding bits in GPIOn_EN0 register set to 1.

Table 5-14: GPIO Port n Configuration Enable Atomic Clear Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN0_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 0** Writing 1 to one or more bits clears the corresponding bits in the GPIOn_EN0 register. 0: No effect. 1: Corresponding bits in GPIOn_EN0 register cleared to 0.

Table 5-15: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO Output Enable** Set bit to 1 to enable the output driver for the corresponding GPIO pin. A bit can be enabled directly by writing to this register or indirectly through GPIOn_OUTEN_SET or GPIOn_OUTEN_CLR. 0: Pin is set to input mode; output driver disabled. 1: Pin is set to output mode.

Table 5-16: GPIO Port n Output Enable Atomic Set Register

GPIO Port n Output Enable Atomic Set			GPIOn_OUTEN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Output Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* set to 1.

Table 5-17: GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear			GPIOn_OUTEN_CLR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Output Enable Atomic Clear** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0.

Table 5-18: GPIO Port n Output Register

GPIO Port n Output			GPIOn_OUT	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	level	R/W	0	**GPIO Output** Set the corresponding output pin high or low. 0: Drive the corresponding output pin low (logic 0). 1: Drive the corresponding output pin high (logic 1).

Table 5-19: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Output Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* set to 1.

Table 5-20: GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear			GPIOn_OUT_CLR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	WO	0	**GPIO Output Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0.

Table 5-21: GPIO Port n Input Register

GPIO Port n Input			GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	level	RO	-	**GPIO Input** Returns the state of the input pin only if the corresponding bit in the *GPIOn_INEN* register is set. The state is not affected by the pin's configuration as an output or alternate function. 0: Input pin low (logic 0). 1: Input pin high (logic 1).

Table 5-22: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode			GPIOn_INTMODE	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	mode	R/W	0	**GPIO Interrupt Mode** Selects interrupt mode for the corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INTEN register is set.*

Table 5-23: GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity			GPIOn_INTPOL	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	pol	R/W	0	**GPIO Interrupt Polarity** Interrupt polarity selection bit for the corresponding GPIO pin. Level triggered mode (*GPIOn_INTMODE.mode[pin]* = 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INTMODE.mode[pin]*= 1): 0: Falling edge triggers interrupt. 1: Rising edge triggers interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INTEN register is set.*

Table 5-24: GPIO Port n Input Enable Register

GPIO Port n Input Enable			GPIOn_INEN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	1	**GPIO Input Enable** Connects the corresponding input pad to the specified input pin for reading the pin state using the *GPIOn_IN* register. 0: Input not connected. 1: Input pin connected to the pad for reading through the *GPIOn_IN* register.

Table 5-25: GPIO Port n Interrupt Enable Register

GPIO Port n Interrupt Enable			GPIOn_INTEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ie	R/W	0	**GPIO Interrupt Enable** Enable or disable the interrupt for the corresponding GPIO pin. 0: GPIO interrupt disabled. 1: GPIO interrupt enabled. *Note: Disabling a GPIO interrupt does not clear pending interrupts for the associated pin. Use the GPIOn_INTFL_CLR register to clear pending interrupts.*

Table 5-26: GPIO Port n Interrupt Enable Atomic Set Register

GPIO Port Interrupt Enable Atomic Set			GPIOn_INTEN_SET	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Interrupt Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the GPIOn_INTEN register. 0: No effect. 1: Corresponding bits in GPIOn_INTEN register set to 1.

Table 5-27: GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear			GPIOn_INTEN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Interrupt Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the GPIOn_INTEN register. 0: No effect. 1: Corresponding bits in GPIOn_INTEN register cleared to 0.

Table 5-28: GPIO Port n Interrupt Status Register

GPIO Port Interrupt Status			GPIOn_INTFL	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	if	RO	0	**GPIO Interrupt Status** An interrupt is pending for the associated GPIO pin when this bit reads 1. 0: No interrupt pending for associated GPIO pin. 1: GPIO interrupt pending for associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INTFL_CLR register to clear the interrupt pending status flag.*

Table 5-29: GPIO Port n Interrupt Clear Register

GPIO Port Interrupt Clear			GPIOn_INTFL_CLR	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1C	0	**GPIO Interrupt Clear** Write 1 to clear the associated interrupt status (GPIOn_INTFL). 0: No effect on the associated GPIOn_INTFL flag. 1: Clear the associated interrupt pending flag in the GPIOn_INTFL register.

Table 5-30: GPIO Port n Wakeup Enable Register

GPIO Port n Wakeup Enable			GPIOn_WKEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	we	R/W	0	**GPIO Wakeup Enable** Enable the I/O as a wakeup from low-power modes (*SLEEP*, *DEEPSLEEP*, *BACKUP*). 0: GPIO is not enabled as a wakeup source from low-power modes. 1: GPIO is enabled as a wakeup source from low-power modes.

Table 5-31: GPIO Port n Wakeup Enable Atomic Set Register

GPIO Port Wakeup Enable Atomic Set			GPIOn_WKEN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Wakeup Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_WKEN*r register. 0: No effect. 1: Corresponding bits in *GPIOn_WKEN* register set to 1.

Table 5-32: GPIO Port n Wakeup Enable Atomic Clear Register

GPIO Port Wakeup Enable Atomic Clear			GPIOn_WKEN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Wakeup Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_WKEN*r register. 0: No effect. 1: Corresponding bits in *GPIOn_WKEN* register cleared to 0.

Table 5-33: GPIO Port n Interrupt Dual Edge Mode Register

GPIO Port n Interrupt Dual Edge Mode			GPIOn_DUALEDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	de_en	R/W	0	**GPIO Interrupt Dual-Edge Mode Select** Setting this bit triggers interrupts on both the rising and falling edges of the corresponding GPIO if the associated *GPIOn_INTMODE* bit is set to edge triggered. The associated polarity (*GPIOn_INTPOL*) setting has no effect when this bit is set. 0: Disabled 1: Enabled

Table 5-34: GPIO Port n Pad Configuration 1 Register

GPIO Port n Pad Configuration 1			GPIOn_PADCTRL0	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Pad Configuration 1** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 5-3*.

Table 5-35: GPIO Port n Pad Configuration 2 Register

GPIO Port n Pad Configuration 2			GPIOn_PADCTRL1	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Pad Configuration 2** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 5-3*.

Table 5-36: GPIO Port n Configuration Enable Bit 1 Register

GPIO Port n Configuration Enable Bit 1			GPIOn_EN1	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Configuration Enable Bit 1** These bits, in conjunction with bits in *Table 5-2* configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN1_SET* or *GPIOn_EN1_CLR*. *Table 5-5* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

Table 5-37: GPIO Port n Configuration Enable Atomic Set Bit 1 Register

GPIO Port n Configuration Enable Atomic Set Bit 1			GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 1** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1.

Table 5-38: GPIO Port n Configuration Enable Atomic Clear Bit 1 Register

GPIO Port n Configuration Enable Atomic Clear Bit 1			GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 1** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0.

Table 5-39: GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2			GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in These bits, in conjunction with bits in *Table 5-2* configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN2_SET* or *GPIOn_EN2_CLR*. *Table 5-5* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

Table 5-40: GPIO Port n Configuration Enable Atomic Set Bit 2 Register

GPIO Port n Configuration Enable Atomic Set Bit 2			GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Alternate Function Select Atomic Set Bit 2** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1.

Table 5-41: GPIO Port n Configuration Enable Atomic Clear Bit 2 Register

GPIO Port n Configuration Enable Atomic Clear Bit 2			GPIOn_EN2_CLR	[0x007C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Alternate Function Select Atomic Clear Bit 2** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0.

Table 5-42: GPIO Port n Hysteresis Enable Register

GPIO Port n Hysteresis Enable			GPIOn_HYSEN	[0x00A8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	RO	0	**Reserved**

Table 5-43: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0			GPIOn_SRSEL	[0x00AC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	sr_sel	R/W	0	**Reserved**

Table 5-44: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0			GPIOn_DS0	[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Output Drive Strength Selection 0** See *Table 5-4* for details on how to set the GPIO output drive strength and other electrical characteristics.

Table 5-45: GPIO Port n Output Drive Strength Bit 1 Register

GPIO Port n Output Drive Strength Bit 1			GPIOn_DS1	[0x00B4]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO Output Drive Strength Selection 1** See *Table 5-4* for details on how to set the GPIO output drive strength and other electrical characteristics.

Table 5-46: GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n Pulldown/Pullup Strength Select			GPIOn_PS	[0x00B8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	pull_sel	R/W	0	**GPIO Pulldown/Pullup Strength Select** Selects the strength of the pullup or pulldown resistor for a pin configured for input mode. 0: Weak pulldown/pullup resistor for input pin. 1: Strong pulldown/pullup resistor for input pin. *Note: Refer to the data sheet for specific electrical characteristics of the pulldown/pullup resistances.*

Table 5-47: GPIO Port n Voltage Select Register

GPIO Port n Voltage Select			GPIOn_VSSEL	[0x00C0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	v_sel	R/W	0	**GPIO Supply Voltage Select** Selects the voltage rail used for the pin. 0: V_{DDIO} 1: V_{DDIOH}

6. Flash Controller (FLC)

The MAX32655 flash controller manages read, write, and erase accesses to the internal flas,h and provides the following features:

- Up to 512KB total internal flash memory
- 64 pages
- 8,192 bytes per page
- 2,048 words by 128 bits per page
- 128-bit data reads and writes
- Page erase and mass erase support
- Write protection

6.1 Instances

The device includes one instance of the FLC. The 512KB of internal flash memory is programmable through serial wire debug interface (in-system) or directly with software (in-application).

The flash is organized as an array of 2,048 words by 128 bits, or 8,192 bytes per page. *Table 6-1* shows the page start address, and page end address of the internal flash memory.

Table 6-1: MAX32655 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC0	1	8,192 Bytes	0x1000 0000	0x1000 1FFF
	2	8,192 Bytes	0x1000 2000	0x1000 3FFF
	3	8,192 Bytes	0x1000 4000	0x1000 5FFF
	4	8,192 Bytes	0x1000 6000	0x1000 7FFF
	…	…	…	…
	63	8,192 Bytes	0x1007 C000	0x1007 DFFF
	64	8,192 Bytes	0x1007 E000	0x1007 FFFF

6.2 Usage

The flash controller manages write and erase operations for internal flash memory, and provides a lock mechanism to prevent unintentional writes to the internal flash. In-application and in-system programming, page erase and mass erase operations are supported.

6.2.1 Clock Configuration

The FLC requires a 1MHz internal clock. See *Oscillator Sources* for details. Use the FLC clock divisor to generate $f_{FLCn\_CLK} =$ 1MHz, as shown in *Equation 6-1*. If using the IPO as the system clock, the *FLCn_CLKDIV.clkdiv* should be set to 100 (0x64).

Equation 6-1: FLC Clock Frequency

$$f_{FLCn\_CLK} = \frac{f_{SYS\_CLK}}{FLCn\_CLKDIV.clkdiv} = 1MHz$$

6.2.2 Lock Protection

A locking mechanism prevents accidental memory writes, and erases. All writes and erase operations require the *FLCn_CTRL.unlock* field be set to 2 prior to starting the operation. Writing any other value to this field, *FLCn_CTRL.unlock*, results in:

1. The flash instance remaining locked,

   or,
2. The flash instance becoming locked from the unlocked state.

Note: If a write, page erase or mass erase operation is started and the unlock code was not set to 2, the flash controller hardware sets the access fail flag, FLCn_INTR.af, to indicate an access violation occurred.

6.2.3 Flash Write Width

The FLC supports write widths of 128-bits only. The target address bits *FLCn_ADDR[3:0]* are ignored resulting in 128-bit address alignment.

Table 6-2: Valid Addresses Flash Writes

	FLCn_ADDR[31:0]																														
Bit Number	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
128-bit Write	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0

6.2.4 Flash Write

Writes to a flash address are only successful if the target address is already in its erased state. Perform the following steps to write to a flash memory address:

1. If desired, enable the flash controller interrupts by setting the *FLCn_INTR.afie* and *FLCn_INTR.doneie* bits.
2. Read the *FLCn_CTRL.pend* bit until it returns 0.
3. Configure the *FLCn_CLKDIV.clkdiv* field to achieve a 1MHz frequency based on the selected SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid target address, see *Table 6-2* for details.
5. Set *FLCn_DATA3*, *FLCn_DATA2*, *FLCn_DATA1*, and *FLCn_DATA0* to the data to write.
   a. *FLCn_DATA3* is the most significant word and *FLCn_DATA0* is the least significant word.
      i. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte and the most significant byte is stored at the highest-numbered byte.
6. Set the *FLCn_CTRL.unlock* field to 2 to unlock the flash.
7. Set the *FLCn_CTRL.wr* field to 1.
   a. The hardware automatically clears this field when the write operation is complete.
8. The *FLCn_INTR.done* field is set to 1 by the hardware when the write completes and an interrupt is generated if the *FLCn_INTR.doneie* is set to 1.
   a. If an error occurred, the *FLCn_INTR.af* field is set to 1 by the hardware and an interrupt is generated if the *FLCn_INTR.afie* field is set to 1.
9. Set the *FLCn_CTRL.unlock* field to any value other than 2 to re-lock the flash.

Note: Code execution can occur within the same flash instance as targeted programming.

6.2.5 Page Erase

CAUTION: Care must be taken to not erase the page from which the software is currently executing.

Perform the following to erase a page of a flash memory instance:

1. If desired, enable flash controller interrupts by setting the *FLCn_INTR.afie* and *FLCn_INTR.doneie bits*.
2. Read the *FLCn_CTRL.pend* bit until it returns 0.
3. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to an address within the target page to be erased. *FLCn_ADDR[12:0]* are ignored by the FLC to ensure the address is page aligned.
5. Set *FLCn_CTRL.unlock* to 2 to unlock the flash instance.
6. Set *FLCn_CTRL.erase_code* to 0x55 for page erase.
7. Set *FLCn_CTRL.pge* to 1 to start the page erase operation.
8. The *FLCn_CTRL.pend* bit is set by the flash controller while the page erase is in progress and the *FLCn_CTRL.pge* and *FLCn_CTRL*.pend are cleared by the flash controller when the page erase is complete.
9. *FLCn_INTR.done* is set by the hardware when the page erase completes and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
10. Set *FLCn_CTRL.unlock* to any value other than 2 to re-lock the flash instance.

6.2.6 Mass Erase

CAUTION: Care must be taken to not erase the flash from which the software is currently executing.

Mass erase clears the internal flash memory on an instance basis. Perform the following steps to mass erase a single flash memory instance:

1. Read the *FLCn_CTRL.pend* bit until it returns 0.
2. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
3. Set *FLCn_CTRL.unlock* to 2 to unlock the internal flash.
4. Set *FLCn_CTRL.erase_code* to 0xAA for mass erase.
5. Set *FLCn_CTRL.me* to 1 to start the mass erase operation.
6. The *FLCn_CTRL.pend* bit is set by the flash controller while the mass erase is in progress and the *FLCn_CTRL.me* and *FLCn_CTRL.pend* are cleared by the flash controller when the mass erase is complete.
7. *FLCn_INTR.done* is set by the flash controller when the mass erase completes and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
8. Set *FLCn_CTRL.unlock* to any value other than 2 to re-lock the flash instance.

6.3 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 6-3*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 6-3: Flash Controller Register Summary

Offset	Register Name	Access	Description
[0x0000]	*FLCn_ADDR*	R/W	Flash Controller Address Pointer Register
[0x0004]	*FLCn_CLKDIV*	R/W	Flash Controller Clock Divisor Register
[0x0008]	*FLCn_CTRL*	R/W	Flash Controller Control Register

Offset	Register Name	Access	Description
[0x0024]	*FLCn_INTR*	R/W	Flash Controller Interrupt Register
[0x0030]	*FLCn_DATA0*	R/W	Flash Controller Data Register 0
[0x0034]	*FLCn_DATA1*	R/W	Flash Controller Data Register 1
[0x0038]	*FLCn_DATA2*	R/W	Flash Controller Data Register 2
[0x003C]	*FLCn_DATA3*	R/W	Flash Controller Data Register 3
[0x0040]	*FLCn_ACNTL*	R/W	Flash Controller Access Control Register
[0x0080]	*FLCn_WELR0*	R/W	Flash Write/Erase Lock 0 Register
[0x0088]	*FLCn_WELR1*	R/W	Flash Write/Erase Lock 1 Register
[0x0090]	*FLCn_RLR0*	R/W	Flash Read Lock 0 Register
[0x0098]	*FLCn_RLR1*	R/W	Flash Read Lock 1Register

6.3.1 Register Details

Table 6-4: Flash Controller Address Pointer Register

Flash Controller Address Pointer		FLCn_ADDR		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0x1000 0000	**Flash Address** This field contains the target address for a write operation. A valid internal flash memory address is required for all write operations.

Table 6-5: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor		FLCn_CLKDIV		[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	clkdiv	R/W	0x64	**Flash Controller Clock Divisor** The APB clock is divided by the value in this field to generate the FLCn peripheral clock, f_{FLCn_CLK}. The FLCn peripheral clock must equal 1MHz. The default on POR, System Reset and Watchdog reset is 100, resulting in f_{FLCn_CLK} = 1MHz when IPO is the system oscillator. The FLCn peripheral clock is only used during erase and program functions and not during read functions. See *Clock Configuration* for additional details.

Table 6-6: Flash Controller Control Register

Flash Controller Control		FLCn_CTRL		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 2, prior to any flash write or erase operation to unlock the flash. Writing any other value to this field locks the internal flash. 2: Flash unlock code
27:26	-	RO	-	**Reserved**
25	lve	R/W	0	**Low Voltage Enable** Set this field to 1 to enable low voltage operation for the flash memory. 0: Low voltage operation disabled (Default). 1: Low voltage operation enabled. *Note: The PWRSEQ_LPCN.ovr field must be set to 0b00 prior to setting this field to 1.*

Flash Controller Control			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all flash registers except the *FLCn_INTR* register are ignored by the Flash Controller. This bit is cleared by hardware once the flash is accessible. *Note: If the Flash Controller is busy (FLCn_CTRL.pend = 1), reads, writes and erase operations are not allowed and result in an access failure (FLCn_INTR.af = 1).* 0: Flash idle. 1: Flash busy.
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Prior to an erase operation, this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked prior to setting the erase code. This field is automatically cleared after the erase operation is complete. 0x00: Erase disabled. 0x55: Page erase code. 0xAA: Enable mass erase.
7:5	-	RO	0	**Reserved**
4	width	R/W	0	**Width** Do not modify this field.
3	-	RO	0	**Reserved**
2	pge	R/W1	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR.addr*. The flash must be unlocked prior to attempting a page erase, see *FLCn_CTRL.unlock* for details. The Flash Controller hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.
1	me	R/W1	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked prior to attempting a mass erase, see *FLCn_CTRL.unlock* for details. The Flash Controller hardware clears this bit when the mass erase operation completes. 0: No operation. 1: Initiate mass erase.
0	wr	R/W1O	0	**Write** If this field reads 0, no write operation is pending for the flash. To initiate a write operation, set this bit to 1 and the Flash Controller writes to the address set in the *FLCn_ADDR* register. 0: No write operation in process or write operation complete. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Table 6-7: Flash Controller Interrupt Register

Flash Controller Interrupt			FLCn_INTR	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**

Flash Controller Interrupt			FLCn_INTR	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
9	afie	R/W	0	**Flash Access Fail Interrupt Enable** Set this bit to 1 to enable interrupts on flash access failures. 0: Disabled 1: Enabled
8	doneie	R/W	0	**Flash Operation Complete Interrupt Enable** Set this bit to 1 to enable interrupts on flash operations complete. 0: Disabled 1: Enabled
7:2	-	RO	0	**Reserved**
1	af	R/W0C	0	**Flash Access Fail Interrupt Flag** This bit is set when an attempt is made to write or erase the flash while the flash is busy or locked. Only the hardware can set this bit to 1. Writing a 1 to this bit has no effect. This bit is cleared by writing a 0. 0: No access failure has occurred. 1: Access failure occurred.
0	done	R/W0C	0	**Flash Operation Complete Interrupt Flag** This flag is automatically set by the hardware after a flash write or erase operation completes. 0: Operation not complete or not in process. 1: Flash operation complete.

Table 6-8: Flash Controller Data 0 Register

Flash Controller Data 0			FLCn_DATA0	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 0** Flash data for bits 31:0.

Table 6-9: Flash Controller Data Register 1

Flash Controller Data 1			FLCn_DATA1	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 1** Flash data for bits 63:32

Table 6-10: Flash Controller Data Register 2

Flash Controller Data 2			FLCn_DATA2	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 2** Flash data for bits 95:64

Table 6-11: Flash Controller Data Register 3

Flash Controller Data 3			FLCn_DATA3	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 3** Flash data for bits 127:96.

Table 6-12: Flash Controller Access Control Register

Flash Controller Access Control			FLCn_ACNTL	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	actrl	R/W	0	**Access Control** Access the information block by writing the access control sequence. See *Information Block Flash Memory* for details.

Table 6-13: Flash Write/Lock 0 Register

Flash Write/Lock 0			FLCn_WELR0	[0x0080]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr0	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_WELR0*[0] maps to page 0 of the flash and *FLCn_WELR0*[31] maps to page 31. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 6-14: Flash Write/Lock 1 Register

Flash Write/Lock 1			FLCn_WELR1	[0x0088]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr1	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_WELR1*[0] maps to page 32 of the flash and *FLCn_WELR1*[31] maps to page 63. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected.

Table 6-15: Flash Read Lock 0 Register

Flash Read Lock 0			FLCn_RLR0	[0x0090]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr0	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_RLR0*[0] maps to page 0 of the flash and *FLCn_RLR0*[31] maps to page 31. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

Table 6-16: Flash Read Lock 1 Register

Flash Read Lock 1			FLCn_RLR1	[0x0098]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr1	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_RLR1*[0] maps to page 32 of the flash and *FLCn_RLR1*[31] maps to page 63. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

7. Debug Access Port (DAP)

Some device versions might provide an Arm debug access port (DAP) that supports debugging during application development. Refer to the ordering information in the device data sheet to determine if a specific part number supports a customer-accessible DAP. Parts with a customer-accessible DAP should only be used for development and never used in a final customer product.

GCR_SYSST.icelock = 0 if the device provides a customer-accessible DAP.

7.1 Instances

The DAP interface communicates through the serial wire debug (SWD) and/or JTAG interface signals shown in *Table 7-1*.

Table 7-1: MAX32655 DAP Instances

Instance	Pin	Alternate Function	SWD Signal	JTAG Signal	Pullup
DAP	P0.28	AF1	SWDIO	N/A	100kΩ to V_{DD}
	P0.29	AF1	SWCLK	N/A	

7.2 Access Control

7.2.1 Factory Disabled DAP

Device versions that do not provide a DAP interface have *GCR_SYSST.icelock* = 1 set at the factory, permanently disabling the DAP interface. No software action is needed to secure these devices.

7.2.2 Software Accessible DAP

Device versions that provide a DAP (*GCR_SYSST.icelock* = 0) always have their interface(s) enabled and running unless the software explicitly sets *GCR_SYSCTRL.sw_dis* field to 1. The read-only field *GCR_SYSST.icelock* is cleared to 0 and the software has read and write access to the *GCR_SYSCTRL.sw_dis* field. The *GCR_SYSCTRL.sw_dis* field resets to 0 after every POR to allow access to the DAP during development.

Software can disable the DAP by setting the *GCR_SYSCTRL.sw_dis* field to 1. The only practical application for disabling the DAP is to release the interface pins to operate as standard GPIO or in one of the supported alternate function modes, in a development environment. Customers can use device versions with the DAP enabled for development but should only use device versions with the factory disabled DAP in a final product.

7.3 Pin Configuration

Instances of SWD or JTAG signals in GPIO and Alternate Function matrices are for determining which GPIO pins are associated with a signal. It is not necessary to configure a pin for an alternate function to use the DAP following a POR.

By default, the pin associated with the bidirectional SWDIO/TMS signal is configured as a GPIO high-impedance input after any POR. While the DAP is in use, a pullup resistor should be connected to the SWDIO/TMS pin as shown in *Table 7-1*. The pullup ensures the signal is in a known state when control of the SWDIO/TMS pin is transferred between the host and target. The pullup resistor should be removed if the associated pin is used as a GPIO to avoid unnecessary current consumption.

8. Semaphores

The semaphore peripheral allows multiple cores in a system to cooperate when accessing shared resources. The peripheral contains eight semaphore registers that can be atomically set and cleared. Reading the status field of a semaphore register returns the current state of the status field, and if the field is 0 automatically sets the status to 1. The semaphore status register reflects the state of each of the semaphore register's status. The status register enables checking each of the semaphore's states but is read only, it is not guaranteed that the semaphore status fields will not change after checking the status register's value.

It is left to the discretion of the software architect to decide how and when the semaphores are used and how they are allocated. Existing hardware does not have to be modified for this type of cooperative sharing, and the use of semaphores is exclusively within the software domain.

The semaphore peripheral includes two general purpose mailbox registers that enable communication between the RV32 and CM4 cores. Additionally, either core can generate a semaphore interrupt for either the CM4 or the RV32 providing immediate notification of communication through the mailbox registers.

8.1 Instances

There is one instance of the semaphore peripheral, shown in *Table 8-1*.

Table 8-1: MAX32655 Semaphore Instances

Instance	Number of Semaphores
SEMA	8

8.2 Multiprocessor Communications

The semaphore includes support for multicore communications through two mailbox registers and provides the ability to generate an RV32 semaphore interrupt and a CM4 semaphore interrupt.

The mailbox registers, *SEMA_MAIL0* and *SEMA_MAIL1*, are general purpose 32-bit registers. The CM4 and RV32 have read and write access to both registers. Application firmware should manage how these registers are used to prevent collisions if both cores attempt to modify the registers at the same time.

8.2.1 Reset

Globally reset the semaphore peripheral by setting *GCR_RST1.smphr* to 1.

8.2.2 CM4 Semaphore Interrupt Generation

The *SEMA_IRQ0* register can generate a CM4 semaphore interrupt. Setting the *SEMA_IRQ0.cm4_irq* bit to 1 and then setting the *SEMA_IRQ0.en* bit to 1 generates a CM4 semaphore interrupt. The CM4 interrupt handler should write the *SEMA_IRQ0.en and/or the SEMA_IRQ0.cm4_irq* field(s) to 0 to clear the interrupt condition.

8.2.3 RV32 Semaphore Interrupt Generation

The *SEMA_IRQ1* register can generate a RV32 semaphore interrupt. Setting the *SEMA_IRQ1.rv32_irq* bit to 1 and then setting the *SEMA_IRQ1.en* bit to 1 generates a RV32 semaphore interrupt. The RV32 interrupt handler should write the *SEMA_IRQ1.en and/or the SEMA_IRQ1.rv32_irq* field(s) to 0 to clear the interrupt condition.

8.3 Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 8-2: Semaphore Register Summary

Offset	Register	Name
[0x0000]	*SEMA_SEMIPHORES0*	Semaphore 0 Register
[0x0004]	*SEMA_SEMIPHORES1*	Semaphore 1 Register
[0x0008]	*SEMA_SEMIPHORES2*	Semaphore 2 Register
[0x000C]	*SEMA_SEMIPHORES3*	Semaphore 3 Register
[0x0010]	*SEMA_SEMIPHORES4*	Semaphore 4 Register
[0x0014]	*SEMA_SEMIPHORES5*	Semaphore 5 Register
[0x0018]	*SEMA_SEMIPHORES6*	Semaphore 6 Register
[0x0020]	*SEMA_SEMIPHORES7*	Semaphore 7 Register
[0x0040]	*SEMA_IRQ0*	Semaphore Interrupt 0 Register
[0x0044]	*SEMA_MAIL0*	Semaphore Mailbox 0 Register
[0x0048]	*SEMA_IRQ1*	Semaphore Interrupt 1 Register
[0x004C]	*SEMA_MAIL1*	Semaphore Mailbox 1 Register
[0x0100]	*SEMA_STATUS*	Semaphore Status Register

8.3.1 Register Details

Table 8-3: Semaphore 0 Register

Semaphore 0			SEMA_SEMIPHORES0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status0* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-4: Semaphore 1 Register

Semaphore 1			SEMA_SEMIPHORES1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status1* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-5: Semaphore 2 Register

Semaphore 2			SEMA_SEMIPHORES2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status2* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-6: Semaphore 3 Register

Semaphore 3			SEMA_SEMIPHORES3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status3* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-7: Semaphore 4 Register

Semaphore 4			SEMA_SEMIPHORES4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status4* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-8: Semaphore 5 Register

Semaphore 5			SEMA_SEMIPHORES5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status5* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-9: Semaphore 6 Register

Semaphore 6			SEMA_SEMIPHORES6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

Semaphore 6			SEMA_SEMIPHORES6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status6* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-10: Semaphore 7 Register

Semaphore 7			SEMA_SEMIPHORES7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and if 0, automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status7* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 8-11: Semaphore Interrupt 0 Register

Semaphore Interrupt 0			SEMA_IRQ0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	cm4_irq	R/W	0	**CM4 Interrupt** The RV32 can use this bit to communicate with the CM4 through the semaphore interrupt. The RV32 generates a semaphore interrupt for the CM4 by setting this field to 1 and also setting the *SEMA_IRQ0.en* bit to 1.
15:1	-	RO	0	**Reserved**
0	en	R/W	0	**Interrupt Enable** Set this field to enable interrupt generation on semaphore events. 0: Disabled 1: Enabled

Table 8-12: Semaphore Mailbox 0 Register

Semaphore Mailbox 0			SEMA_MAIL0	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register is readable and writable by both the CM4 and RV32 cores allowing communication between the two cores. In conjunction with the *SEMA_IRQ0* register, the RV32 can write data to this register and then notify the CM4 by generating a semaphore interrupt. Alternately, the CM4 can write to this register and then notify the RV32 using the *SEMA_IRQ1* register to generate an RV32 semaphore interrupt event. *Note: The management of the SEMA_MAIL0 and SEMA_MAIL1 registers is left to the software. It is recommended that one mailbox is used for communication from the CM4 to the RV32 and the other mailbox register is used for communication from the RV32 to the CM4. However, there are no hardware read/write restrictions on the mailbox registers.*

Table 8-13: Semaphore Interrupt 1 Register

Semaphore Interrupt 1			SEMA_IRQ1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	rv32_irq	R/W	0	**RV32 Interrupt** The CM4 can use this bit to communicate with the RV32 through the semaphore interrupt. The CM4 generates a semaphore interrupt for the RV32 by setting this field to 1 and also setting the *SEMA_IRQ1.en* bit to 1. 0: RV32 interrupt event not active or received by RV32. 1: RV32 interrupt event is generated when the *SEMA_IRQ1.en* bit is also set to 1.
15:1	-	RO	0	**Reserved**
0	en	R/W	0	**Interrupt Enable** Set this field to generate a RV32 semaphore interrupt when the *SEMA_IRQ1.rv32_irq* is also set to 1. The RV32 should write this bit to 0 when a semaphore interrupt is generated to prevent repeat interrupt generation. 0: Disabled 1: Enabled

Table 8-14: Semaphore Mailbox 1 Register

Semaphore Mailbox 1			SEMA_MAIL1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register is readable and writable by both the CM4 and RV32 cores allowing communication between the two cores. In conjunction with the *SEMA_IRQ0* register, the RV32 can write data to this register and then notify the CM4 by generating a semaphore interrupt. Alternately, the CM4 can write to this register and then notify the RV32 using the *SEMA_IRQ1* register to generate an RV32 semaphore interrupt event. *Note: The management of the SEMA_MAIL0 and SEMA_MAIL1 registers is left to the software. It is recommended that one mailbox is used for communication from the CM4 to the RV32 and the other mailbox register is used for communication from the RV32 to the CM4. However, there are no hardware read/write restrictions on the mailbox registers.*

Table 8-15: Semaphore Status Register

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	status7	R	0	**Semaphore 7 Status** This field mirrors the semaphore 7 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES7.status* is 0. 1: *SEMA_SEMIPHORES7.status* is 1.
6	status6	R	0	**Semaphore 6 Status** This field mirrors the semaphore 6 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES6.status* is 0. 1: *SEMA_SEMIPHORES6.status* is 1.

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
5	status5	R	0	**Semaphore 5 Status** This field mirrors the semaphore 5 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES5*.*status* is 0. 1: *SEMA_SEMIPHORES5*.*status* is 1.
4	status4	R	0	**Semaphore 4 Status** This field mirrors the semaphore 4 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES4*.*status* is 0. 1: *SEMA_SEMIPHORES4*.*status* is 1.
3	status3	R	0	**Semaphore 3 Status** This field mirrors the semaphore 3 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES3*.*status* is 0. 1: *SEMA_SEMIPHORES3*.*status* is 1.
2	status2	R	0	**Semaphore 2 Status** This field mirrors the semaphore 2 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES2*.*status* is 0. 1: *SEMA_SEMIPHORES2*.*status* is 1.
1	status1	R	0	**Semaphore 1 Status** This field mirrors the semaphore 1 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES1*.*status* is 0. 1: *SEMA_SEMIPHORES1*.*status* is 1.
0	status0	R	0	**Semaphore 0 Status** This field mirrors the semaphore 0 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMIPHORES0*.*status* is 0. 1: *SEMA_SEMIPHORES0*.*status* is 1.

9. Standard DMA (DMA)

The DMA peripheral is a hardware feature that can perform high-speed, block memory transfers of data independent of a CPU core. All DMA transactions consist of a burst read from a source location into the internal DMA FIFO followed by a burst write from the internal DMA FIFO to the destination location.

DMA transfers are one of three types:

- From a receive FIFO to a memory address.
- To a transmit FIFO from a memory address.
- From a source memory address to a destination memory address.

The DMA supports multiple channels. Each channel provides the following features:

- Full 32-bit source and destination addresses with 24-bit (16 Mbytes) address increment capability.
- Ability to chain DMA buffers when a count-to-zero (CTZ) condition occurs.
- Interrupt upon CTZ.
- Up to 16 Mbytes for each DMA transfer.
- 8 × 32 byte transmit and receive FIFO.
- Programmable channel timeout period.
- Programmable burst size.
- Programmable priority levels.
- Abort on error.

9.1 Instances

There is one instance of the DMA, generically referred to as DMA. Each instance provides four channels, generically referred to as DMA_CHn. Each instance of the DMA has a set of interrupt registers common to all its channels, and a set of registers unique to each channel instance.

Table 9-1: MAX32655 DMA and Channel Instances

DMA Instance	DMA_CHn Channel Instance
DMA	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3

9.2 DMA Channel Operation (DMA_CH)

9.2.1 DMA Channel Arbitration and DMA Bursts

The DMA peripheral contains an internal arbiter that allows enabled channels to access the AHB and move data. Once a channel is programmed and enabled, it generates a request to the arbiter immediately (for memory-to-memory DMA) or whenever its associated peripheral requests DMA (for memory-to-peripheral or peripheral-to-memory DMA).

Granting is done based on priority; a higher priority request is always granted. Within a given priority level, requests are granted on a round-robin basis. The *DMA_CHn_CTRL*.*pri* field determines the DMA channel priority.

When a channel's request is granted, the channel runs a DMA transfer. The arbiter grants requests to a single channel at a time. Once the DMA transfer completes, the channel relinquishes its grant.

A DMA channel is enabled using the *DMA_CHn_CTRL*.*en* bit.

When disabling a channel, poll the *DMA_CHn_STATUS.status* bit to determine if the channel is truly disabled. In general, *DMA_CHn_STATUS.status* follows the setting of the *DMA_CHn_CTRL.en* bit. However, the *DMA_CHn_STATUS.status* bit is automatically cleared under the following conditions:

- Bus error (cleared immediately)
- CTZ when the *DMA_CHn_CTRL.rlden* = 0 (cleared at the end of the AHB R/W burst)
- *DMA_CHn_CTRL.en* bit transitions to 0 (cleared at the end of the AHB R/W burst)

Whenever *DMA_CHn_STATUS.status* transitions from 1 to 0, the corresponding *DMA_CHn_CTRL.en* bit is also cleared. If an active channel is disabled during an AHB read/write burst, the current burst is continued until completed.

Only an error condition can interrupt an ongoing data transfer.

9.2.2 DMA Source and Destination Addressing

The source and destination for DMA transfers are dictated by the request select dedicated to the peripheral instance. The *DMA_CHn_CTRL.request* field dictates the source and destination for a channel's DMA transfer as shown in *Table 9-2*. The *DMA_CHn_SRC* and *DMA_CHn_DST* registers hold the source and/or destination memory addresses, depending on the specific operation.

The *DMA_CHn_CTRL.srcinc* field is ignored when the DMA source is a peripheral memory, and the *DMA_CHn_CTRL.dstinc* field is ignored when the DMA destination is a peripheral memory.

Table 9-2: DMA Source and Destination by Peripheral

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0	Memory-to-Memory	*DMA_CHn_SRC*	*DMA_CHn_DST*
1	SPI1	SPI1 Receive FIFO	*DMA_CHn_DST*
2-3	Reserved	-	-
4	UART0	UART0 Receive FIFO	*DMA_CHn_DST*
5	UART1	UART1 Receive FIFO	*DMA_CHn_DST*
6	Reserved	-	-
7	I^2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
8	I^2C1	I2C1 Receive FIFO	*DMA_CHn_DST*
9	ADC	ADC Data Register	*DMA_CHn_DST*
10	I^2C2	I2C2 Receive FIFO	*DMA_CHn_DST*
11-13	Reserved	-	-
14	UART2	UART2 Receive FIFO	*DMA_CHn_DST*
15	SPI0	SPI0 Receive FIFO	*DMA_CHn_DST*
16-27	Reserved	-	-
28	LPUART0 (UART3)	LPUART0 (UART3) Receive FIFO	*DMA_CHn_DST*
29	Reserved	-	-
30	I^2S	I^2S Data Register	*DMA_CHn_DST*
31-32	Reserved	-	-
33	SPI1	*DMA_CHn_SRC*	SPI1 Transmit FIFO
34-35	Reserved	-	-
36	UART0	*DMA_CHn_SRC*	UART0 Transmit FIFO

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
37	UART1	DMA_CHn_SRC	UART1 Transmit FIFO
38	Reserved	-	-
39	I2C0	DMA_CHn_SRC	I2C0 Transmit FIFO
40	I2C1	DMA_CHn_SRC	I2C1 Transmit FIFO
41	Reserved	-	-
42	I2C2	DMA_CHn_SRC	I2C2 Transmit FIFO
43	Reserved	-	-
44	CRC	DMA_CHn_SRC	CRC Data Register
45	Reserved	-	-
46	UART2	DMA_CHn_SRC	UART2 Transmit FIFO
47	SPI0	DMA_CHn_SRC	SPI0 Transmit FIFO
48-59	Reserved	-	-
60	LPUART0 (UART3)	DMA_CHn_SRC	LPUART0 (UART3) Transmit FIFO
61	Reserved	-	-
62	I2S	DMA_CHn_SRC	I2S Data Register
63	Reserved	-	-

9.2.3 Data Movement from Source to DMA

Table 9-3 shows the fields that control the burst movement of data into the DMA FIFO. The source is a peripheral or memory.

Table 9-3: Data Movement from Source to DMA FIFO

Register/Field	Description	Comments
DMA_CHn_SRC	Source address	If the increment enable is set, this increments on every read cycle of the burst. This field is ignored when the DMA source is a peripheral.
DMA_CHn_CNT	Number of bytes to transfer before a CTZ condition occurs	This register is decremented on each read of the burst.
DMA_CHn_CTRL.*burst_size*	Burst size (1-32)	This maximum number of bytes moved during the burst read.
DMA_CHn_CTRL.*srcwd*	Source width	This determines the maximum data width used during each read of the AHB burst (byte, two bytes, or four bytes). The actual AHB width might be less if DMA_CHn_CNT is not great enough to supply all the needed bytes.
DMA_CHn_CTRL.*srcinc*	Source increments enable	Increments DMA_CHn_SRC. This field is ignored when the DMA source is a peripheral.

9.2.4 Data Movement from DMA to Destination

Table 9-4 shows the fields that control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

Table 9-4: Data Movement from the DMA FIFO to Destination

Register/Field	Description	Comments
DMA_CHn_DST	Destination address	If the increment enable is set, this increments on every write cycle of the burst. This field is ignored when the DMA destination is a peripheral.
DMA_CHn_CTRL.burst_size	Burst size (1-32)	The maximum number of bytes moved during a single AHB read/write burst.
DMA_CHn_CTRL.dstwd	Destination width	This determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes).
DMA_CHn_CTRL.dstinc	Destination address increment enable	Increments *DMA_CHn_DST*. This field is ignored when the DMA destination is a peripheral.

9.3 Usage

Use the following procedure to perform a DMA transfer from a peripheral's receive FIFO to memory, from memory to a peripheral's transmit FIFO, or from memory to memory.

1. Ensure *DMA_CHn_CTRL.en, DMA_CHn_CTRL.rlden* = 0, and *DMA_CHn_STATUS.ctz_if* = 0.
2. If using memory for the destination of the DMA transfer, configure *DMA_CHn_DST* to the starting address of the destination in memory.
3. If using memory for the source of the DMA transfer, configure *DMA_CHn_SRC* to the starting address of the source in memory.
4. Write the number of bytes to transfer to the *DMA_CHn_CNT* register.
5. Configure the following *DMA_CHn_CTRL* register fields in one or more instructions. Do not set *DMA_CHn_CTRL.en* to 1 or *DMA_CHn_CTRL.rlden* to 1 in this step:
    a. Configure *DMA_CHn_CTRL.request* to select the transfer operation associated with the DMA channel.
    b. Configure *DMA_CHn_CTRL.burst_size* for the desired burst size.
    c. Configure *DMA_CHn_CTRL.pri* to set the channel priority relative to other DMA channels.
    d. Configure *DMA_CHn_CTRL.dstwd* to dictate the number of bytes written in each transaction.
    e. If desired, set *DMA_CHn_CTRL.dstinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
    f. Configure *DMA_CHn_CTRL.srcwd* to dictate the number of bytes read in each transaction.
    g. If desired, set *DMA_CHn_CTRL.srcinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
    h. If desired, set *DMA_CHn_CTRL.dis_ie* = 1 to generate an interrupt when the channel becomes disabled. The channel becomes disabled when the DMA transfer completes, or a bus error occurs.
    i. If desired, set *DMA_CHn_CTRL.ctz_ie* 1 to generate an interrupt when the *DMA_CHn_CNT* register is decremented to zero.
    j. If using the reload feature, configure the reload registers to set the destination, source, and count for the following DMA transaction.
        1) Load the *DMA_CHn_SRCRLD* register with the source address reload value.
        2) Load the *DMA_CHn_DSTRLD* register with the destination address reload value.
        3) Load the *DMA_CHn_CNTRLD* register with the count reload value.
    k. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CTRL.to_clkdiv* to 0 to disable the channel timeout feature.
6. Set *DMA_CHn_CTRL.rlden* to 1 to enable the reload feature if using.
7. Set *DMA_CHn_CTRL.en* = 1 to immediately start the DMA transfer.
8. Wait for the interrupt flag to become 1 to indicate the completion of the DMA transfer.

9.4 Count-To-Zero (CTZ) Condition

When an AHB channel burst completes, a CTZ condition exists if *DMA_CHn_CNT* is decremented to 0.

At this point, there are two possible responses depending on the value of the *DMA_CHn_CTRL.rlden*:

1. If *DMA_CHn_CTRL.rlden* = 1, then the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers are loaded from the reload registers, and the channel remains active and continues operating using the newly-loaded address/count values and previously programmed configuration values.
2. If *DMA_CHn_CTRL.rlden* = 0, then the channel is disabled, and *DMA_CHn_STATUS.status* is cleared.

9.5 Chaining Buffers

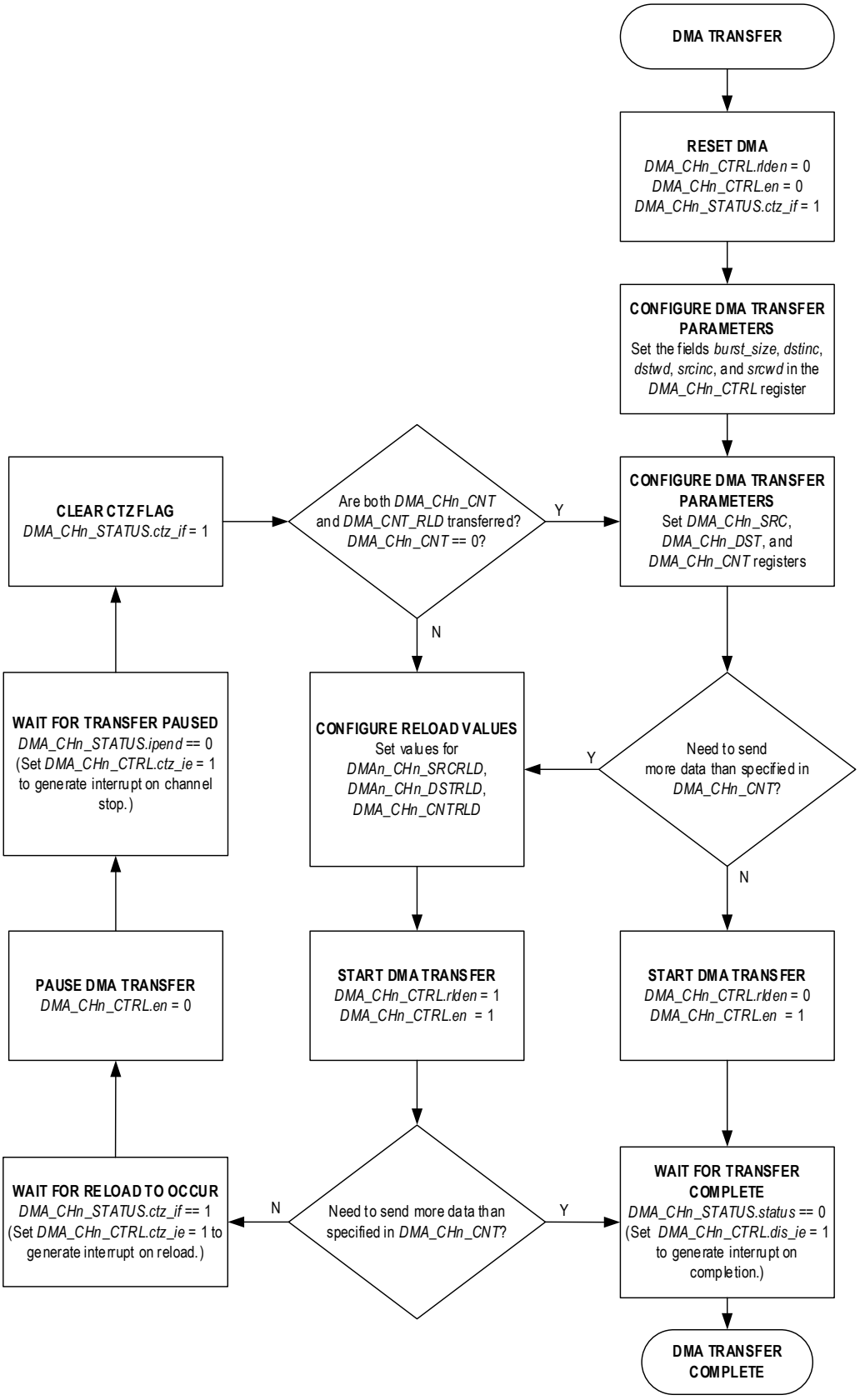
Chaining buffers reduces the DMA ISR response time and allows DMA to service requests without intermediate processing from the CPU. *Figure 9-1* shows the procedure for generating a DMA transfer using one or more chain buffers.

Configure the following reload registers to configure a channel for chaining:

- *DMA_CHn_SRC*
- *DMA_CHn_DST*
- *DMA_CHn_CNT*
- *DMA_CHn_SRCRLD*
- *DMA_CHn_DST*
- *DMA_CHn_CNTRLD*

Writing to any register while a channel is disabled is supported, but there are certain restrictions when a channel is enabled. The *DMA_CHn_STATUS.status* bit indicates if the channel is enabled or not. Because an active channel might be in the middle of an AHB read/write burst, do not write to the *DMA_CHn_SRC*, *DMA_CHn_DST*, or *DMA_CHn_CNT* registers while a channel is active (*DMA_CHn_STATUS.status* = 1). To disable any DMA channel, clear the *DMA_INTEN.ch<n>_ien* bit. Then, poll the *DMA_CHn_STATUS.status* bit to verify the channel is disabled.

Figure 9-1: DMA Block-Chaining Flowchart



9.6 DMA Interrupts

Enable interrupts for each channel by setting *DMA_INTEN.ch<n>_ien*. When an interrupt for a channel is pending, the corresponding *DMA_INTFL.ch<n>_ipend* = 1. Set the corresponding enable bit to cause an interrupt when the flag is set.

A channel interrupt (*DMA_CHn_STATUS.ipend* = 1) is caused by:

- *DMA_CHn_CTRL.ctz_ie* = 1
  - If enabled all CTZ occurrences set the *DMA_CHn_STATUS.ipend* bit.
- *DMA_CHn_CTRL.dis_ie* = 1
  - If enabled, any clearing of the *DMA_CHn_STATUS.status* bit sets the *DMA_CHn_STATUS.ipend* bit. Examine the *DMA_CHn_STATUS* register to determine which reasons caused the disable. The *DMA_CHn_CTRL.dis_ie* bit also enables the *DMA_CHn_STATUS.to_if* bit. The *DMA_CHn_STATUS.to_if* bit does not clear the *DMA_CHn_STATUS.status* bit.

To clear the channel interrupt, write 1 to the cause of the interrupt (the *DMA_CHn_STATUS.ctz_if*, *DMA_CHn_STATUS.rld_if*, *DMA_CHn_STATUS.bus_err*, or *DMA_CHn_STATUS.to_if* bits).

When running in normal mode without buffer chaining (*DMA_CHn_CTRL.rlden* = 0), set the *DMA_CHn_CTRL.dis_ie* bit only. An interrupt is generated upon DMA completion or an error condition (bus error or timeout error).

When running in buffer chaining mode (*DMA_CHn_CTRL.rlden* = 1), set both the *DMA_CHn_CTRL.dis_ie* and *DMA_CHn_CTRL.ctz_ie* bits. The CTZ interrupts occur on completion of each DMA (count reaches zero and reload occurs). The setting of *DMA_CHn_CTRL.dis_ie* ensures that an error condition generates an interrupt. If *DMA_CHn_CTRL.ctz_ie* = 0, then the only interrupt occurs when the DMA completes and *DMA_CHn_CTRL.rlden* = 0 (final DMA).

9.7 Channel Timeout Detect

Each channel can optionally generate an interrupt when its associated peripheral does not request a transfer in a user-configurable period. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CTRL.to_clkdiv*, and *DMA_CHn_CTRL.to_per* shown in *Table 9-5*. A channel timeout event is generated if the timer is not reset by one of the events listed below before the timeout period expires.

Table 9-5: DMA Channel Timeout Configuration

DMA_CHn_CTRL.to_clkdiv	**Timeout Period (µs)**
0	Channel timeout disabled
1	$\frac{2^8 \times [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
2	$\frac{2^{16} \times [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
3	$\frac{2^{24} \times [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$

The start of the timeout period is controlled by *DMA_CHn_CTRL*.*to_wait*:

- If *DMA_CHn_CTRL*.*to_wait* = 0, the timer begins counting immediately after *DMA_CHn_CTRL*.*to_per* is configured to a value other than 0 and the channel is enabled.
- If *DMA_CHn_CTRL*.*to_wait* = 1, the timer begins counting when the first DMA request is received from the peripheral.

The timer is reset whenever:

- The DMA request line programmed for the channel is activated.
- The channel is disabled for any reason (*DMA_CHn_STATUS*.*status* = 0).

If the timeout timer period expires, hardware sets *DMA_CHn_STATUS*.*to_if* = 1 to indicate a channel timeout event has occurred. A channel timeout does not disable the DMA channel.

9.8 Memory-to-Memory DMA

Memory-to-memory transfers are processed as if the request is always active. This means that the DMA channel generates an almost constant request for the bus until its transfer is complete. For this reason, assign a lower priority to channels executing memory-to-memory transfers to prevent starvation of other DMA channels.

9.9 DMA Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 9-6: DMA Register Summary

Offset	Register	Description
[0x0000]	*DMA_INTEN*	DMA Control register
[0x0004]	*DMA_INTFL*	DMA Interrupt Status register

9.9.1 DMA Register Details

Table 9-7: DMA Interrupt Flag Register

DMA Interrupt Enable			DMA_INTEN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ch<*n*>_ien	R/W	0	**DMAn Channel Interrupt Enable** Each bit in this field enables the corresponding channel interrupt *n* in *DMA_INTFL*. Register bits associated with unimplemented channels should not be changed from their default reset value. 0: Disabled 1: Enabled

Table 9-8: DMA Interrupt Enable Register

DMA Interrupt Enable			DMA_INTFL	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ch<*n*>_ipend	RO	0	**DMA Channel Interrupt Flag** Each bit in this field represents an interrupt for the corresponding channel interrupt <n>. To clear an interrupt, clear the corresponding active interrupt bit in the *DMA_CHn_STATUS* register. An interrupt bit in this field is set only if the corresponding interrupt enable field is set in the *DMA_INTEN* register. Register bits associated with unimplemented channels should be ignored. 0: No interrupt 1: Interrupt pending

9.10 DMA Channel Registers

Table 9-9: Standard DMA Channel 0 to Channel 3 Register Summary

Offset	DMA Channel	Description
[0x0100]	DMA_CH0	DMA Channel 0
[0x0120]	DMA_CH1	DMA Channel 1
[0x0140]	DMA_CH2	DMA Channel 2
[0x0160]	DMA_CH3	DMA Channel 3

9.10.1 DMA Channel n Register Details

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 9-10*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 9-10: DMA Channel Registers Summary

Offset	Register	Description
[0x0000]	*DMA_CHn_CTRL*	DMA Channel n Configuration Register
[0x0004]	*DMA_CHn_STATUS*	DMA Channel n Status Register
[0x0008]	*DMA_CHn_SRC*	DMA Channel n Source Register
[0x000C]	*DMA_CHn_DST*	DMA Channel n Destination Register
[0x0010]	*DMA_CHn_CNT*	DMA Channel n Count Register
[0x0014]	*DMA_CHn_SRCRLD*	DMA Channel n Source Reload Register
[0x0018]	*DMA_CHn_DSTRLD*	DMA Channel n Destination Reload Register
[0x001C]	*DMA_CHn_CNTRLD*	DMA Channel n Count Reload Register

Table 9-11: DMA_CHn Control Register

DMA Channel n Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31	ctz_ie	R/W	0	**CTZ Interrupt Enable** 0: Disabled 1: Enabled. *DMA_INTFL.ch<n>_ipend* is set to 1 whenever a CTZ event occurs.
30	dis_ie	R/W	0	**Channel Disable Interrupt Enable** 0: Disabled 1: Enabled. *DMA_INTFL.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_STATUS*.status changes from 1 to 0.
29	-	RO	0	**Reserved**
28:24	burst_size	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0: 1 byte 1: 2 bytes 2: 3 bytes ...: ... 31: 32 bytes
23	-	RO	0	**Reserved**
22	dstinc	R/W	0	**Destination Increment Enable** This bit enables the automatic increment of the *DMA_CHn_DST* register upon every AHB transaction. This bit is ignored for a DMA transmit to peripherals. 0: Disabled 1: Enabled
21:20	dstwd	R/W	0	**Destination Width** Indicates the width of each AHB transaction to the destination peripheral or memory (the actual width might be less than this if there are insufficient bytes in the DMA FIFO for the full width). 0: One byte 1: Two bytes 2: Four bytes 3: Reserved
19	-	RO	0	**Reserved**
18	srcinc	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled 1: Enabled
17:16	srcwd	R/W	0	**Source Width** Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMA_CHn_CNT* register indicates a smaller value. 0: One byte 1: Two bytes 2: Four bytes 3: Reserved

DMA Channel n Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
15:14	to_clkdiv	R/W	0	**Timeout Timer Clock Pre-Scale Select** Selects the Pre-Scale divider for the timer clock input. 0: Timer disabled. 1: $\frac{f_{HCLK}}{28}$ 2: $\frac{f_{HCLK}}{216}$ 3: $\frac{f_{HCLK}}{224}$
13:11	to_per	R/W	0	**Timeout Period Select** Selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers 0: 3 to 4 1: 7 to 8 2: 15 to 16 3: 31 to 32 4: 63 to 64 5: 127 to 128 6: 255 to 256 7: 511 to 512
10	to_wait	R/W	0	**Request DMA Timeout Timer Wait Enable** This field controls when the timeout timer starts, either immediately when the timeout timer is enabled or after the first DMA transaction occurs. 0: Start timer immediately when enabled. 1: Delay timer start until after the first DMA transaction occurs.
9:4	request	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 9-2*.
3:2	pri	R/W	0	**Channel Priority** Sets the priority of the channel relative to other channels of the DMA peripheral. Channels of the same priority are serviced in a round-robin fashion. 0: High 1: Medium high 2: Medium low 3: Low
1	rlden	R/W	0	**Reload Enable** Setting this bit to 1 allows reloading the *DMA_CHn_SRC*, *DMA_CHn_DST* and *DMA_CHn_CNT* registers upon a CTZ. When this bit is set to 0 and a CTZ occurs, the channel is disabled and the *DMA_CHn_STATUS.status* bit is set to 0. 0: The channel is disabled when a CTZ occurs and the *DMA_CHn_STATUS.status* is set to 0. 1: Automatically reload the *DMA_CHn_SRC*, *DMA_CHn_DST* and *DMA_CHn_CNT* registers on a CTZ.

DMA Channel n Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_STATUS*.*status* changes from 1 to 0. 0: Disabled 1: Enabled

Table 9-12: DMA Status Register

DMA Channel n Status			DMA_CHn_STATUS	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	to_if	R/W1C	0	**Timeout Interrupt Flag** Timeout. Write 1 to clear. 0: No time out. 1: A channel time out has occurred.
5	-	RO	0	**Reserved**
4	bus_err	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred and the channel was disabled by the hardware. Write 1 to clear. 0: No error found. 1: An AHB bus error occurred .
3	rld_if	R/W1C	0	**Reload Interrupt Flag** Reload. Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.
2	ctz_if	R/W1C	0	**CTZ Interrupt Flag** Write 1 to clear. 0: CTZ has not occurred. 1: CTZ has occurred.
1	ipend	RO	0	**Channel Interrupt Pending** 0: No interrupt. 1: Interrupt pending.
0	status	RO	0	**Channel Status** This bit indicates when it is safe to change the configuration, address, and count registers for the channel. Whenever this bit is cleared by the hardware, the *DMA_CHn_CTRL*.*en* bit is also cleared. 0: Disabled 1: Enabled

Table 9-13: DMA Channel n Source Register

DMA Channel n Source			DMA_CHn_SRC	[0x0108]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Source Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CTRL*.*srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width. If *DMA_CHn_CTRL*.*srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CTRL*.*rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_SRCRLD* register.

Table 9-14: DMA Channel n Destination Register

DMA Channel n Destination			DMA_CHn_DST	[0x010C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Destination Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CTRL*.*dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width. If a CTZ condition occurs while *DMA_CHn_CTRL*.*rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_DSTRLD* register.

Table 9-15: DMA Channel n Count Register

DMA Channel n Count			DMA_CHn_CNT	[0x0110]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:0	cnt	R/W	0	**DMA Counter** Load this register with the number of bytes to transfer. This field decreases on every AHB access to the DMA FIFO. The decrement is one, two, or four bytes depending on the data width. When the counter reaches 0, a CTZ condition is triggered. If a CTZ condition occurs while *DMA_CHn_CTRL*.*rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_CNTRLD*.*cnt_rld* field.

Table 9-16: DMA Channel n Source Reload Register

DMA Channel n Source Reload			DMA_CHn_SRCRLD	[0x0114]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:0	addr	R/W	0	**Source Address Reload Value** If *DMA_CHn_CTRL*.*rlden* = 1, then the value of this register is loaded into *DMA_CHn_SRC* upon a CTZ condition.

Table 9-17: DMA Channel n Destination Reload Register

DMA Channel n Destination Reload			DMA_CHn_DSTRLD	[0x0118]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:0	addr	R/W	0	**Destination Address Reload Value** If *DMA_CHn_CTRL*.*rlden* = 1, then the value of this register is loaded into *DMA_CHn_DST* upon a CTZ condition.

Table 9-18: DMA Channel n Count Reload Register

DMA Channel n Count Reload			DMA_CHn_CNTRLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:0	cnt	R/W	0	**Count Reload Value** If *DMA_CHn_CTRL*.*rlden* = 1, then the value of this register is loaded into *DMA_CHn_CNT* upon a CTZ condition.

10. Analog-to-Digital Converter (ADC) and Comparators

The ADC is a 10-bit sigma-delta ADC with a single-ended input multiplexer and an integrated reference generator. The multiplexer selects an input channel from either of the 8 external analog input signals or the internal power supply inputs. The external analog input signals are defined as alternate functions on GPIO as shown in *Table 10-1*.

The 10-bit ADC conversions are stored as a 16-bit value selectable as most-significant bit (MSB) or least-significant bit (LSB) aligned. The 8 external analog inputs can be configured by the software as 4 two-input comparators with interrupt capabilities. Comparator 0, COMP0, is configurable to wake the device from *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP*. The remaining three comparators, COMP1, COMP2, and COMP3, are configurable as wakeup sources from *SLEEP*, *LPM*, and *UPM*.

10.1 Features

- Maximum 8MHz ADC clock rate.
- Two reference source options:
  - An internal 1.22V bandgap.
  - $\frac{V_{DDA}}{2}$ supply.
- 8 external analog inputs configurable as 4 two-input comparators.
- 8 internal power supply monitor inputs.
- Fixed 10-bit word conversion time of 1024 ADC clock cycles.
- Programmable out-of-range (limit) detection.
- Interrupt generation for limit detection, conversion start, conversion complete, and internal reference powered on.
- Serial ADC data measurements.
- ADC conversion 10-bit output either MSB or LSB aligned.

10.2 Instances

Table 10-1: MAX32655 ADC Input Pins for the 81-CTBGA Package

Function	81 CTBGA Pin	81 CTBGA Alternate Function
AIN0/AIN0N	P2.0	AF1
AIN1/AIN0P	P2.1	AF1
AIN2/AIN1N	P2.2	AF1
AIN3/AIN1P	P2.3	AF1
AIN4/AIN2N	P2.4	AF1
AIN5/AIN2P	P2.5	AF1
AIN6/AIN3N	P2.6	AF1
AIN7/AIN3P	P2.7	AF1

10.3 Architecture

The ADC is a first-order sigma-delta converter with a 10-bit output. The ADC operates at a maximum frequency of 8MHz with a fixed-sample rate as shown in *Equation 10-1*. Details of selecting the ADC clock frequency, f_{adcclk}, are covered in the *Clock Configuration* section.

Equation 10-1: ADC 10-bit Word Sample Rate

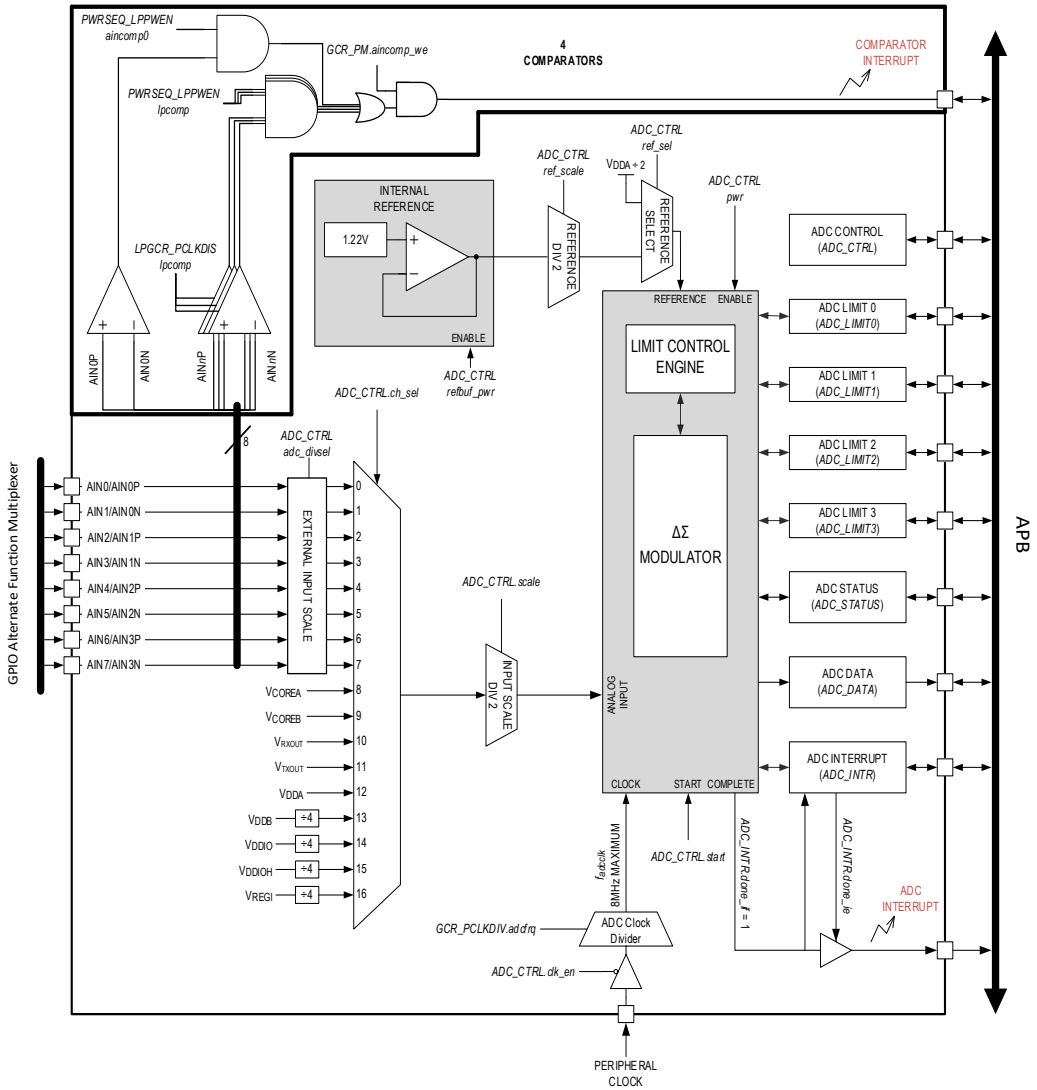
$$t_{adc\_sample} = 1024 \times \left(\frac{1}{f_{adcclk}}\right)$$

The ADC offset is factory trimmed and automatically loaded into the ADC controller during system power-up.

The ADC uses a switched capacitor network to perform the conversion; this results in dynamic switching current and requires settling time for the external analog input signals (AIN0 – AIN7). This dynamic switching current sets the upper limit of the source impedance of the external analog input signals to approximately 10kΩ.

The ADC supports a gain of 2 × to provide additional conversion resolution if the input signals are less than half the reference voltage.

Figure 10-1: Analog-to-Digital Converter Block Diagram



10.4 Clock Configuration

The ADC clock, *adcclk*, is controlled by the *GCR_PCLKDIV.adcfrq* register field. Configure this field to achieve the target ADC sample frequency. The maximum clock frequency supported by the ADC is 8MHz. The divisor selection, *GCR_PCLKDIV.adcfrq*, for the ADC depends on the peripheral clock. *Equation 10-2* shows the calculation for the ADC clock.

Equation 10-2: ADC Clock Frequency

$$f_{adcclk} = \frac{f_{PCLK}}{GCR\_PCLKDIV.adcfrq}$$

The *GCR_PCLKDIV.adcfrq* field setting must result in a value for $f_{adcclk} \leq 8MHz$ as shown in *Table 10-2* with IPO set as the system clock.

Table 10-2: MAX32655 ADC Clock Frequency and ADC Conversion Time with the System Clock set to the IPO

GCR_PCLKDIV.adcfrq	ADC Clock Frequency (Hz) f_{adcclk}	10-Bit Word Conversion Time (µs) t_{adc_sample}
0 – 7	Invalid	Invalid
8	6,250,000	164
9	5,555,555	184
10	5,000,000	205
11	4,545,454	225
12	4,166,666	246
13	3,846,153	266
14	3,571,428	287
15	3,333,333	307

10.5 Power-Up Sequence

Complete the following steps to configure the ADC:

1. Disable the ADC clock by setting the *ADC_CTRL.clk_en* field to 0.
2. Set the ADC clock (f_{adcclk}) using the *GCR_PCLKDIV.adcfrq* field. See *Clock Configuration* for details.
3. Enable the ADC clock by setting the *ADC_CTRL.clk_en* field to 1
4. Clear the ADC reference ready interrupt flag by writing a 1 to *ADC_INTR.ref_ready_if*.
5. Optionally enable the ADC reference ready interrupt by setting the *ADC_INTR.ref_ready_ie* field to 1 and enable the ADC interrupt handler (ADC_IRQHandler).
6. Select one of the following ADC reference sources:
   a. Internal 1.22V bandgap reference (*ADC_CTRL.ref_sel* = 0).
   b. $\frac{V_{DDA}}{2}$ reference (*ADC_CTRL.ref_sel* = 1).
7. Complete the following steps to enable power to the ADC and optionally the internal ADC reference:
   a. Set *ADC_CTRL.pwr* to 1 to turn on the ADC.
   b. Set *ADC_CTRL.refbuf_pwr* to 1 to turn on the internal reference buffer If using the internal reference.
   c. Wait until hardware sets the *ADC_INTR.ref_ready_if* field to 1 indicating the internal reference is fully powered on and ready.
   d. Clear the ADC reference ready interrupt flag by writing 1 to *ADC_INTR.ref_ready_if*.
   e. Optionally disable the ADC reference ready interrupt by clearing the *ADC_INTR.ref_ready_ie* field to 0.

10.6 Conversion

After the power-up sequence is complete, the ADC is ready for data conversion. Complete the following steps to perform a data conversion.

1. Select the ADC input channel for the conversion by setting the *ADC_CTRL.ch_sel* field. See *ADC Channel Select* for details.
2. Optionally set input and reference scaling. See *Scale Limitations for All Other Input Channels* for details on each input channel's scale requirements.
3. Set the data alignment for the conversion output data using the *ADC_CTRL.data_align* field.
   a. 0 for LSB alignment or 1 for MSB alignment. See *Table 10-3* for alignment details of the *ADC_DATA* register.
4. Clear the ADC done interrupt flag by writing 1 to the *ADC_INTR.done_if* field.
5. Optionally enable the ADC done interrupt (*ADC_INTR.done_ie* = 1) and enable the ADC interrupt vector (ADC_IRQHandler).
6. Start the ADC conversion by setting the *ADC_CTRL.start* field to 1.
7. Poll the *ADC_INTR.done_if* flag until it reads 1 or wait for the ADC interrupt to occur if enabled.
8. Read the data from the *ADC_DATA.data* field and clear the ADC done interrupt flag by writing 1 to the *ADC_INTR.done_if* field.

10.6.1 Data Conversion Output Alignment

The ADC outputs a total of 10-bits per conversion and stores the data in the DATA register LSB justified by default. *Table 10-3* shows the ADC data alignment based on the value of the *ADC_CTRL.data_align* bit.

Table 10-3: ADC Data Register Alignment Options

***ADC_CTRL.data_align* = 0**																
	MSB															**LSB**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADC_DATA	0	0	0	0	0	0	data									

***ADC_CTRL.data_align* = 1**																
	MSB															**LSB**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADC_DATA	data										0	0	0	0	0	0

10.6.2 Data Conversion Value Equations

Use the following equations to calculate the ADC data value for a conversion for the selected channel. If using the internal reference, V_{REF} = 1.22V; otherwise V_{REF} = V_{DDA}.

Equation 10-3: ADC Data Calculation for Input Signal ADC_CTRL.ch_sel = 0 through 7 (AIN0 - AIN7)

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{Input\ Signal}{2^{scale} * (adc\_divsel + 1)}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: Must satisfy Equation 10-6.

Equation 10-4: ADC Data Equation for Input Signal ADC_CTRL.ch_sel = 8 through 12 (V_{COREA}, V_{COREB}, V_{RXOUT}, V_{TXOUT}, V_{DDA})

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{Input\ Signal}{2^{scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 10-4 for limitations.

Equation 10-5: ADC Data Calculation Input Signal ADC_CTRL.ch_sel = 14 through 16 (V_{DDIO}, V_{DDIOH}, V_{REGI})

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{\frac{Input\ Signal}{4}}{2^{scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 10-4 for limitations.

10.7 Reference Scaling and Input Scaling

For small signals, the ADC input, ADC reference, or both can be scaled by 50%. This enables flexibility to achieve better resolution on the ADC conversion. Each input channel supports the default of no scaling of the input (*ADC_CTRL.scale* = 0) and no scaling of the reference (*ADC_CTRL.ref_scale* = 0). The following sections describe the scale options for each of the ADC input channels.

10.7.1 AIN0 – AIN7 Scale Limitations

The external inputs, AIN0 through AIN7, support scaling of the input by 50%, the reference by 50%, or both by 50%. Also, the scaling can further be modified by additional factors of 2, 3, or 4 as defined by *ADC_CTRL.adc_divsel*. The scale settings for the given input signal and reference must satisfy *Equation 10-6* to be valid:

Equation 10-6: Input and Reference Scale Requirements Equation

$$\frac{AINn}{2^{scale}} < \frac{V_{REF}}{2^{ref\_scale}}$$

10.7.2 Scale Limitations for All Other Input Channels

For the remaining internal input channels, the scale settings must either both be disabled, or both be enabled as shown in *Table 10-4*.

Table 10-4: MAX32655 Input and Reference Scale Support by ADC Input Channel

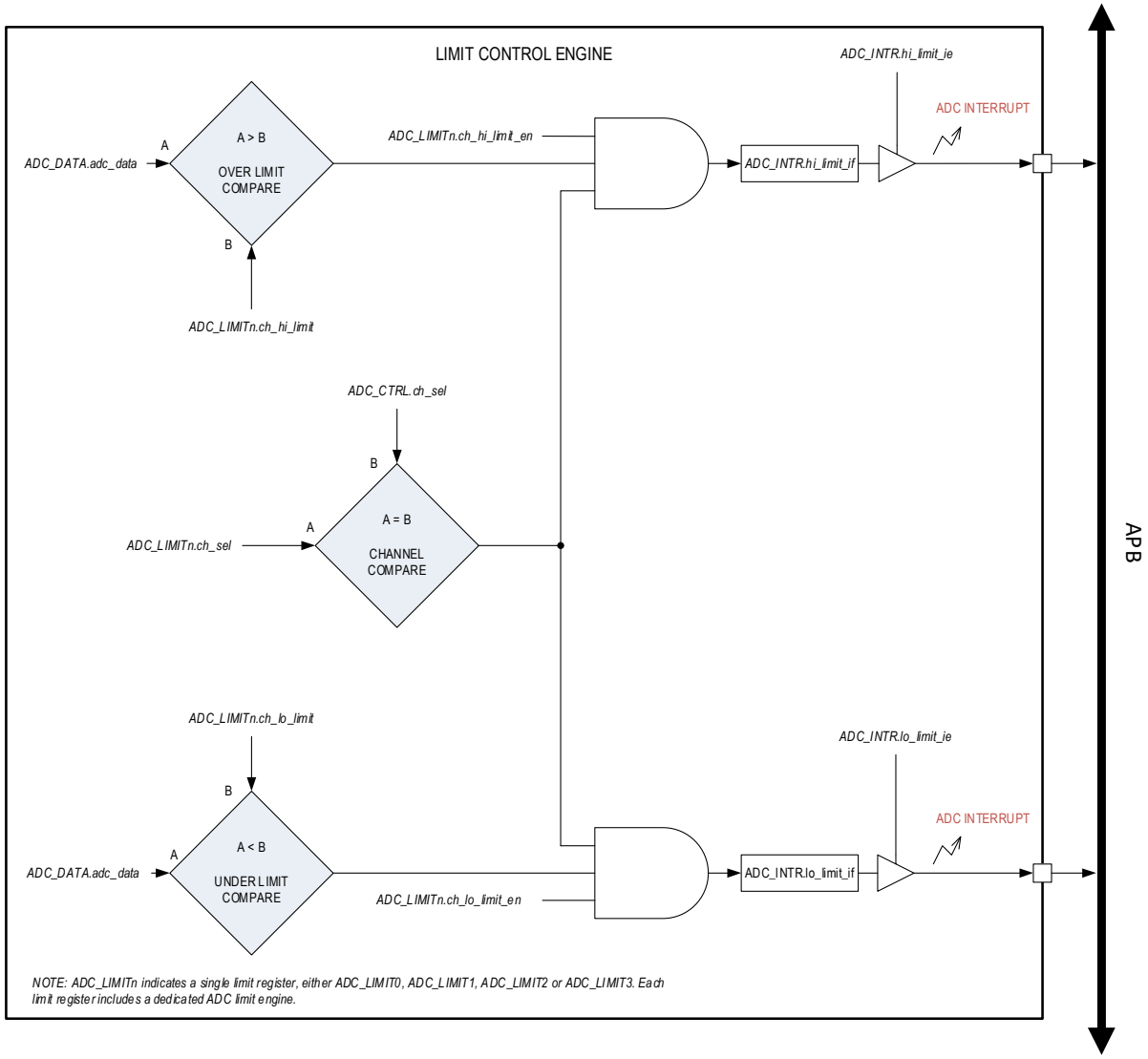
ADC Channel	ADC Input Signal	ADC_CTRL.scale	ADC_CTRL.ref_scale
8	V_{COREA}	0	0
		1	1
9	V_{COREB}	0	0
		1	1
10	V_{RXOUT}	0	0
		1	1
11	V_{TXOUT}	0	0
		1	1
12	V_{DDA}	0	0
		1	1

ADC Channel	ADC Input Signal	*ADC_CTRL.scale*	*ADC_CTRL.ref_scale*
14	$\frac{V_{DDIO}}{4}$	0	0
		1	1
15	$\frac{V_{DDIOH}}{4}$	0	0
		1	1
16	$\frac{V_{REGI}}{4}$	0	0
		1	1

10.8 Data Limits and Out of Range Interrupts

Channel limits are implemented to minimize power consumption for power supply monitoring. The ADC includes four limit registers, *ADC_LIMIT0* to *ADC_LIMIT3*, that are used to set a high limit, low limit, and the ADC channel number to apply the limits against. A block diagram of the limit engine for each of the four limit registers is shown in *Figure 10-2*.

Figure 10-2: ADC Limit Engine



When a measurement is taken on the ADC, the limit engine determines if the channel measured matches one of the channels selected by the limit registers. If it does and the data converted is above or below the high or low limit, an interrupt flag is set resulting in an ADC interrupt if the interrupt is enabled.

Complete the following steps to enable a high and low limit for an ADC input channel using the *ADC_LIMIT0* register. Perform these steps after the ADC is configured for measurement, and the configuration is identical for all four limit registers except for the limit register name:

1. Verify the ADC is not actively taking a measurement by checking *ADC_STATUS.active* until it reads 0.
2. Set *ADC_LIMIT0.ch_sel* field to the selected channel for the high and low limit.
3. Set the high limit, *ADC_LIMIT0.ch_hi_limit*, to the selected 10-bit trip point. When enabled, an ADC measurement greater than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.
4. Set the low limit, *ADC_LIMIT0.ch_lo_limit*, to the selected 10-bit low trip point. When enabled, an ADC measurement lower than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.
5. Enable the high limit, the low limit, or both interrupt signals by writing a 1 to *ADC_LIMIT0.ch_high_limit_en*, *ADC_LIMIT0.ch_low_limit_en*, or both. Note: Each limit register is independently enabled for high- and low-limit interrupts.
6. Clear the ADC interrupt high and low interrupt flags by writing 1 to *ADC_INTR.hi_limit_if* and *ADC_LIMIT0.lo_limit_if*.
7. Enable the high, low, or both interrupts for the ADC by setting *ADC_INTR.hi_limit_if* to 1, *ADC_INTR.lo_limit_ie* to 1, or both.
8. If an ADC conversion occurs above or below the enabled limits, an ADC_IRQ is generated with the *ADC_LIMIT0.adc_high_limit_if*, *ADC_LIMIT0.adc_low_limit_if*, or both set to 1. The *ADC_CTRL.ch_sel* value indicates the channel that caused the interrupt, and the value of the ADC conversion that is out of bounds is in the *ADC_DATA.data* field.

10.9 Power-Down Sequence

Complete the following steps to power-down the ADC:

1. Set *ADC_CTRL.pwr* to 0, disabling the ADC converter power.
2. Set *ADC_CTRL.refbuf_pwr* to 0, disabling the internal reference buffer power.
3. Set *ADC_CTRL.clk_en* to 0, disabling the ADC internal clock.

10.10 Comparator Operation

10.10.1 Comparator 0 Usage

Comparator 0 is controlled individually using the *MCR_CMP0_CTRL* register. Enable comparator 0 by setting the *MCR_CMP0_CTRL.en* field to 1. Comparator 0s output is readable using the *MCR_CMP0_CTRL.out*. Enable interrupt events for comparator 0 by setting the *MCR_CMP0_CTRL.int_en* field to 1. Interrupts for comparator 0 occur when the output changes to its active state. The active state is controlled using the *MCR_CMP0_CTRL.pol* field. When the output state is active, the hardware automatically sets the *MCR_CMP0_CTRL.if* flag to 1. To clear the interrupt flag, write 1 to *MCR_CMP0_CTRL.if*.

10.10.2 Low-Power Comparators 1, 2, and 3 Usage

Comparator 1, 2, and 3 are controlled using the low-power comparator, *LPCOMPn*, registers.

10.10.3 Using Comparator 0 as a Wakeup Source

After configuring comparator 0, configure it as a wakeup source from *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP* by performing the following steps:

1. Enable comparator wakeup events by setting *GCR_PM.aincomp_we* to 1.
2. Enable comparator 0 as a wakeup source by setting *PWRSEQ_LPPWST.comp0* to 1.
3. If desired, provide an IRQ handler for the comparators (LPCOMP_IRQHandler).

After the device exists a low-power mode, determine if the wakeup event was the result of comparator 0 by checking the *PWRSEQ_LPPWST.comp0* and the *MCR_CMP0_CTRL.if*. Wakeup events generated by comparator 0 from *STANDBY* and *BACKUP* mode result in the *PWRSEQ_LPPWST.comp0* bit being set. Write 1 to clear the *PWRSEQ_LPPWST.comp0* bit and the *MCR_CMP0_CTRL.if* bit.

10.10.4 Low-Power Comparators 1, 2, and 3 Wakeup

Wake up from the low-power comparators, *LPCOMPn*, by setting *PWRSEQ_LPPWST.lpcomp* bit to 1. If any of the three low-power comparators (*LPCOMPn*) cause the device to wake up, the specific comparator's interrupt flag is set to 1. Inspection of each comparator's interrupt flag identifies which comparator resulted in the wakeup event. See *LPCOMPn.if* for details.

Enable wakeup events from the low-power comparators by setting the *GCR_PM.aincomp_we* field to 1. If a comparator event occurs and wakes the device from a low-power operating mode, the *PWRSEQ_LPPWST.comp* field is set to 1. Clear the comparator wakeup status flag by writing 1 to *PWRSEQ_LPPWST.comp*.

Note: Comparator 0, if enabled, wakes the device from SLEEP, LPM, UPM, STANDBY, and BACKUP. Comparators 1, 2,and 3, if enabled, wake the device from SLEEP, LPM, and UPM.

10.11 ADC Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 10-5: ADC Registers Summary

Offset	Name	Description
[0x0000]	*ADC_CTRL*	ADC Control Register
[0x0004]	*ADC_STATUS*	ADC Status Register
[0x0008]	*ADC_DATA*	ADC Output Data Register
[0x000C]	*ADC_INTR*	ADC Interrupt Control Register
[0x0010]	*ADC_LIMIT0*	ADC Limit 0 Register
[0x0014]	*ADC_LIMIT1*	ADC Limit 1 Register
[0x0018]	*ADC_LIMIT2*	ADC Limit 2 Register
[0x001C]	*ADC_LIMIT3*	ADC Limit 3 Register

10.11.1 ADC Register Details

Table 10-6: ADC Control Register

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	DNM	0x50	**Reserved, Do Not Modify**

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
20	data_align	R/W	0	**ADC Data Alignment** Selects the alignment of the 16-bit data conversion stored in the DATA register. 0: Data is LSB justified in 16-bit DATA register. DATA[15:10] = 0. 1: Data is MSB justified in 16-bit DATA register. DATA[5:0] = 0.
19	-	RO	0	**Reserved**
18:17	adc_divsel	R/W	0	**External Input Scale** Scales the external inputs, AIN0-AIN7. All eight of external inputs are scaled by the same value 0x0: No scaling. 0x1: Divide by 2. 0x2: Divide by 3. 0x3: Divide by 4.
16:12	ch_sel	R/W	0	**ADC Channel Select** Selects the active channel for the next ADC conversion.
11	clk_en	R/W	0	**ADC Clock Enable** 0: Disabled 1: Enabled
10	-	RO	0	**Reserved**
9	scale	R/W	0	**ADC Input Scale** Scales ADC input by 50 percent. 0: ADC input is not scaled. 1: ADC input is scaled by ½. *Note: See Reference Scaling and Input Scaling for valid settings for each ADC input.*

ch_sel	ADC Channel	Input
0	0	AIN0
1	1	AIN1
2	2	AIN2
3	3	AIN3
4	4	AIN4
5	5	AIN5
6	6	AIN6
7	7	AIN7
8	8	V_{COREA}
9	9	V_{COREB}
10	10	V_{RXOUT}
11	11	V_{TXOUT}
12	12	V_{DDA}
13	13	-
14	14	$\frac{V_{DDIO}}{4}$
15	15	$\frac{V_{DDIOH}}{4}$
16	16	$\frac{V_{REGI}}{4}$
17 - 31	Reserved	Reserved

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
8	ref_scale	R/W	0	**Reference Scale** Scales the internal bandgap reference by 50 percent. 0: Internal bandgap reference is not scaled. 1: Internal bandgap reference is scaled by ½. *Note: See Reference Scaling and Input Scaling for valid settings for each ADC input*
7:5	-	RO	0	**Reserved**
4	ref_sel	R/W	0	**ADC Reference Select** 0: Internal bandgap reference. 1: $\frac{V_{DD}}{2}$ reference.
3	refbuf_pwr	R/W	0	**Reference Buffer Power Enable** 0: Disabled 1: Enabled
2	-	RO	0	**Reserved**
1	pwr	R/W	0	**ADC Power Enable** 0: Disabled 1: Enabled
0	start	R/W	0	**Start ADC Conversion** Write this bit to 1 to start an ADC conversion. When the conversion is complete, the hardware automatically sets this bit to 0 indicating the conversion is complete. 0: ADC inactive or data conversion complete. 1: Starts an ADC conversion or ADC conversion in process.

Table 10-7: ADC Status Register

ADC Status			ADC_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	overflow	RO	0	**ADC Overflow Flag** 0: No overflow on last conversion 1: Overflow on last conversion
2	afe_pwr_up_active	RO	0	**ADC Power-Up State** This field is set to 1 when the ADC charge pump is powering up. 0: AFE is not in power-up delay. 1: AFE is currently in the power-up delay state.
1	-	RO	0	**Reserved**
0	active	RO	0	**ADC Conversion in Progress** 0: ADC is idle. 1: ADC conversion is in progress.

Table 10-8: ADC Data Register

ADC Data			ADC_DATA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	data	RO	0	**ADC Data** This field holds the ADC conversion output data. See section *Conversion* for details.

Table 10-9: ADC Interrupt Control Register

ADC Interrupt Control			ADC_INTR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	-	RO	0	**Reserved**

ADC Interrupt Control			ADC_INTR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
22	pending	RO	0	**ADC Interrupt Pending** 0: No ADC interrupt pending. 1: At least one ADC interrupt is pending, and the corresponding interrupt enable bit is set.
21	-	RO	0	**Reserved**
20	overflow_if	R/W1C	0	**ADC Overflow Interrupt Flag** 1: The last conversion resulted in an overflow
19	lo_limit_if	R/W1C	0	**ADC Low Limit Interrupt Flag** 1: The last conversion resulted in a low-limit condition for one of the limit registers.
18	hi_limit_if	R/W1C	0	**ADC High Limit Interrupt Flag** 1: The last conversion resulted in a high-limit condition for one of the limit registers.
17	ref_ready_if	R/W1C	0	**ADC Reference Ready Interrupt Flag** 0: Not Ready 1: Ready.
16	done_if	R/W1C	0	**ADC Conversion Complete Interrupt Flag** Set by the ADC hardware when an ADC conversion is complete. 1: ADC conversion complete
15:5	-	RO	0	**Reserved**
4	overflow_ie	R/W	0	**ADC Overflow Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.overflow_if*.
3	lo_limit_ie	R/W	0	**ADC Low Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets the *ADC_INTR.lo_limit_if*.
2	hi_limit_ie	R/W	0	**ADC High Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.lo_limit_if*.
1	ref_ready_ie	R/W	0	**ADC Reference Ready Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.ref_ready_if*.
0	done_ie	R/W	0	**ADC Conversion Complete** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.done_if*.

Table 10-10: ADC Limit 0 to 3 Registers

ADC Limit 0			ADC_LIMIT0	[0x0010]
ADC Limit 1			ADC_LIMIT1	[0x0014]
ADC Limit 2			ADC_LIMIT2	[0x0018]
ADC Limit 3			ADC_LIMIT3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0	**Reserved**
29	ch_hi_limit_en	R/W	0	**High Limit Monitoring Enable** If set, then an ADC conversion that results in a value greater than the *ch_high_limit* field generates an ADC interrupt if the ADC high-limit interrupt is enabled. (*ADC_INTR.hi_limit_ie* = 1). 1: The high-limit comparison for the ch_sel channel is active. 0: The high-limit comparison is not enabled.

ADC Limit 0	ADC_LIMIT0	[0x0010]
ADC Limit 1	ADC_LIMIT1	[0x0014]
ADC Limit 2	ADC_LIMIT2	[0x0018]
ADC Limit 3	ADC_LIMIT3	[0x001C]

Bits	Field	Access	Reset	Description
28	ch_lo_limit_en	R/W	0	**Low Limit Monitoring Enable** If set, then an ADC conversion that results in a value less than the ch_high_limit field generates an ADC interrupt if the ADC low-limit interrupt is enabled (*ADC_INTR*.*lo_limit_ie* = 1). 1: The low-limit comparison for the ch_sel channel is active. 0: The low-limit comparison is not enabled.
27:24	ch_sel	R/W	0	**ADC Channel for Limit Monitoring** Sets the ADC input channel for high- and low-limit thresholds. See *ADC_CTRL*.*ch_sel* for valid values for this field.
23:22	-	RO	0	**Reserved**
21:12	ch_hi_limit	R/W	0x3FF	**High Limit Threshold** Sets the threshold for high-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the *ch_sel* field. ADC conversions greater than this field are over threshold and can result in interrupt assertion if the *ch_hi_limit_en* field is set. Valid values for this field are 0x000 to 0x3FF.
11:10	-	RO	0	**Reserved**
9:0	ch_lo_limit	R/W	0	**Low Limit Threshold** Sets the threshold for low-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the *ch_sel* field. ADC conversions less than this field are under threshold and can result in interrupt assertion if the *ch_lo_limit_en* field is set. Valid values for this field are 0x000 to 0x3FF.

10.12 Low-Power Comparator Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 10-11: Low-Power Comparator Registers Summary

Offset	Name	Description
[0x0000]	*LPCOMP1*	Low-Power Comparator 1 Register
[0x0004]	*LPCOMP2*	Low-Power Comparator 2 Register
[0x0008]	*LPCOMP3*	Low-Power Comparator 3 Register

10.12.1 Low-Power Comparator Register Details

Table 10-12: Low-Power Comparator n Registers

Low-Power Comparator 1	**LPCOMP1**	**[0x0000]**
Low-Power Comparator 2	**LPCOMP2**	**[0x0004]**
Low-Power Comparator 3	**LPCOMP3**	**[0x0008]**

Bits	Field	Access	Reset	Description
31:16	-	RO	0	**Reserved**
15	if	R/W1C	0	**Low-Power Comparator n Interrupt Flag** This field is set to 1 by hardware when the comparator output changes to the active state as set using the *pol* field. Write 1 to clear this flag. 0: No interrupt 1: Interrupt occurred
14	out	RO	*	**Low-Power Comparator n Output** This field is the comparator's output state. 0: Output low 1: Output high
13:7	-	RO	0	**Reserved**
6	int_en	R/W	0	**Low-Power Comparator n Interrupt Enable** Set this field to 1 to enable the IRQ for specific low-power comparator. 0: Interrupt disabled 1: Interrupt enabled
5	pol	R/W	0	**Comparator n Interrupt Polarity Select** Set this field to select the polarity of the output change that generates a low-power comparator interrupt. 0: Interrupt occurs from a transition from low to high 1: Interrupt occurs from a transition from high to low
4:1	-	RO	0	**Reserved**
0	en	R/W	0	**Low-Power Comparator n Enable** Set this field to 1 to enable the comparator 0: Comparator disabled 1: Comparator enable

11. UART (UART)

The universal asynchronous receiver/transmitter (UART) and the low-power universal asynchronous receiver/transmitter (LPUART) interfaces communicate with external devices using industry standard serial communications protocols. The UARTs are full-duplex serial ports. Each UART instance is independently configurable unless using a shared external clock source.

The LPUART is a special version of the peripheral that can receive characters at up to 9600 baud while in low-power modes. The hardware loads valid received characters into the receive FIFO and wakes the device when an enabled interrupt condition occurs.

The peripheral provides the following features:

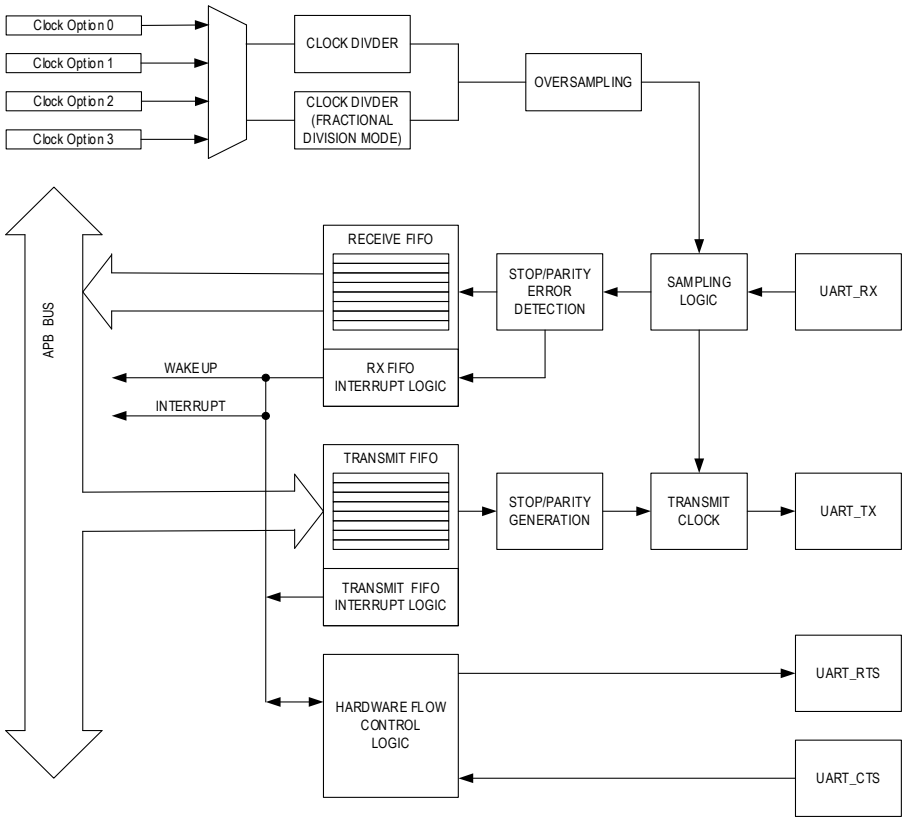
- Flexible baud rate generation up to 12.5Mbps for UART
- Programmable character size of 5-bits to 8-bits
- Stop bit settings of 1, 1.5, or 2-bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic frame error detection
- Separate 8-byte transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control (HFC) using ready-to-send (RTS) and clear-to-send (CTS) pins
- Separate DMA channels for transmit and receive
  - DMA support is available in *ACTIVE* and *SLEEP*

The LPUART instance provides these additional features:

- Baud rate support for up to 1.85Mbps in *ACTIVE*
- Receive characters in *SLEEP*, *LPM*, and *UPM* at up to 9600 baud
- Fractional baud rate divisor improves baud rate accuracy for 9600 and lower baud rates
- Wakeup from low-power modes to *ACTIVE* on multiple receive FIFO conditions

Figure 11-1 shows a high-level diagram of the UART peripheral.

Figure 11-1: UART Block Diagram



Note: See Table 11-1 for the clock options supported by each UART instance.

11.1 Instances

Instances of the peripheral are shown in *Table 11-1*. The standard UARTs and the LPUARTs are functionally similar; for common functionality they are referred to as UART. The LPUART instance supports fractional division mode (FDM) and is referenced as LPUART for feature specific options.

Table 11-1: MAX32655 UART/LPUART Instances

Instance	Register Access Name	LPUART	Power Modes	Clock Option				HFC	Transmit FIFO Depth	Receive FIFO Depth
				0	1	2	3			
UART0	UART0	No	*ACTIVE SLEEP*	PCLK	-	IBRO	-	Yes	8	8
UART1	UART1									
UART2	UART2									
LPUART0	UART3	Yes	*ACTIVE SLEEP LPM UPM*	-	-	IBRO	ERTCO	No		

11.2 DMA

Each UART instance supports DMA for both transmit and receive; separate DMA channels can be connected to the receive and transmit FIFOs.

The UART DMA channels are configured using the UART DMA configuration register, *UARTn_DMA*. Enable the receive FIFO DMA channel by setting *UARTn_DMA.rx_en* to 1 and enable the transmit FIFO DMA channel by setting *UARTn_DMA.tx_en* to 1. DMA transfers are automatically triggered by the hardware based on the number of bytes in the receive FIFO and transmit FIFO.

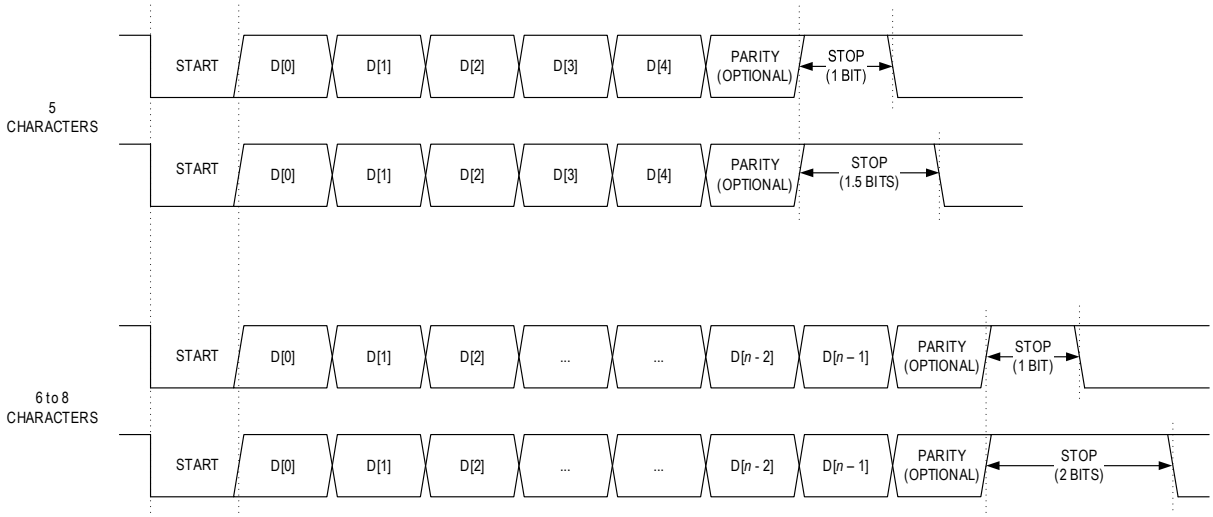
When DMA is enabled, the following describes the behavior of the DMA requests:

- A receive DMA request is asserted when the number of bytes in the receive FIFO transitions to be greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of bytes in the transmit FIFO transitions to be less than the transmit FIFO threshold.

11.3 UART Frame

Figure 11-2 shows the UART frame structure. Character sizes of 5 to 8 bits are configurable through the *UARTn_CTRL.char_size* field. Stop bits are configurable as 1 or 1.5 bits for 5-character frames and 1 or 2 stop bits for 6, 7, or 8-character frames. Parity support includes even, odd, mark, space, and none.

Figure 11-2: UART Frame Structure



11.4 FIFOs

Separate receive and transmit FIFOs are provided. They are both accessed through the same *UARTn_FIFO.data* field. The current level of the TX FIFO is read from *UARTn_STATUS.tx_lvl*, and the current level of the RX FIFO is read from *UARTn_STATUS.rx_lvl*. Data for character sizes less than 7 bits are right justified.

11.4.1 TX FIFO Operation

Writing data to *UARTn_DMA.data* field increments the TX FIFO pointer, *UARTn_STATUS.tx_lvl*, and loads the data into the TX FIFO. The *UARTn_TXPEEK.data* register provides a feature that allows software to "peek" at the current value of the write-only TX FIFO without changing the *UARTn_STATUS.tx_lvl*. Writes to the TX FIFO are ignored while *UARTn_STATUS.tx_lvl* = C_TX_FIFO_DEPTH.

11.4.2 RX FIFO Operation

Reads of the *UARTn_FIFO.data* field return the character values in the RX FIFO and decrement the *UARTn_STATUS.rx_lvl*. An overrun event occurs if a valid frame, including parity, is detected while *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH. When an overrun event occurs the data is discarded by hardware.

A parity error event indicates that the value read from *UARTn_.data* contains a parity error.

11.4.3 Flushing

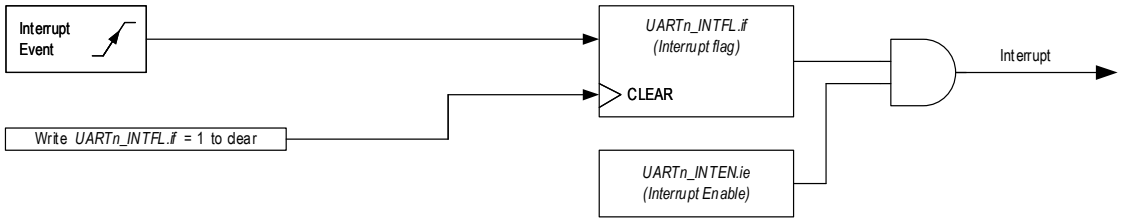
The FIFOs are flushed on the following conditions:

- Setting the *UARTn_CTRL.rx_flush* field to 1 flushes the RX FIFO by setting its pointer to 0.
- Setting the *UARTn_CTRL.tx_flush* field to 1 flushes the TX FIFO by setting its pointer to 0.
- Flush the FIFOs by setting the *GCR_RST0* respective uart's reset field to 1.

11.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 11-2*. Unless noted otherwise, each instance has its own set of interrupts, and higher-level flag and enable fields as shown in *Table 11-2*

Figure 11-3: UART Interrupt Functional Diagram



Some activity may cause more than one event; setting one or more event flags. An event interrupt occurs if the corresponding interrupt enable is set. The interrupt flags, when set, must be cleared by the software by writing 1 to the corresponding interrupt flag field.

Table 11-2: MAX32655 Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Frame Error	*UARTn_INT_FL.rx_ferr*	*UARTn_INT_EN.rx_ferr*
Parity Error	*UARTn_INT_FL.rx_par*	*UARTn_INT_EN.rx_par*
CTS Signal Change	*UARTn_INT_FL.cts_ev*	*UARTn_INT_EN.cts_ev*
Receive FIFO Overrun	*UARTn_INT_FL.rx_ov*	*UARTn_INT_EN.rx_ov*
Receive FIFO Threshold	*UARTn_INT_FL.rx_thd*	*UARTn_INT_EN.rx_thd*
Transmit FIFO Half-Empty	*UARTn_INT_FL.tx_he*	*UARTn_INT_EN.tx_he*

11.5.1 Frame Error

A frame error is generated when the UART sampling circuitry detects an invalid bit. Each bit is sampled three times, as shown in *Figure 11-4*, and can generate a frame error on the start bit, stop bit, data bits, and optionally the parity bit. When a frame error occurs, the data is discarded.

The frame error criteria are different based on the following:

- Standard UART and LPUART with FDM disabled
  - The start bit is sampled 3 times and all samples must be 0 or a frame error is generated.
  - Each data bit is sampled and 2 of the 3 samples must match or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times and all samples must match or a frame error is generated.
  - The stop bit is sampled 3 times and all samples must be 1 or a frame error is generated.
  - See *Table 11-3* for details
- LPUART with FDM enabled (*UARTn_CTRL.fdm* = 1) and data/parity edge detect enabled (*UARTn_CTRL.dpfe_en* = 1).
  - The start bit is sampled 3 times and all samples must be 0 or a frame error is generated.
  - Each data bit is sampled 3 times and all samples must match or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times and all samples must match or a frame error is generated.
  - The stop bit is sampled 3 times and all samples must be 1 or a frame error is generated.
  - See *Table 11-4* for details.

Table 11-3: Frame Error Detection for Standard UARTs and LPUART

UARTn_CTRL .par_en	UARTn_CTRL .par_md	UARTn_CTRL .par_eo	Start Samples	Data Samples	Parity Samples	Stop Samples
0	N/A	N/A	3 of 3 must be 0	2/3 must match	Not Present	3 of 3 must be 1
1	0	0			3/3 = 1 if even number "1" 3/3 = 0 if odd number "0"	
	0	1			3/3 = 1 if odd number "1" 3/3 = 0 if even number "0"	
	1	0			3/3 = 1 if even number "0" 3/3 = 0 if odd number "1"	
	1	1			3/3 = 1 if odd number "0" 3/3 = 0 if even number "1"	

Table 11-4: Frame Error Detection for LPUARTs with UARTn_CTRL.fdm = 1 and UARTn_CTRL.dpfe_en = 1

UARTn_CTRL .par_en	UARTn_CTRL .par_md	UARTn_CTRL .par_eo	Start Samples	Data Samples	Parity Samples	Stop Samples
0	N/A	N/A	3 of 3 must be 0	3 of 3 must match	Not Present	3 of 3 must be 1
1	0	0			3 of 3 = 1 if even number of 1s 3 of 3 = 0 if odd number 0s	
	0	1			3 of 3 = 1 if odd number 1s 3 of 3 = 0 if even number 0s	
	1	0			3 of 3 = 1 if even number 0s 3 of 3 = 0 if odd number 1s	
	1	1			3 of 3 = 1 if odd number 0s 3 of 3 = 0 if even number 1s	

11.5.2 Parity Error

Set *UARTn_CTRL.par_en* = 0 to enable parity checking of the received frame. If the calculated parity does not match the parity bit, then the corresponding interrupt flag is set. The data received is saved to the receive FIFO when a parity error occurs.

11.5.3 CTS Signal Change

A CTS signal change condition occurs if HFC is enabled, the UART baud clock is enabled, and the CTS pin changes state.

11.5.4 Overrun

An overrun condition occurs if a valid frame is received when the receive FIFO is full. The interrupt flag is set at the end of the stop bit and the frame is discarded.

11.5.5 Receive FIFO Threshold

A receive FIFO threshold event occurs when a valid frame is received that causes the number of bytes to exceed the configured receive FIFO threshold *UARTn_CTRL.rx_thd_val*.

11.5.6 Transmit FIFO Half-Empty

The transmit FIFO half-empty event occurs when *UARTn_STATUS.tx_lvl* transitions from more than half-full to half-empty as shown in *Equation 11-1*.

Note: When this condition occurs, verify the number of bytes in the transmit FIFO (UARTn_STATUS.tx_lvl) prior to refilling.

Equation 11-1: UART Transmit FIFO Half-Empty Condition

$$\left(\frac{C\_TX\_FIFO\_DEPTH}{2}+1\right) \xrightarrow{Transistions\ from} \left(\frac{C\_TX\_FIFO\_DEPTH}{2}\right)$$

11.6 LPUART Wakeup Events

LPUART instances can receive characters while in the low-power modes listed in *Table 11-1*. If enabled, each of the receive FIFO conditions shown in *Table 11-5* wakes the device, exits the low-power mode, and returns the device to *ACTIVE*.

Unlike interrupts, wakeup activity is based on a condition, not an event. As long as the condition is true and the wakeup enable field is set to 1, the wakeup flag remains set.

Table 11-5: MAX32655 Wakeup Events

Receive FIFO Condition	Wakeup Flag *UARTn_WKFL*	Wakeup Enable *UARTn_WKEN*	Low-Power Peripheral Wakeup Flag	Low-Power Peripheral Wakeup Enable	Low-Power Clock Disable
Threshold	*rx_thd*	*rx_thd*	*PWRSEQ_LPPWST.uart3*	*PWRSEQ_LPPWEN.uart3*	*LPGCR_PCLKDIS.uart3*
Full	*rx_full*	*rx_full*			
Not Empty	*rx_ne*	*rx_ne*			

11.6.1 RX FIFO Threshold

This condition persists while *UARTn_STATUS.rx_lvl* ≥ *UARTn_CTRL.rx_thd_val*.

11.6.2 RX FIFO Full

This condition persists while *UARTn_STATUS.rx_lvl* ≥ C_RX_FIFO_DEPTH.

11.6.3 RX Not Empty

This condition persists while *UARTn_STATUS.rx_lvl* > 0.

11.7 Inactive State

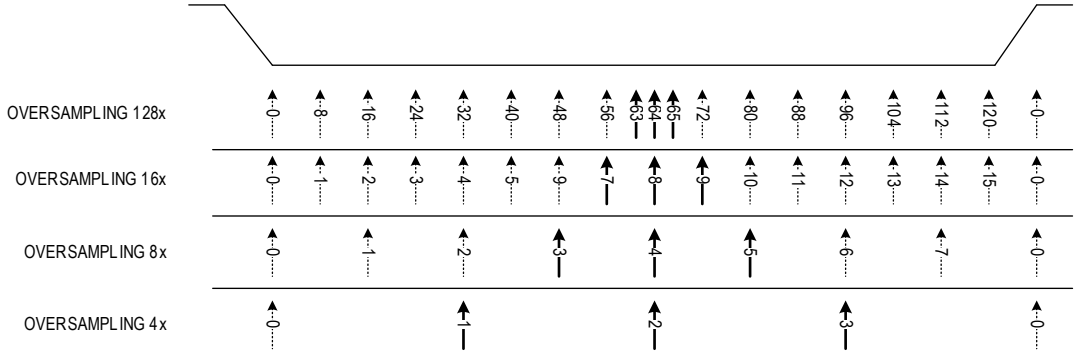
The following conditions result in the UART being inactive:

- When *UARTn_CTRL.bclken* = 0
- After setting *UARTn_CTRL.bclken* to 1 until *UARTn_CTRL.bclkrdy* = 1
- Any write to the *UARTn_CLKDIV.clkdiv field* while *UARTn_CTRL.bclken* = 1
- Any write to the *UARTn_OSR.osr* field when *UARTn_CTRL.bclken* = 1

11.8 Receive Sampling

Each bit of a frame is oversampled to improve noise immunity. The oversampling rate (OSR) is configurable with the *UARTn_OSR.osr* field. In most cases, the bit is evaluated based on three samples at the midpoint of each bit time as shown in *Figure 11-4*.

Figure 11-4: Oversampling Example



Whenever *UARTn_CLKDIV.clkdiv* < 0x10 (i.e., division rate less than 8.0), OSR is not used and the over sampling rate is adjusted to full sampling by the hardware. In full sampling, the receive input is sampled on every clock cycle regardless of the OSR setting.

Note: For 9600 baud low-power operation, the dual-edge sampling mode must be enabled (*UARTn_CTRL.desm* = 1).

11.9 Baud Rate Generation

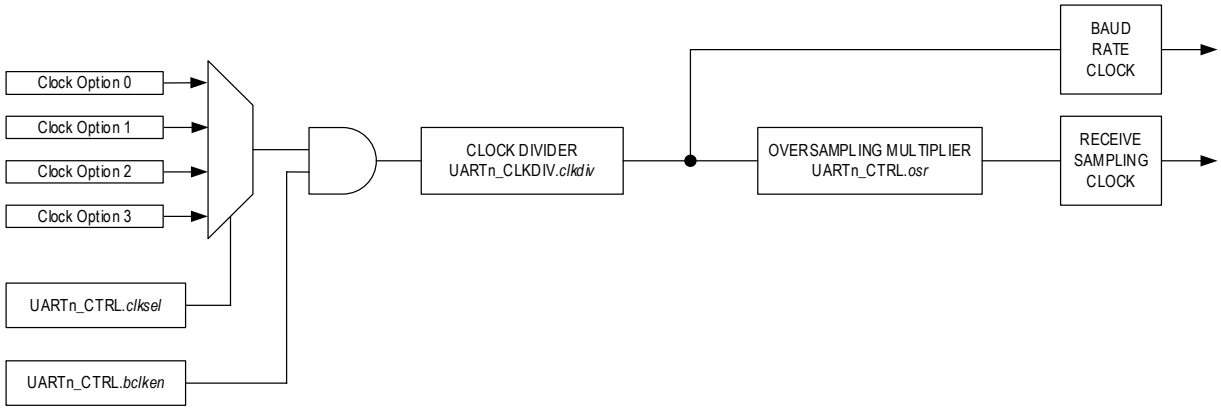
The baud rate is determined by the selected UART clock source and value of the clock divisor. Multiple clock sources are available for each UART instance. See *Table 11-1* for available clock sources.

Note: Changing the clock source should only be done between data transfers to avoid corrupting an ongoing data transfer.

11.9.1 UART Clock Sources

Standard UART instances operate only in *ACTIVE* and *SLEEP*. Standard UART instances can only wake the device from *SLEEP*. *Figure 11-5* shows the baud rate generation path for standard UARTs.

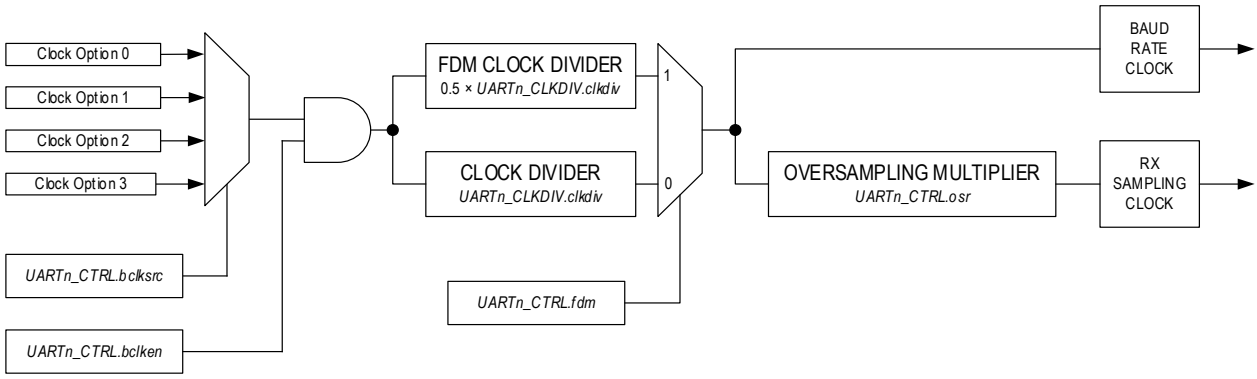
Figure 11-5: UART Baud Rate Generation



11.9.2 LPUART Clock Sources

LPUART instances support FDM and are configurable for operation at 9600 and lower baud rates for operation in *SLEEP*, *LPM*, and *UPM*. Operation in *LPM* and *UPM* require the use of the *ERTCO* as the baud rate clock source. The *ERTCO* can be configured to remain active in *LPM* and *UPM*, allowing the LPUART to receive data and serve as a wakeup source, while power consumption is at a minimum.

Figure 11-6: LPUART Timing Generation



11.9.3 Baud Rate Calculation

The transmit and receive circuits share a common baud rate clock, which is the selected UART clock source divided by the clock divisor. Instances that support FDM offer a 0.5 fractional clock division when enabled by setting *UARTn_CTRL.fdm* = 1. This allows for greater accuracy when operating at low baud rates, as well as finer granularity for the oversampling rate.

Use the following formula to calculate the *UARTn_CLKDIV.clkdiv* value based on the clock source, and desired baud rate, and integer or fractional divisor.

Equation 11-2: UART Clock Divisor Formula

$$UARTn\_CTRL.fdm = 0:$$

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{UART\ Clock}{Baud\ Rate}\right]$$

Equation 11-3: LPUART Clock Divisor Formula for UARTn_CTRL.fdm = 1

$$UARTn\_CTRL.fdm = 1:$$

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{UART\ Clock}{Baud\ Rate} \times 2\right]$$

For example, in a case where the UART clock is 50MHz and target baud rate is 115,200 bps:

- When *UARTn_CTRL.fdm* = 0, $UARTn\_CLKDIV.clkdiv = \left(\frac{50{,}000000}{115{,}200}\right) = 434$
- When *UARTn_CTRL.fdm* = 1, $UARTn\_CLKDIV.clkdiv = \left(\frac{50{,}000{,}000}{115{,}200}\right) \times 2 = 434.03 \times 2 = 868$

11.9.4 Low-Power Mode Operation of LPUARTs for 9600 Baud and Below

LPUART instances have the option to configure the receiver for 9600 and lower baud rates and enable the LPUART in the low- power modes *SLEEP*, *LPM*, and *UPM*. Receipt of a valid frame loads the receive FIFO and increments *UARTn_STATUS.rx_lvl*. If a wakeup event, shown in *Table 11-5*, is enabled, the device exits the current low-power mode and

returns to *ACTIVE*. See *Baud Rate Calculation* and *Equation 11-3* for details on setting the baud rate for LPUART instances with *UARTn_CTRL.fdm* set to 1.

Table 11-6: LPUART Low Baud Rate Generation Examples (UARTn_CTRL.fdm = 1)

Clock Source	BAUD (bits/s)	Ratio (Clock/BAUD)	Calculated *UARTn_CLKDIV.clkdiv*	Error	*UARTn_OSR.osr*
ERTCO	9,600	3.413	7	-2.5%	N/A (1×)
	7,200	4.551	9	+1.1%	N/A (1×)
	4,800	6.827	14	-2.5%	N/A (1×)
	2,400	13.653	27	+1.1%	0: 8× 1: 12×
	1,800	18.204	36	+1.1%	0: 8× 1: 12× 2: 16×
	1,200	27.307	54	+1.1%	0: 8× 1: 12× 2: 16× 3: 20× 4: 24×

11.9.4.1 Configurating a LPUART for Low-Power Modes of Operation

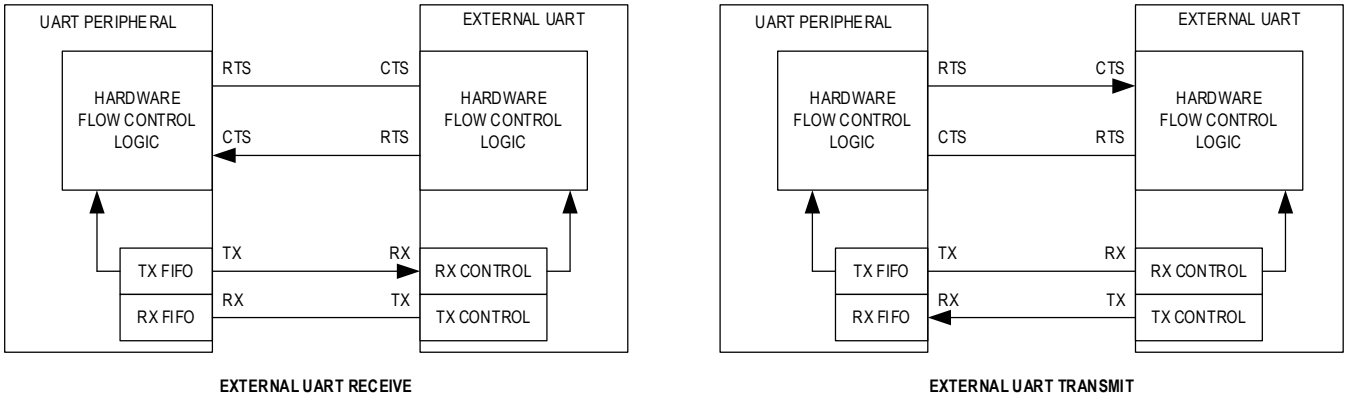
Use the following procedure to receive characters at 9600 or lower baud rates while in low-power modes:

1. Clear *UARTn_CTRL.bclken* = 0 to disable the baud clock. The hardware immediately clears *UARTn_CTRL.bclkrdy* to 0.
2. Set *PWRSEQ_LPCN.x32ken*= 1 to ensure the 32kHz clock source remains active in *LPM* and *UPM* modes.
3. Ensure *UARTn_CTRL.ucagm* = 1.
4. Configure *UARTn_CTRL.bclksrc* to select the *ERTCO*.
5. Set *UARTn_CTRL.fdm* to 1 to enable FDM.
6. Set *UARTn_CLKDIV.clkdiv* to the calculated clock divisor shown in *Table 11-6* for the required baud rate.
7. Set *UARTn_CTRL.desm* to 1 to enable receive dual edge sampling mode.
8. Choose the desired wakeup conditions from *Table 11-5*.
   a. Clear any of the wakeup conditions chosen if currently active in the *UARTn_WKFL* register.
   b. Enable the wakeup condition; set the wakeup field to 1 in the *UARTn_WKEN* register.
9. Set the *UARTn_CTRL.bclken* field to 1 to enable the baud clock.
10. Poll the *UARTn_CTRL.bclkrdy* field until it reads 1.
11. Enter the desired low-power mode.

11.10 Hardware Flow Control

The optional HFC uses two additional pins, CTS and RTS, as a handshaking protocol to manage UART communications. For full duplex operation, the RTS output pin on the peripheral is connected to the CTS input pin on the external UART and the CTS input pin on the peripheral is connected to the RTS output pin on the external UART as shown in *Figure 11-7*.

Figure 11-7: HFC Physical Connection



In HFC operation, a UART transmitter waits for the external device to assert its CTS pin. When CTS is asserted, the UART transmitter sends data to the external device. The external device keeps CTS asserted until it is unable to receive additional data, typically because the external device's receive FIFO is full. The external device then deasserts CTS until the device is able to receive more data. The external device then asserts CTS again allowing additional data to be sent.

HFC can be fully automated by the peripheral hardware or by software through direct monitoring of the CTS input signal and control of the RTS output signal.

11.10.1 Automated HFC

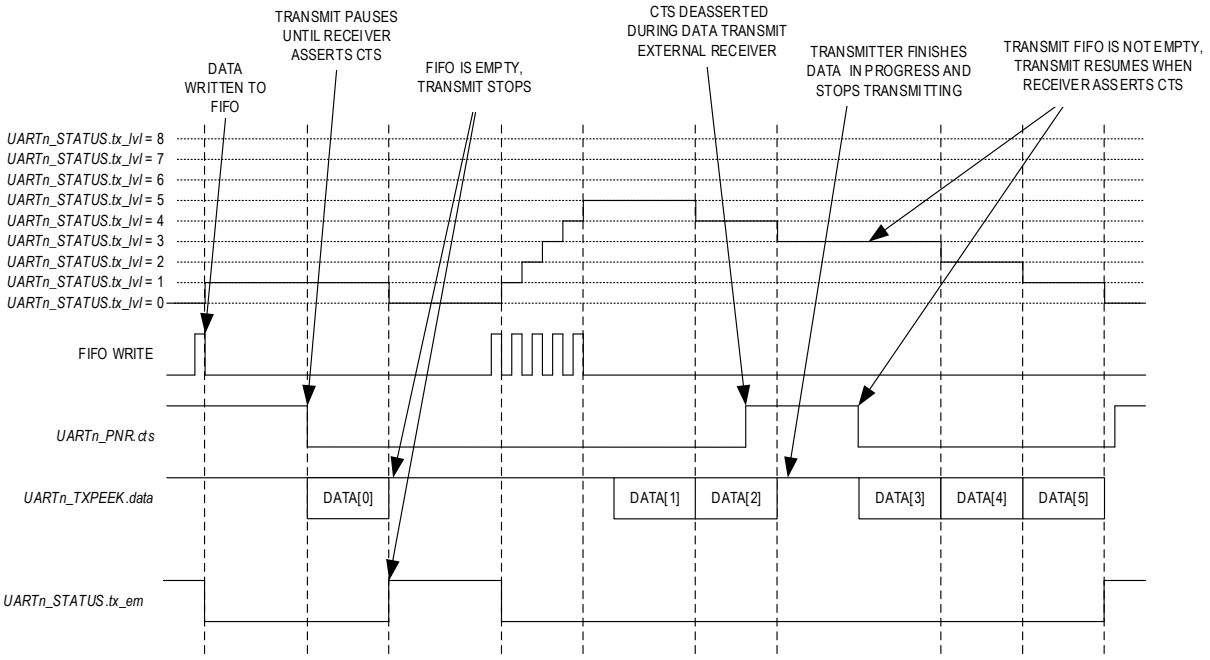
Setting *UARTn_CTRL*.*hfc_en* = 1 enables automated HFC. When automated HFC is enabled, the hardware manages the CTS and RTS signals. The deassertion of the RTS signal is configurable using the *UARTn_CTRL*.*rtsdc* field:

- *UARTn_CTRL*.*rtsdc* = 0: Deassert RTS when *UARTn_STATUS*.*rx_lvl* = C_RX_FIFO_DEPTH
- *UARTn_CTRL*.*rtsdc* = 1: Deassert RTS while *UARTn_STATUS*.*rx_lvl* ≥= *UARTn_CTRL*.*rx_thd_val*

The transmitter continues to send data as long as the CTS signal is asserted and there is data in the transmit FIFO. If the receiver deasserts the CTS pin, the transmitter finishes transmission of the current character and then waits until the CTS pin state is asserted before continuing transmission. *Figure 11-8* shows the state of the CTS pin during a transmission under automated HFC.

Automated HFC does not generate interrupt events related to the state of the transmit FIFO or the receive FIFO. FIFO management must be handled by the software. See *Interrupt Events* for additional information.

Figure 11-8: HFC Signaling for Transmitting to an External Receiver



11.10.2 Application Controlled HFC

Application controlled HFC requires the software to manually control the RTS output pin and monitor the CTS input pin. Using application controlled HFC requires the automated HFC to be disabled by setting the *UARTn_CTRL.hfc_en* field to 1. Additionally, the software should enable CTS sampling (*UARTn_CTRL.cts_dis* = 0) if performing application controlled HFC.

11.10.2.1 RTC/CTS Handling for Application Controlled HFC

The software can manually monitor the CTS pin state by reading the field *UARTn_PNR.cts*. The software can manually set the state of the RTS output pin and read the current state of the RTS output pin using the field *UARTn_PNR.rts*. The software must manage the state of the RTS pin when performing application controlled HFC.

Interrupt support for CTS input signal change events is supported even when automated HFC is disabled. Software can enable the CTS interrupt event by setting the *UARTn_INT_EN.cts_ev* field to 1. The CTS signal change interrupt flag is set by the hardware any time the CTS pin state changes. The software must clear this interrupt flag manually by writing 1 to the *UARTn_INT_FL.cts_ev* field.

Note: CTS pin state monitoring is disabled any time the UART baud clock is disabled (UARTn_CTRL.bclken = 0). The software must enable CTS pin monitoring by setting the field UARTn_CTRL.cts_dis to 0 after enabling the baud clock if CTS pin state monitoring is required.

11.11 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 11-7*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

All registers and fields apply to both UART and LPUART instances unless specified otherwise.

Table 11-7: UART/LPUART Register Summary

Offset	Register	Name
[0x0000]	*UARTn_CTRL*	UART Control Register
[0x0004]	*UARTn_STATUS*	UART Status Register
[0x0008]	*UARTn_INT_EN*	UART Interrupt Enable Register
[0x000C]	*UARTn_INT_FL*	UART Interrupt Flag Register
[0x0010]	*UARTn_CLKDIV*	UART Clock Divisor Register
[0x0014]	*UARTn_OSR*	UART Oversampling Control Register
[0x0018]	*UARTn_TXPEEK*	UART Transmit FIFO
[0x001C]	*UARTn_PNR*	UART Pin Control Register
[0x0020]	*UARTn_FIFO*	UART FIFO Data Register
[0x0030]	*UARTn_DMA*	UART DMA Control Register
[0x0034]	*UARTn_WKEN*	UART Wakeup Interrupt Enable Register
[0x0038]	*UARTn_WKFL*	UART Wakeup Interrupt Flag Register

11.11.1 Register Details

Table 11-8: UART Control Register

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:23	-	DNM	0	**Reserved**	
22	desm	R/W	0	**Receive Dual Edge Sampling Mode** LPUART instances only. This field is reserved in standard UART instances. 0: Sample receive input signal on clock rising edge only. 1: Sample receive input signal on both rising and falling edges.	
21	fdm	R/W	0	**Fractional Division Mode** LPUART instances only. This field is reserved in standard UART instances. 0: Baud rate divisor is an integer. 1: Baud rate divisor supports 0.5 division resolution.	
20	ucagm	R/W	0	**UART Clock Auto Gating Mode** *Note: Software must set this field to 1 for proper operation.* 0: No gating. 1: UART clock is paused during transmit and receive idle states.	
19	bclkrdy	R	0	**Baud Clock Ready** 0: Baud clock not ready. 1: Baud clock ready.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
18	dpfe_en	R/W	0	**Data/Parity Bit Frame Error Detection Enable** LPUART instances only. This field is reserved in standard UART instances. 0: Disable. Do not detect frame errors on receive between start bit and stop bit. 1: Enable. Detect frame errors when receive changes at the center of bit time.	
17:16	bclksrc	R/W	0	**Baud Clock Source** Selects the baud clock source. See *Table 11-1* for available clock options for each UART instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.	
15	bclken	R/W	0	**Baud Clock Enable** 0: Disabled 1: Enabled	
14	rtsdc	R	0	**HFC RTS Deassert Condition** 0: Deassert RTS when the receive FIFO Level = C_RX_FIFO_DEPTH (FIFO full). 1: Deassert RTS while the receive FIFO Level >= *UARTn_CTRL.rx_thd_val*.	
13	hfc_en	R/W	0	**HFC Enable** 0: Disabled 1: Enabled	
12	stopbits	R/W	0	**Number of Stop Bits** 0: 1 stop bit. 1: 1.5 stop bits (for 5 bit mode) or 2 stop bits (for 6/7/8 bit mode).	
11:10	char_size	R/W	0	**Character Length** 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits	
9	rx_flush	W1	0	**Receive FIFO Flush** Write 1 to flush the receive FIFO. This bit will always read 0. 0: N/A 1: Flush FIFO	
8	tx_flush	W1	0	**Transmit FIFO Flush** Write 1 to flush the transmit FIFO. This bit will always read 0. 0: N/A 1: Flush FIFO	
7	cts_dis	R/W	1	**CTS Sampling Disable** 0: Enabled 1: Disabled	
6	par_md	R/W	0	**Parity Value Select** 0: Parity calculation is based on 1 bits (mark). 1: Parity calculation is based on 0 bits (space).	
5	par_eo	R/W	0	**Parity Odd/Even Select** 0: Even parity. 1: Odd parity.	
4	par_en	R/W	0	**Transmit Parity Generation Enable** 0: Parity transmission disabled. 1: Parity bit is calculated and transmitted after the last character bit.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
3:0	rx_thd_val	R/W	0	**Receive FIFO Threshold** Valid settings are from 1 to (C_RX_FIFO_DEPTH – 1). 0: Reserved 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9 - 15: Reserved	

Table 11-9: UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	
15:12	tx_lvl	RO	0	**Transmit FIFO Level** Number of characters in the transmit FIFO. 0 - 8: Number of bytes in the transmit FIFO. 9 - 15: Reserved	
11:8	rx_lvl	RO	0	**Receive FIFO Level** Number of characters in the Receive FIFO. 0 - 8: Number of bytes in the receive FIFO 9 - 15: Reserved	
7	tx_full	RO	0	**Transmit FIFO Full** 0: Not full 1: Full	
6	tx_em	RO	1	**Transmit FIFO Empty** 0: Not empty 1: Empty	
5	rx_full	RO	0	**Receive FIFO Full** 0: Not full 1: Full	
4	rx_em	RO	1	**Receive FIFO Empty** 0: Not empty 1: Empty	
3:2	-	RO	0	**Reserved**	
1	rx_busy	RO	0	**Receive Busy** 0: UART is not receiving a character. 1: UART is receiving a character.	
0	tx_busy	RO	0	**Transmit Busy** 0: UART is not transmitting data. 1: UART is transmitting data.	

Table 11-10: UART Interrupt Enable Register

UART Interrupt Enable Register			UARTn_INT_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	tx_he	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** 0: Disabled 1: Enabled
5	-	RO	0	**Reserved**
4	rx_thd	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** 0: Disabled 1: Enabled
3	rx_ov	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** 0: Disabled 1: Enabled
2	cts_ev	R/W	0	**CTS Signal Change Event Interrupt Enable** 0: Disabled 1: Enabled
1	rx_par	R/W	0	**Receive Parity Event Interrupt Enable** 0: Disabled 1: Enabled
0	rx_ferr	R/W	0	**Receive Frame Error Event Interrupt Enable** 0: Disabled 1: Enabled

Table 11-11: UART Interrupt Flag Register

UART Interrupt Flag			UARTn_INT_FL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	tx_he	R/W1C	0	**Transmit FIFO Half-Empty Interrupt Flag** 0: Disabled 1: Enabled
5	-	RO	0	**Reserved**
4	rx_thd	R/W1C	0	**Receive FIFO Threshold Interrupt Flag** 0: Disabled 1: Enabled
3	rx_ov	R/W1C	0	**Receive FIFO Overrun Interrupt Flag** 0: Disabled 1: Enabled
2	cts_ev	R/W1C	0	**CTS Signal Change Interrupt Flag** 0: Disabled 1: Enabled
1	rx_par	R/W1C	0	**Receive Parity Error Interrupt Flag** 0: Disabled 1: Enabled
0	rx_ferr	R/W1C	0	**Receive Frame Error Interrupt Flag** 0: Disabled 1: Enabled

Table 11-12: UART Clock Divisor Register

UART Clock Divisor			UARTn_CLKDIV	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	clkdiv	R/W	0	**Baud Rate Divisor** This field sets the divisor used to generate the baud tick from the baud clock. For LPUART instances, if *UARTn_CTRL.fdm* = 1, the fractional divisors are in increments of 0.5. The over-sampling rate must be no greater than this divisor. See *Baud Rate Generation* for information on how to use this field.

Table 11-13: UART Oversampling Control Register

UART Oversampling Control			UARTn_OSR	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	osr	R/W	0	**LPUART Over Sampling Rate** For LPUART instances with FDM enabled (*UARTn_CTRL.fdm* = 1): 0: 8 × 1: 12 × 2: 16 × 3: 20 × 4: 24 × 5: 28 × 6: 32 × 7: 36 × For LPUART instances with FDM disabled (*UARTn_CTRL.fdm* = 0): 0: 128 × 1: 64 × 2: 32 × 3: 16 × 4: 8 × 5: 4 × 6 - 7: Reserved

Table 11-14: UART Transmit FIFO Register

UART Transmit FIFO			UARTn_TXPEEK	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	RO	0	**Transmit FIFO Data** Read the transmit FIFO next data without effecting the contents of the transmit FIFO. If there are no entries in the transmit FIFO, this field reads 0. *Note: The parity bit is available from this field.*

Table 11-15: UART Pin Control Register

UART Pin Control			UARTn_PNR	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rts	R/W	1	**RTS Pin Output State** 0: RTS signal is driven to 0. 1: RTS signal is driven to 1.

UART Pin Control			UARTn_PNR	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
0	cts	RO	1	**CTS Pin State** Returns the current sampled state of the GPIO associated with the CTS signal. 0: CTS state is 0. 1: CTS state is 1.

Table 11-16: UART Data Register

UART Data			UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	rx_par	R	0	**Receive FIFO Byte Parity** If parity feature is disabled, this bit always read 0. If a parity error occurred during the reception of the character at the output end of the receive FIFO (that would be returned by reading the *UARTn_FIFO*.*data* field), this bit reads 1, otherwise it reads 0.
7:0	data	R/W	0	**Transmit/Receive FIFO Data** Writing to this field loads the next character into the transmit FIFO, if the transmit FIFO is not full. Reading from this field returns the next character from the receive FIFO, if the receive FIFO is not empty. If the receive FIFO is empty, 0 is returned by the hardware. For character widths less than 8, the unused bit(s) are ignored when the transmit FIFO is loaded, and the unused high bit(s) read 0 on characters read from the receive FIFO.

Table 11-17: UART DMA Register

UART DMA			UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	rx_en	0	0	**Receive DMA Channel Enable** 0: Disabled 1: Enabled
8:5	rx_thd_val	0	0	**Receive FIFO Level DMA Threshold** If *UARTn_STATUS*.*rx_lvl* < *UARTn_DMA*.*rx_thd_val*, then the receive FIFO DMA interface sends a signal to the DMA indicating characters are available in the UART receive FIFO to transfer to memory.
4	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled 1: Enabled
3:0	tx_thd_val	R/W	0	**Transmit FIFO Level DMA Threshold** If *UARTn_STATUS*.*tx_lvl* < *UARTn_DMA*.*tx_thd_val*, the transmit DMA channel sends a signal to the DMA indicating that the UART transmit FIFO is ready to receive data from memory.

Table 11-18: UART Wakeup Enable

UART Wakeup Enable			UARTn_WKEN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wakeup Event Enable** 0: Disabled 1: Enabled
1	rx_full	R/W	0	**Receive FIFO Full Wakeup Event Enable** 0: Disabled 1: Enabled
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wakeup Event Enable** 0: Disabled 1: Enabled

Table 11-19. UART Wakeup Flag Register

UART Wakeup Flag			UARTn_WKFL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wakeup Event** 0: Disabled 1: Enabled
1	rx_full	R/W	0	**Receive FIFO Full Wakeup Event** 0: Disabled 1: Enabled
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wakeup Event** 0: Disabled 1: Enabled

12. Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a highly configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, and a single, dual or quad data lines, and one or more slave select lines for communication with external SPI devices.

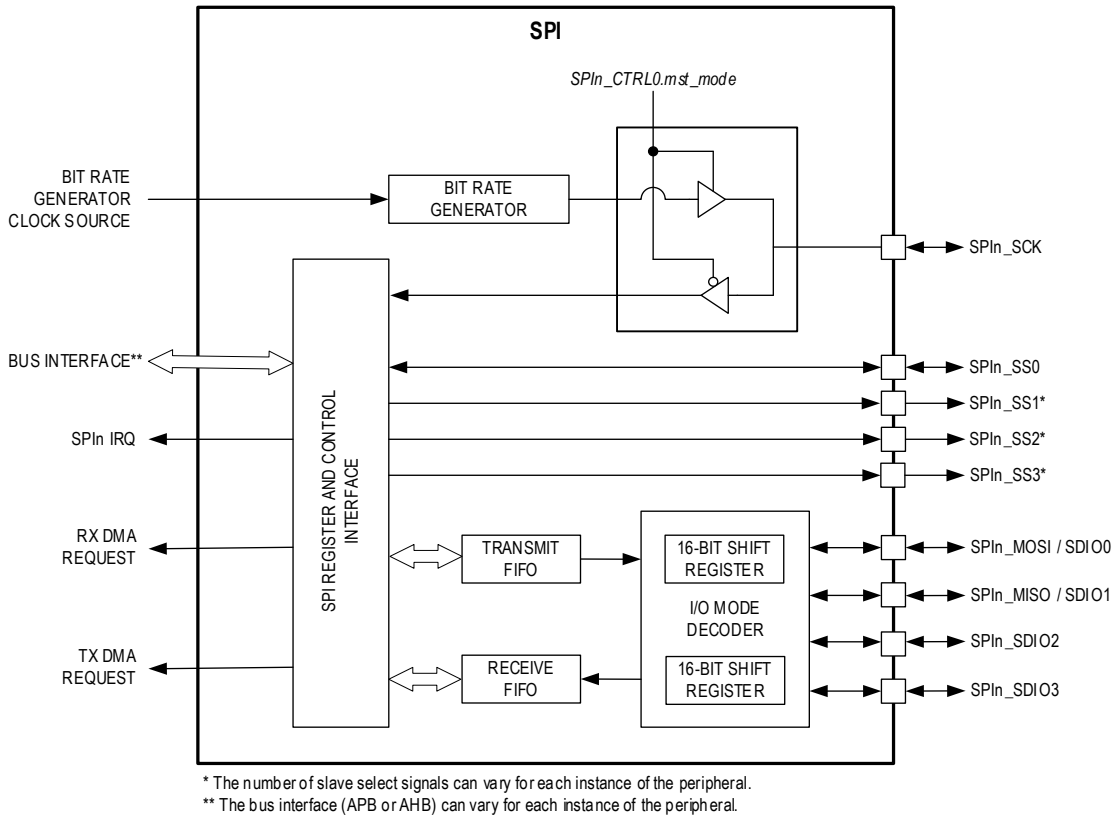
The provided SPI ports support full-duplex, bi-direction I/O and each SPI includes a Bit Rate Generator (BRG) for generating the clock signal when operating in master mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both master and slave modes, and support single master and multi-master networks.

Features include:

- Dedicated Bit Rate Generator for precision serial clock generation in master mode
  - Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus.
  - Programmabxle SCK duty cycle timing.
- Full-duplex, synchronous communication of 2 to 16-bit characters
  - 1-bit and 9-bit characters are not supported.
  - 2-bit and 10-bit characters do not support maximum clock speed. *SPIn_CLKCTRL*.*clkdiv* must be > 0.
- 3-wire and 4-wire SPI operation for single-bit communication.
- Single, Dual or Quad I/O operation.
- Byte-wide Transmit and Receive FIFOs with 32-byte depth
  - For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the transmit and receive FIFO.
- Transmit and receive DMA support.
- SPI modes 0, 1, 2, 3.
- Configurable slave select lines
  - Programmable slave select level.
- Programmable slave select timing with respect to SCK starting edge and ending edge.
- Multi-master mode fault detection.

Figure 12-1 shows a high-level block diagram of the SPI peripheral. See *Table 12-1* for the peripheral-specific peripheral bus assignment and bit rate generator clock source.

Figure 12-1: SPI Block Diagram



12.1 Instances

There are two instances of the SPI peripheral as shown in *Table 12-1*. *Table 12-2* lists the locations of the SPI signals for each of the SPI instances.

Table 12-1: MAX32655 SPI Instances

Instance	Formats				Hardware Bus	Bit Rate Generator Clock Source	Slave Select Signals
	3-Wire	4-Wire	Dual	Quad			81-CTBGA
SPI0	Yes	Yes	Yes	Yes	AHB	f_{SYS_CLK}	3
SPI1	Yes	Yes	Yes	Yes	APB	f_{PCLK}	1

Note: Refer to the MAX32655 data sheet for the definitive list of alternate function assignments for each peripheral.

Table 12-2: MAX32655 SPI Peripheral Pins

Instance	Signal Description	Alternate Function	Alternate Function Number	81 CTBGA
SPI0	SPI Clock	SPI0_SCK	AF1	P0.7
	Slave Select 0	SPI0_SS0	AF1	P0.4
	Slave Select 1	SPI0_SS1	AF2	P0.11
	Slave Select 2	SPI0_SS2	AF2	P0.10
	MOSI (SDIO0)	SPI0_MOSI	AF1	P0.5
	MISO (SDIO1)	SPI0_MISO	AF1	P0.6
	SDIO2	SPI0_SDIO2	AF1	P0.8
	SDIO3	SPI0_SDIO3	AF1	P0.9
SPI1	SPI Clock	SPI1_SCK	AF1	P0.23
	Slave Select 0	SPI1_SS0	AF1	P0.20
	MOSI (SDIO0)	SPI1_MOSI	AF1	P0.21
	MISO (SDIO1)	SPI1_MISO	AF1	P0.22
	SDIO2	SPI1_SDIO2	AF1	P0.24
	SDIO3	SPI1_SDIO3	AF1	P0.25

12.2 Formats

12.2.1 Four-Wire SPI

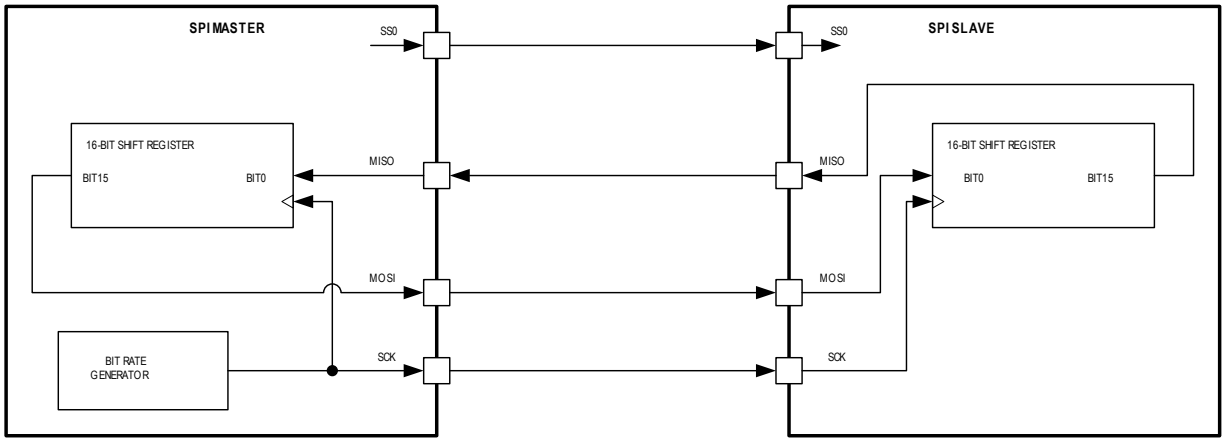
SPI devices operate as either a master or slave device. In four-wire SPI, four signals are required for communication as shown in *Table 12-3*.

Table 12-3: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the Serial Clock signal, which is an output from the master and an input to the slave.
MOSI	Master Output Slave Input	In master mode, this signal is used as an output for sending data to the slave. In slave mode this is the input data from the master.
MISO	Master Input Slave Output	In master mode, this signal is used as an input for receiving data from the slave. In slave mode, this signal is an output for transmitting data to the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device prior to communication. Peripherals may have multiple slave select outputs to communicate with one or more external slave devices. In slave mode, SPIn_SS0 is a dedicated input that indicates an external master is going to start communication. Other slave select signals into the peripheral are ignored in slave mode.

The SPI master starts communication with a slave by asserting the slave select output. The master then starts the SPI clock through the SCK output pin. When a slave device's slave select pin is deasserted, the slave device is required to put the SPI pins in tri-state mode.

Figure 12-2: 4-Wire SPI Connection Diagram



12.2.2 Three-Wire SPI

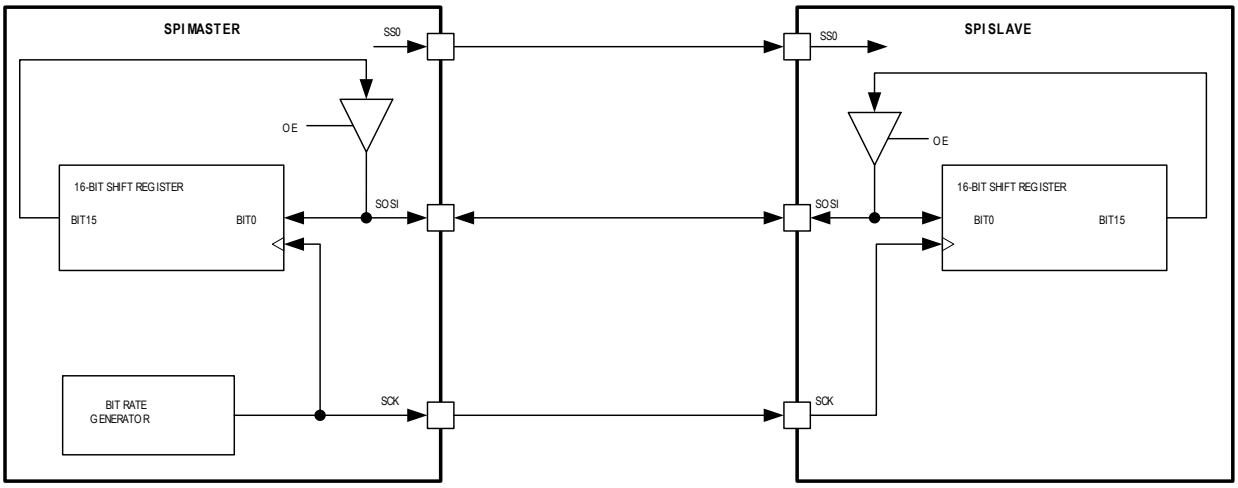
The signals in three-wire SPI operation are shown in *Table 12-4*, The MOSI signal is used as a bidirectional, half-duplex I/O referred to as slave input slave output (SISO). Three-wire SPI also uses a serial clock signal generated by the master and a slave select pin controlled by the master.

Table 12-4: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the serial clock signal, which is an output from the master and an input to the slave.
MOSI	Slave Input Slave Output	This is a half-duplex, bidirectional I/O pin used for communication between the SPI master and slave. This signal is used to transmit data from the master to the slave and to receive data from the slave by the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device prior to communication. In slave mode, SPIn_SS0 is a dedicated input that indicates an external master is going to start communication. Other slave select signals into the peripheral are ignored in slave mode

A three-wire SPI network is shown in *Figure 12-3*. The master device selects the slave device using the slave select output. The communication starts with the master asserting the slave select line and then starting the clock (SCK). In three-wire SPI communication, the master and slave must both know the intended direction of the data to prevent bus contention. For a write, the master drives the data out the SISO pin. For a read, the master must release the SISO line and let the slave drive the SISO line. The direction of transmission is controlled using the FIFO. Writing to the FIFO starts the three-wire SPI write and reading from the FIFO starts a three-wire SPI read transaction.

Figure 12-3: Generic 3-Wire SPI Master to Slave Connection



12.3 Pin Configuration

Before configuring the SPI peripheral, first disable any SPI activity for the port by clearing the *SPIn_CTRL0.en* field to 0.

12.3.1 SPIn Alternate Function Mapping

Pin selection and configuration is required to use the SPI port. The following information applies to SPI master and slave operation as well as three-wire, four-wire, dual and quad mode communications. Determine the pins required for the SPI type and mode in the application, and configure the required GPIO as described in the following sections. Refer to the MAX32655 data sheet for pin availability for a specific package.

When the SPI port is disabled, *SPIn_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

12.3.2 Four-wire Format Configuration

Four-wire SPI uses SCK, MISO, MOSI, and one or more SS pins. Four-wire SPI may use more than one slave select pin for a transaction, resulting in more than four wires total, however the communication is referred to as four-wire for historical reasons.

Note: Select the pins mapped to the SPI external device in the design and modify the setup accordingly. There is no restriction on which alternate function is used for a specific SPI pin and each SPI pin can be used independently from the other pins chosen. However, it is recommended that only one set of GPIO port pins be used for any network.

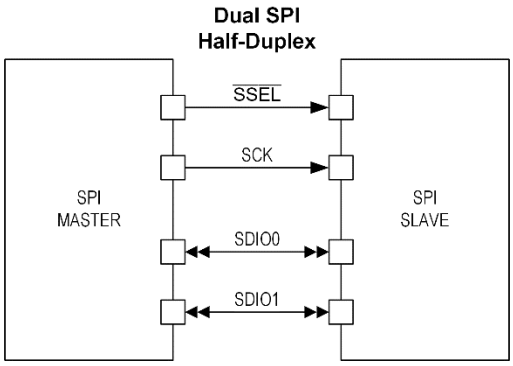
12.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK and MOSI, and one or more slave select pins for an SPI transaction. Three-wire SPI configuration is identical to the four-wire configuration except SPIn_MISO does not need to be set up for the SPI alternate function. The direction of communication in three-wire SPI mode is controlled by the SPIn Transmit and Receive FIFO enables. Enabling the Receive FIFO and disabling the Transmit FIFO indicates a read transaction. Enabling the Transmit FIFO and disabling the Receive FIFO indicates a write transaction. It is an illegal condition to enable both the Transmit and Receive FIFOs in three-wire SPI operation.

12.3.4 Dual-Mode Format Configuration

In dual-mode SPI, two I/O pins are used to transmit 2-bits of data per SCK clock cycle. The communication is half-duplex and the direction of the data transmission must be known by both the master and slave for a given transaction. Dual-mode SPI uses SCK, SDIO0, SDIO1, and one or more slave select lines as shown in *Figure 12-4*. The configuration of the GPIO pins for dual-mode SPI is identical to four-wire SPI and the mode is controlled by setting *SPIn_CTRL2.data_width* to 1, indicating to the SPIn hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 12-4: Dual Mode SPI Connection Diagram



12.3.5 Quad-Mode Format Pin Configuration

Quad-mode SPI uses four I/O pins to transmit four-bits of data per transaction. In quad-mode SPI, the communication is half-duplex, and the master and slave must know the direction of transmission for each transaction. Quad-mode SPI uses SCK, SDIO0, SDIO1, SDIO2, SDIO3, and one or more slave select pins.

Quad-mode SPI transmits four bits per SCK cycle. Selection of quad-mode SPI is selected by setting *SPIn_CTRL2.data_width* to 2.

12.4 Clock Configuration

12.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The master drives SCK as an output to the slave's SCK pin. When SPIn is set to master mode, the SPIn bit rate generator creates the serial clock and outputs it on the configured SPIn_SCK pin. When SPIn is configured for slave operation, the SPIn_SCK pin is an input from the external master and the SPIn hardware synchronizes communications using the SCK input. Operating as a slave, if a SPIn slave select input is not asserted, the SPIn ignores any signals on the serial clock and serial data lines.

In both master and slave devices, data is shifted on one edge of the SCK and is sampled on the opposite edge where data is stable. Data availability and sampling time is controlled using the SPI phase control field, *SPIn_CTRL2.clkpha*. The SCK clock polarity field, *SPIn_CTRL2.clkpol*, controls if the SCK signal is active high or active low.

The SPIn peripheral supports four combinations of SCK phase and polarity referred to as SPI modes 0, 1, 2, and 3. Clock Polarity (*SPIn_CTRL2.clkpol*) selects an active low/high clock and has no effect on the transfer format. Clock Phase (*SPIn_CTRL2.clkpha*) selects one of two different transfer formats.

For proper data transmission, the clock phase and polarity must be identical for the SPI master and slave. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data. See section *Clock Phase and Polarity Control* for additional details.

12.4.2 Peripheral Clock

See *Table 12-1* for the specific input clock, f_{INPUT_CLK}, used for each SPI instance. For SPI instances assigned to the AHB bus, the SPI input clock is the system clock, SYS_CLK. For SPI instances mapped to the APB bus, the SPI input clock is the system

peripheral clock, PCLK. The SPI input clock drives the SPIn peripheral clock. The SPIn provides an internal clock, *SPIn_CLK*, that is used within the SPI peripheral for the base clock to control the module and generate the SCK clock when in master mode. Set the SPIn internal clock using the field *SPIn_CLKCTRL*.*scale* as shown in *Equation 12-1*. Valid settings for *SPIn_CLKCTRL*.*scale* are 0 to 8, allowing a divisor of 1 to 256.

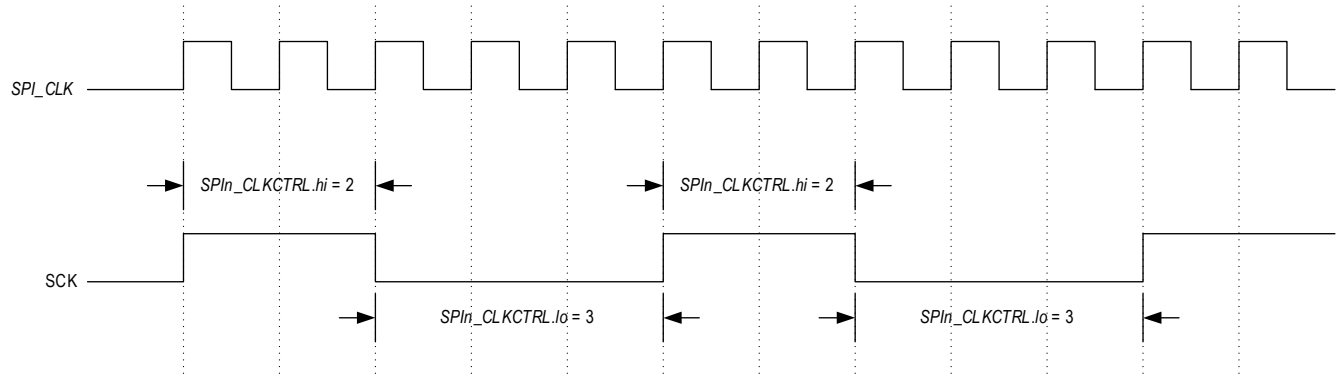
Equation 12-1: SPI Peripheral Clock

$$f_{SPI\_CLK} = \frac{f_{INPUT\_CLK}}{2^{clkdiv}}$$

12.4.3 Master Mode Serial Clock Generation

In master and multi-master mode, the SCK clock is generated by the master. The SPIn provides control for both the high time and low time of the SCK clock. This control allows setting the high and low times for the SCK to duty cycles other than 50% if required. The SCK clock uses the SPIn peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 12-5*, visually represents the use of the *SPIn_CLKCTRL*.*hi* and *SPIn_CLKCTRL*.*lo* fields for a non-50% duty cycle serial clock generation. See *Equation 12-2* and *Equation 12-3* for calculating the SCK high and low time from the *SPIn_CLKCTRL*.*hi* and *SPIn_CLKCTRL*.*lo* field values.

Figure 12-5: SCK Clock Rate Control



Equation 12-2: SCK High Time

$$t_{SCK\_HI} = t_{SPIn\_CLK} \times SPIn\_CLKCTRL.hi$$

Equation 12-3: SCK Low Time

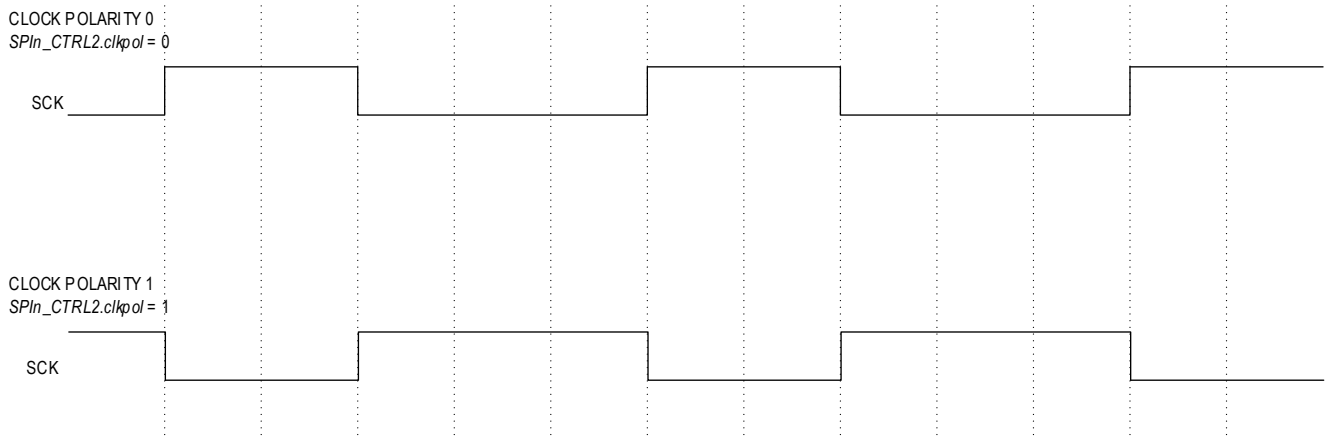
$$t_{SCK\_LOW} = t_{SPIn\_CLK} \times SPIn\_CLKCTRL.lo$$

12.4.4 Clock Phase and Polarity Control

SPIn supports four combinations of clock and phase polarity as shown in *Table 12-5*. Clock polarity is controlled using the bit *SPIn_CTRL2*.*clkpol* and determines if the clock is active high or active low as shown in *Figure 12-6*. Clock polarity does not affect the transfer format for SPI. Clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2*.*clkpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity.

Phase 1, *SPIn_CTRL2.clkpha* = 1, results in data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 12-6: SPI Clock Polarity



For proper data transmission, the clock phase and polarity must be identical for the SPI master and slave. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data.

Table 12-5: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2 clkpha*	*SPIn_CTRL2 clkpol*	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	High
2	1	0	Rising	Falling	Low
3	1	1	Falling	Rising	High

12.4.5 Transmit and Receive FIFOs

The Transmit FIFO hardware is 32 bytes deep. The write data width can be 8-, 16- or 32-bits wide. A 16-bit write queues a 16-bit word to the FIFO hardware. A 32-bit write queues two 16-bit words to the FIFO hardware with the least significant word dequeued first. Bytes must be written to two consecutive byte addresses, with the odd byte as the most significant byte, and the even byte as the least significant byte. The FIFO logic waits for both the odd and even bytes to be written to this register space before dequeuing the 16-bit result to the FIFO.

The Receive FIFO hardware is 32 bytes deep. Read data width can be 8-, 16- or 32-bits. A byte read from this register dequeues one byte from the FIFO. A 16-bit read from this register dequeues two bytes from the FIFO, least significant byte first. A 32-bit read from this register dequeues four bytes from the FIFO, least significant byte first.

12.4.6 Interrupts and Wakeups

The SPIn supports multiple interrupt sources. Status flags for each interrupt are set regardless of the state of the interrupt enable bit for that event. The event happens once when the condition is satisfied. The status flag must be cleared by the software by writing a 1 to the interrupt flag.

The following FIFO interrupts are supported:

- Transmit FIFO Empty
- Transmit FIFO Threshold
- Receive FIFO Full
- Receive FIFO Threshold
- Transmit FIFO Underrun
  - Slave mode only, master mode stalls the serial clock
- Transmit FIFO Overrun
- Receive FIFO Underrun
- Receive FIFO Overrun
  - Slave mode only, master mode stalls the serial clock
- SPI*n* supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:
- SS*n* asserted or deasserted
- SPI transaction complete
  - Master mode only
- Slave mode transaction aborted
- Multi-master fault

The SPIn port can wake up the microcontroller from low-power modes when the wake event is enabled. SPIn events that can wake the microcontroller are:

- Receive FIFO full
- Transmit FIFO empty
- Receive FIFO threshold
- Transmit FIFO threshold

12.5 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 12-6*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 12-6: SPIn Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SPIn_FIFO32*	R/W	*SPI FIFO Data Register*
[0x0000]	*SPIn_FIFO16*	R/W	*SPI 16-bit FIFO Data Register*
[0x0000]	*SPIn_FIFO8*	R/W	*SPI 8-bit FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	R/W	*SPI Master Signals Control Register*
[0x0008]	*SPIn_CTRL1*	R/W	*SPI Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	R/W	*SPI Static Configuration Register*
[0x0010]	*SPIn_SSTIME*	R/W	*SPI Slave Select Timing Register*
[0x0014]	*SPIn_CLKCTRL*	R/W	*SPI Master Clock Configuration Register*
[0x001C]	*SPIn_DMA*	R/W	*SPI DMA Control Register*
[0x0020]	*SPIn_INTFL*	R/W1C	*SPI Interrupt Flag Register*
[0x0024]	*SPIn_INTEN*	R/W	*SPI Interrupt Enable Register*
[0x0028]	*SPIn_WKFL*	R/W1C	*SPI Wakeup Flags Register*

Offset	Register Name	Access	Description
[0x002C]	*SPIn_WKEN*	R/W	*SPI Wakeup Enable Register*
[0x0030]	*SPIn_STAT*	RO	*SPI Status Register*

12.5.1 Register Details

Table 12-7: SPI FIFO32 Register

SPI FIFO Data			SPIn_FIFO32	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SPIn FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO and writing to this register adds characters to the Transmit FIFO. Read and write this register in either 1-byte, 2-byte, or 4-byte widths only. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-8: SPI 16-bit FIFO Register

SPI FIFO Data			SPIn_FIFO16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	data	R/W	0	**SPIn 16-bit FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO and writing to this register adds characters to the Transmit FIFO. Read and write this register in 2-byte width only for 16-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-9: SPI 8-bit FIFO Register

SPI 8-bit FIFO Data			SPIn_FIFO8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Reserved**
7:0	data	R/W	0	**SPIn 8-bit FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO and writing to this register adds characters to the Transmit FIFO. Read and write this register in 1-byte width only for 8-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-10: SPI Control 0 Register

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	ss_active	R/W	0	**Master Slave Select** The SPIn includes up to four slave select lines for each port. This field selects which slave select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more slave select pins can be selected for each SPI transaction by setting the bit for each slave select pin. For example, use SPIn_SS0 and SPIn_SS2 by setting this field to 0b0101 or select all slave selects by setting this field to 0b1111 *Note: This field is only used when the SPIn is configured for master mode (SPIn_CTRL0.mst_mode = 1).*
15:9	-	R/W	0	**Reserved**
8	ss_ctrl	R/W	0	**Master Slave Select Control** This field controls the behavior of the slave select pins at the completion of a transaction. The default behavior, *ss_ctrl* = 0, deasserts the slave select pin at the completion of the transaction. Set this field to 1 to leave the slave select pins asserted at the completion of the transaction. If the external device supports this behavior, leaving the slave select pins asserted allows multiple transactions without the delay associated with deassertion of the slave select pin between transactions. 0: Slave Select is deasserted at the end of a transmission 1: Slave Select stays asserted at the end of a transmission
7:6	-	R/W	0	**Reserved.**
5	start	R/W1O	0	**Master Start Data Transmission** Set this field to 1 to start a SPI master mode transaction. 0: No master mode transaction active. 1: Master initiates a data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPIn is configured for master mode (SPIn_CTRL0.master = 1).*
4	ss_io	R/W	0	**Master Slave Select Signal Direction** Set the I/O direction for 0: Slave select is an output 1: Slave select is an input *Note: This field is only used when the SPIn is configured for master mode (SPIn_CTRL0.mst_mode = 1).*
3:2	-	R/W	0	**Reserved**
1	mst_mode	R/W	0	**SPI Master Mode Enable** This field selects between slave mode and master mode operation for the SPI port. Write this field to 0 to operate as an SPI slave. Setting this field to 1 sets the port as an SPI master. 0: Slave mode SPI operation. 1: Master mode SPI operation.
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPIn port. Disable the SPIn port by setting this field to 0. Disabling the SPIn port does not affect the SPIn FIFOs or register settings. Access to SPIn registers is always available. 0: SPIn port is disabled 1: SPIn port is enabled

Table 12-11: SPI Control 1 Register

SPI Transmit Packet Size			SPIn_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** Number of characters to receive in receive FIFO. *Note: If the SPIn port is set to operate in 4-wire mode, this field is ignored and the SPIn_CTRL1.tx_num_chars field is used for both the number of characters to receive and transmit.*
15:0	tx_num_char	R/W	0	**Number of Transmit Characters** Number of characters to transmit from transmit FIFO. *Note: If the SPIn port is set to operate in 4-wire mode, this field is used for both the number of characters to receive and transmit.*

Table 12-12: SPI Control 2 Register

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**
19:16	ss_pol	R/W	0	**Slave Select Polarity** Controls the polarity of each individual SS signal where each bit position corresponds to a SS signal. SPIn_SS0 is controlled with bit position 0 and SPIn_SS2 is controlled with bit position 2. For each bit position, 0: SS is active low 1: SS is active high
15	three_wire	R/W	0	**Three-Wire SPI Enable** Set this field to 1 to enable three-wire SPI communication. Set this field to 0 for four-wire full-duplex SPI communication. 0: Four-wire full-duplex mode enabled. 1: Three-wire mode enabled *Note: This field is ignored for Dual SPI, SPIn_CTRL2.data_width =1, and Quad SPI, SPIn_CTRL2.data_width =2.*
14	-	R/W	0	**Reserved**

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. *Three-wire SPI*: *data_width* = 0 Set this field to 0, indicating SPIn_MOSI is used for half-duplex communication. *Four-wire full-duplex SPI*: *data_width* = 0 Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input respectively. *Dual mode SPI*: *data_width* = 1 Set this field to 1, indicating SPIn_SDIO0 and SPIn_SDIO1 are used for half-duplex communication. *Quad mode SPI*: *data_width* = 2 Set this field to 2, indicating SPIn_SDIO0, SPIn_SDIO1, SPIn_SDIO2 and SPIn_SDIO3 are used for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and Four-wire full-duplex SPI) 1: 2-bits per SCK cycle (Dual mode SPI) 2: 4-bits per SCK cycle (Quad mode SPI) 3: Reserved *Note: When this field is set to 0, use the field SPIn_CTRL2.three_wire to select either Three-Wire SPI or Four-Wire SPI operation.*
11:8	numbits	R/W	0	**Number of Bits per Character** Set this field to the number of bits per character for the SPI transaction. Setting this field to 0 indicates a character size of 16. 0: 16-bits per character 1: 1-bit per character (not supported) 2: 2-bits per character … 14: 14-bits per character 15: 15-bits per character *Note: 1-bit and 9-bit character lengths are not supported.* *Note: 2-bit and 10-bit character lengths do not support maximum SCK speeds in master mode. SPIn_CLK.scale must be > 0* *Note: For Dual and Quad mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	R/W	0	**Reserved**
1	clkpol	R/W	0	**Clock Polarity** This field controls the SCK polarity. The default clock polarity is for SPI mode 0 and mode 1 operation and is active high. Invert the SCK polarity for SPI mode 2 and mode 3 operation. 0: Standard SCK for use in SPI mode 0 and mode 1 1: Inverted SCK for use in SPI mode 2 and mode 3
0	clkpha	R/W	0	**Clock Phase** 0: Data sampled on clock rising edge. Use when in SPI mode 0 and mode 2 1: Data sampled on clock falling edge. Use when in SPI mode 1 and mode 3

Table 12-13: SPI Slave Select Timing Register

SPI Slave Select Timing			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	0	**Reserved**

SPI Slave Select Timing			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
23:16	inact	R/W	0	**Inactive Stretch** This field controls the number of system clocks the bus is inactive between the end of a transaction (slave select inactive) and the start of the next transaction (slave select active). 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPIn is operating in master mode (SPIn_CTRL0.master = 1)*
15:8	post	R/W	0	**Slave Select Hold Post Last SCK** Number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPIn is operating in master mode (SPIn_CTRL0.master = 1)*
7:0	pre	R/W	0	**Slave Select Delay to First SCK** Set the number of system clock cycles the slave select is held active prior to the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPIn is operating in master mode (SPIn_CTRL0.master = 1)*

Table 12-14: SPI Master Clock Configuration Registers

SPI Master Clock Configuration			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**

SPI Master Clock Configuration			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	clkdiv	R/W	0	**SPI Peripheral Clock Scale** Scales the SPI input clock (PCLK) by 2^{scale} to generate the SPIn peripheral clock. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{clkdiv}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved for future use. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** 0: Hi duty cycle control disabled. Only valid if *SPIn_CLK.clkdiv* = 0. 1 - 15: The number of SPIn peripheral clocks, $f_{SPInCLK}$, that SCK is high. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and is used to control the overall SCK duty cycle in combination with the *SPIn_CLK.hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if *SPIn_CLK.clkdiv* = 0. 1 to 15: The number of SPIn peripheral clocks, $f_{SPInCLK}$, that the SCK signal is low. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*

Table 12-15: SPI DMA Control Registers

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	dma_rx_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared 1: Enabled
30:24	dma_rx_en	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO
23	rx_flush	W1O	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in *SPIn_INTFL*. This should be done when the receive FIFO is inactive. Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled 1: Enabled
21	-	R/W	0	**Reserved**
20:16	rx_thd_val	R/W	0x00	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO level crosses above this setting, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_tx_en*, and *SPIn_INTFL.rx_thd_val* becomes set. Valid values are 0 to 30. *Note: 31 is an invalid setting and reserved for future use.*

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14:8	tx_lvl	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes currently in the transmit FIFO.
7	tx_flush	R/W	0	**Transmit FIFO Clear** Set this bit to clear the transmit FIFO and all transmit FIFO flags in the *SPIn_INTFL* register. *Note: The transmit FIFO should be disabled (SPIn_DMA.tx_fifo_en = 0) prior to setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** 0: Disabled 1: Enabled
5	-	R/W	0	**Reserved**
4:0	tx_thd_val	R/W	0x10	**Transmit FIFO Threshold Level** Set this value to the desired transmit FIFO threshold level. When the transmit FIFO count (*SPIn_DMA.tx_lvl*) falls below this value, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_tx_en* and *SPIn_INTFL.tx_thd_val* becomes set.

Table 12-16: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_un	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ov	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI is in slave mode, and a write to a full receive FIFO is attempted. If the SPI is in master mode, this bit is not set as the SPI stalls the clock until data is read from the receive FIFO.
13	tx_un	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI is in slave mode, and a read from empty transmit FIFO is attempted. If SPI is in master mode, this bit is not set as the SPI stalls the clock until data is written to the empty transmit FIFO.
12	tx_ov	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted to a full transmit FIFO.
11	mst_done	R/W1C	0	**Master Data Transmission Done Flag** Set if SPIn is in master mode, and all transactions have completed. *SPIn_CTRL1.tx_num_char* has been reached.
10	-	R/W	0	**Reserved**
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** Set if the SPI is in slave mode, and SS is deasserted before a complete character is received.

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
8	fault	R/W1C	0	**Multi-Master Fault Flag** Set if the SPI is in master mode, multi-master mode is enabled, and a slave select input is asserted. A collision also sets this flag.
7:6	-	R/W	0	**Reserved**
5	ssd	R/W1C	0	**Slave Select Deasserted Flag**
4	ssa	R/W1C	0	**Slave Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag**
2	rx_thd	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPIn_DMA.rx_fifo_level*. Cleared once receive FIFO level drops below *SPIn_DMA.rx_fifo_level.*
1	tx_em	R/W1C	1	**Transmit FIFO Empty Flag** The transmit FIFO is empty.
0	tx_thd	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** Set when the transmit FIFO is less than the value in *SPIn_DMA.tx_fifo_level*. Cleared once transmit FIFO level exceeds *SPIn_DMA.tx_fifo_level,*

Table 12-17: SPI Interrupt Enable Registers

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_un	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Disabled 1: Enabled
14	rx_ov	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled 1: Enabled
13	tx_un	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Disabled 1: Enabled
12	tx_ov	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Disabled 1: Enabled
11	mst_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 0: Disabled 1: Enabled
10	-	R/W	0	**Reserved**
9	abort	R/W	0	**Slave Mode Abort Detected Interrupt Enable** 0: Disabled 1: Enabled
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 0: Disabled 1: Enabled
7:6	-	R/W	0	**Reserved**
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 0: Disabled 1: Enabled

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 0: Disabled 1: Enabled
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Disabled 1: Enabled
2	rx_thd	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled 1: Enabled
1	tx_em	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Disabled 1: Enabled
0	tx_thd	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled 1: Enabled

Table 12-18: SPI Wakeup Status Flags Registers

SPI Wakeup Flags			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thd	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
1	tx_em	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thd	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 12-19: SPI Wakeup Enable Registers

SPI Wakeup Enable			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W	0	**Wake on Receive FIFO Full Enable** 0: Wake event is disabled 1: Wake event is enabled.
2	rx_thd	R/W	0	**Wake on Receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.
1	tx_em	R/W	0	**Wake on Transmit FIFO Empty Enable** 0: Wake event is disabled 1: Wake event is enabled.
0	tx_thd	R/W	0	**Wake on Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.

Table 12-20: SPI Slave Select Timing Registers

SPI Status			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved**
0	busy	R	0	**SPI Active Status** SPI status flag, see value descriptions for details of each value. 0: SPIn is not active. In master mode, the *busy* flag is cleared when the last character is sent. In slave mode, the *busy* field is cleared when the configured slave select input is deasserted. 1: SPIn is active. In master mode, the *busy* flag is set when a transaction starts. In slave mode, the *busy* flag is set when a configured slave select input is asserted. *Note: SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SSTIME, and* SPIn_CLKCTRL *should not be configured if this bit is set.*

13. I²C Master/Slave Serial Communications Peripheral (I2C)

The I²C peripherals can be configured as either an I²C master or I²C slave at standard data rates. For simplicity, I2Cn is used throughout this section to refer to any of the I²C peripherals.

For detailed information on I²C bus operation, refer to Maxim Application Note 4024 "SPI/I²C Bus Lines Control Multiple Peripherals" at *https://www.maximintegrated.com/en/app-notes/index.mvp/id/4024*

13.1 I²C Master/Slave Features

Each I²C master/slave is compliant with the I²C Bus Specification and include the following features:

- Communicates through a serial data bus (SDA) and a serial clock line (SCL).
- Operates as either a master or slave device as a transmitter or receiver.
- Supports I²C Standard Mode, Fast Mode, Fast Mode Plus, and High Speed (Hs) Mode.
- Transfers data at rates up to:
  - 100kbps in Standard Mode.
  - 400kbps in Fast Mode.
  - 1Mbps in Fast Mode Plus.
  - 3.4Mbps in Hs Mode.
- Supports multi-master systems, including support for arbitration and clock synchronization for Standard Mode, Fast Mode, and Fast Mode Plus.
- Supports 7- and 10-bit addressing.
- Supports RESTART condition.
- Supports clock stretching.
- Provides transfer status interrupts and flags.
- Provides DMA data transfer support.
- Supports I²C timing parameters fully controllable through firmware.
- Provides glitch filter and Schmitt trigger hysteresis on SDA and SCL.
- Provides control, status, and interrupt events for maximum flexibility.
- Provides independent 8-byte receive FIFO and 8-byte transmit FIFO.
- Provides transmit FIFO preloading.
- Provides programmable interrupt threshold levels for the transmit and receive FIFO.

13.2 Instances

The three instances of the peripheral shown in *Table 13-1* list the locations of the SDA and SCL signals for each of the I2Cn peripherals per package.

Table 13-1: MAX78000 I²C Peripheral Pins

I²C Instance	Alternate Function	Alternate Function #	Package 81-CTBGA
I2C0	I2C0_SCL	AF1	P0.10
	I2C0_SDA	AF1	P0.11
I2C1	I2C1_SCL	AF1	P0.16
	I2C1_SDA	AF1	P0.17
I2C2	I2C2_SCL	AF1	P0.30
	I2C2_SDA	AF1	P0.31

13.3 I²C Overview

13.3.1 I²C Bus Terminology

Table 13-2 contains terms and definitions used in this chapter for the I²C bus terminology.

Table 13-2: I²C Bus Terminology

Term	Definition
Transmitter	The device that sends data to the bus.
Receiver	The device that receives data from the bus.
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one can do so and the resulting message is not corrupted.
Synchronization	Procedure to synchronize the clock signals of two or more devices.
Clock Stretching	When a slave device holds SCL low to pause a transfer until it is ready. This feature is optional according to the I²C Specification; thus, a master does not have to support slave clock stretching if none of the slaves in the system are capable of clock stretching.

13.3.2 I²C Transfer Protocol Operation

The I²C protocol operates over a two-wire bus: a clock circuit (SCL) and a data circuit (SDA). I²C is a half-duplex protocol: only one device is allowed to transmit on the bus at a time.

Each transfer is initiated when the bus master sends a START or repeated START condition. It is followed by the I²C slave address of the targeted slave device plus a read/write bit. The master can transmit data to the slave (a 'write' operation) or receive data from the slave (a 'read' operation). Information is sent most-significant-bit (MSB) first. Following the slave address, the master indicates a read or write operation and then exchanges data with the addressed slave. An acknowledge bit is sent by the receiving device after each byte is transferred. When all necessary data bytes have been transferred, a STOP or RESTART condition is sent by the bus master to indicate the end of the transaction. After the STOP condition has been sent, the bus is idle and ready for the next transaction. After a RESTART condition is sent, the same master begins a new transmission. The number of bytes that can be transmitted per transfer is unrestricted.

13.3.3 START and STOP Conditions

A START condition occurs when a bus master pulls SDA from high to low while SCL is high, and a STOP condition occurs when a bus master allows SDA to be pulled from low to high while SCL is high. Because these are unique conditions that cannot occur during normal data transfer, they are used to denote the beginning and end of the data transfer.

13.3.4 Master Operation

I²C transmit and receive data transfer operations occur through the *I2Cn_FIFO* register. Writes to the register load the transmit FIFO and reads of the register return data from the receive FIFO. If a slave sends a NACK in response to a write operation, the I²C master generates an interrupt. The I²C controller can be configured to issue a STOP condition to free the bus.

The receive FIFO contains the received data. If the receive FIFO is full or the transmit FIFO is empty, the I²C master stops the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

13.3.5 Acknowledge and Not Acknowledge

An acknowledge bit (ACK) is generated by the receiver, whether I²C master or slave, after every byte received by pulling SDA low. The ACK bit is how the receiver tells the transmitter that the byte was successfully received, and another byte might be sent.

A Not Acknowledge (NACK) occurs if the receiver does not generate an ACK when the transmitter releases SDA. A NACK is generated by allowing SDA to float high during the acknowledge time slot. The I²C master can then either generate a STOP condition to abort the transfer, or it can generate a repeated START condition (that is, send a START condition without an intervening STOP condition) to start a new transfer.

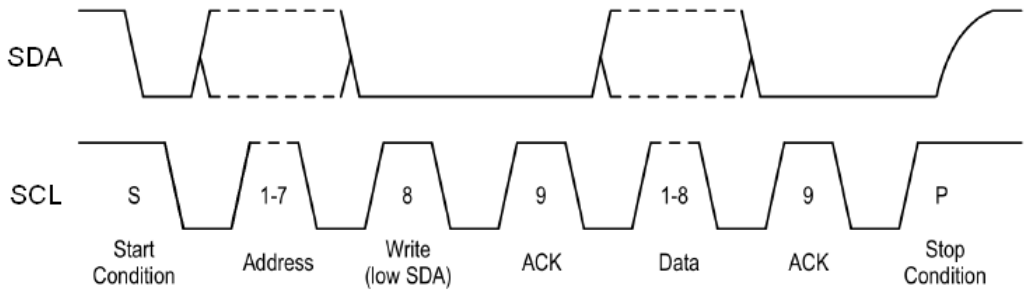
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device responds with an acknowledge signal.
- The receiver is unable to receive or transmit because it is busy and is not ready to start communication with the master.
- During the transfer, the receiver receives data or commands it does not understand.
- During the transfer, the receiver is unable to receive any more data.
- If an I²C master has requested data from a slave, it signals the slave to stop transmitting by sending a NACK following the last byte it requires.

13.3.6 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each requires an external pullup resistor that ensures each circuit is high when idle. The I²C specification states that during data transfer, the SDA line can change state only when SCL is low, and that SDA is stable and able to be read when SCL is high as shown in *Figure 13-1*.

Figure 13-1: I²C Write Data Transfer



An example of an I²C data transfer is as follows:

1. A bus master indicates a data transfer to a slave with a START condition.
2. The master then transmits one byte with a 7-bit slave address and a single read-write bit: a zero for a write or a one for a read.
3. During the next SCL clock following the read-write bit, the master releases SDA. During this clock period, the addressed slave responds with an ACK by pulling SDA low.
4. The master senses the ACK condition and begins transferring data. If reading from the slave, it floats SDA and allows the slave to drive SDA to send data. After each byte, the master drives SDA low to acknowledge the byte. If writing to the slave, the master drives data on the SDA circuit for each of the eight bits of the byte, and then floats SDA during the ninth bit to allow the slave to reply with the ACK indication.
5. After the last byte is transferred, the master indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the master pulls SDA from a low to high while SCL is high.

13.4 Configuration and Usage

13.4.1 SCL and SDA Bus Drivers

SCL and SDA are open-drain signals. In this device, once the I²C peripheral is enabled and the proper GPIO alternate function is selected, the corresponding pad circuits are automatically configured as open-drain outputs. However, SCL can also be optionally configured as a push-pull driver to conserve power and avoid the need for any pullup resistor. This should only be used in systems where no I²C slave device can hold SCL low, such as for clock stretching. Push-pull operation is enabled by setting *I2Cn_CTRL.sclppm* to 1. SDA, on the other hand, always operates in open-drain mode.

13.4.2 SCL Clock Configurations

The SCL frequency is dependent upon the values of I²C peripheral clock and the values of the external pullup resistor and trace capacitance on the SCL clock line.

Note: An external RC load on the SCL line affects the target SCL frequency calculation.

13.4.3 SCL Clock Generation for Standard, Fast and Fast-Plus Modes

The master generates the I²C clock on the SCL line. When operating as a master, application code must configure the*I2Cn_CLKHI* and *I2Cn_CLKLO* registers for the desired I²C operating frequency.

The SCL high time is configured in the I²C Clock High Time register field *I2Cn_CLKHI.hi* using *Equation 13-2*. The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLKLO.lo* using *Equation 13-3*. Each of these fields is 8-bits. The I²C frequency value is shown in *Equation 13-1*.

Equation 13-1: I²C Clock Frequency

$$f_{I2C\_CLK} = \frac{1}{t_{I2C\_CLK}} \; is\ either\ f_{PCLK}\ or\ f_{IBRO}$$

Equation 13-2: I²C Clock High Time Calculation

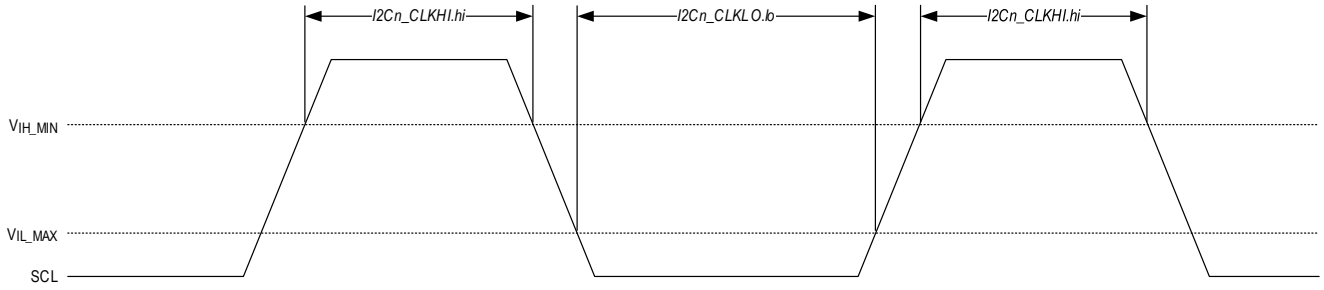
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (\text{I2Cn\_CLKHI}.hi + 1)$$

Equation 13-3: I²C Clock Low Time Calculation

$$t_{SCL\_LO} = t_{I2C\_CLK} \times (\text{I2Cn\_CLKLO}.lo + 1)$$

Figure 13-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 13-2: I²C SCL Timing for Standard, Fast and Fast-Plus Modes



During synchronization, external masters or external slaves may drive SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller determines if an external master or slave is holding SCL low. In either case, the controller waits until SCL is high before starting to count the number of SCL high cycles. Similarly, if an external master pulls SCL low

before the controller has finished counting SCL high cycles, then the controller starts counting SCL low cycles and releases SCL once the time period,.*lo*, has expired.

Because the controller does not start counting the high/low time until the input buffer detects the new value, the actual clock behavior is based on many factors; including bus loading, other devices on the bus holding SCL low, and the filter delay time of this device.

13.4.4 SCL Clock Generation for Hs-mode

To operate the I²C interface in Hs-mode at its maximum speed (~3.4MHz), values to be programmed into the *I2Cn_HSCLK.hsclk_lo* register and *I2Cn_HSCLK.hsclk_hi* register must be determined. Since the Hs-mode operation is entered by first using one of the lower speed modes for preamble, a relevant lower speed mode must also be configured. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding configuration of lower speed modes.

13.4.4.1 Hs-Mode Timing

With I²C bus capacitances less than 100pf, the following specifications are extracted from the I²C-bus Specification User Manual Rev. 6 April 2014 *https://www.nxp.com/docs/en/user-guide/UM10204.pdf*

t_{LOW_MIN} = 160ns, the minimum low time for the I²C bus clock.

t_{HIGH_MIN} = 60ns, the minimum high time for the I²C bus clock.

t_{rCL_MAX} = 40ns, the maximum rise time of the I²C bus clock.

t_{fCL_MAX} = 40ns, the maximum fall time of the I²C bus clock.

13.4.4.2 Hs-Mode Clock Configuration

The maximum Hs-mode bus clock frequency can now be determined. The system clock frequency, f_{SYS_CLK}, must be known. Hs-mode timing information from *Hs-Mode Timing* must be used.

Equation 13-4: I²C Target SCL Frequency

Desired Target Maximum I²C Frequency: $f_{SCL} = \frac{1}{t_{SCL}}$.

In Hs-mode, the analog glitch filter (AF_MIN) within the device adds a minimum delay of t_{AF_MIN} = 10ns.

Equation 13-5: Determining the I2Cn_HSCLK.hsclk_lo Register Value

$$I2Cn\_HS\_CLK.hsclk\_lo = MAX\left\{\left\lfloor\left(\frac{t_{LOW\_MIN} + t_{FCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1, \quad \frac{t_{SCL}}{t_{I2C\_CLK}} - 1\right\}$$

Equation 13-6: Determining the I2Cn_HSCLK.hsclk_hi Register Value

$$I2Cn\_HS\_CLK.hsclk\_hi = \left\lfloor\left(\frac{t_{HIGH\_MIN} + t_{rCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1$$

Equation 13-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 13-5 *and* Equation 13-6

$$Calculated\ Frequency = \left((I2Cn\_HS\_CLK.hsclk\_hi + 1) + (I2Cn\_HS\_CLK.hsclk\_lo + 1)\right) * t_{I2C\_CLK}$$

Table 13-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 13-3: Calculated I²C Bus Clock Frequencies

f_{SYS_CLK} (MHz)	*I2Cn_HSCLK.hsclk_hi*	*I2Cn_HSCLK.hsclk_lo*	Calculated Frequency (MHz)
100	4	9	3.3

f_{SYS_CLK} (MHz)	I2Cn_HSCLK.hsclk_hi	I2Cn_HSCLK.hsclk_lo	Calculated Frequency (MHz)
50	2	4	3.125
25	1	2	2.5

13.4.5 Addressing

After a START condition the I²C slave address byte is transmitted by the hardware. The I²C slave address is composed of a slave address followed by a read/write bit.

Table 13-4: I²C Slave Address Format

Slave Address Bits		R/W Bit	Description
0000	000	0	General Call Address
0000	000	1	START Condition
0000	001	*x*	CBUS Address
0000	010	*x*	Reserved for different bus format
0000	011	*x*	Reserved for future purposes
0000	1*xx*	*x*	HS-mode master code
1111	1*xx*	*x*	Reserved for future purposes
1111	0*xx*	*x*	10-bit slave addressing

In 7-bit addressing mode, the master sends one address byte. To address a 7-bit address slave, first clear the *I2Cn_MSTCTRL.ex_addr_en* field to 0, then write the address to the transmit FIFO formatted as follows where A*n* is address A6:A0.

Master writing to slave: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 0]

Master reading from slave: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 1]

In 10-bit addressing mode (*I2Cn_MSTCTRL.ex_addr_en* = 1), the first byte the master sends is the 10-bit slave Addressing byte that includes the first two bits of the 10-bit address, followed by a 0 for the R/W bit. That is followed by a second byte representing the remainder of the 10-bit address. If the operation is a write, this is followed by data bytes to be written to the slave. If the operation is a read, it is followed by a repeated START. The software then writes the 10-bit address again with a 1 for the R/W bit. This I²C then starts receiving data from the slave device.

13.4.6 Master Mode Operation

The peripheral operates in master mode when master mode enable *I2Cn_CTRL.mst_mode* = 1. To initiate a transfer, the master generates a START condition by setting *I2Cn_MSTCTRL.start* = 1. If the bus is busy, it does not generate a START condition until the bus is available.

A master can communicate with multiple slave devices without relinquishing the bus. Instead of generating a STOP condition after communicating with the first slave, the master generates a Repeated START condition, or RESTART, by setting *I2Cn_MSTCTRL.restart* = 1. If a transaction is in progress, the peripheral finishes the transaction before generating a RESTART. The peripheral then transmits the slave address stored in the transmit FIFO. The *I2Cn_MSTCTRL.restart* bit is automatically cleared to 0 as soon as the master begins a RESTART condition.

I2Cn_MSTCTRL.start is automatically cleared to 0 after the master has completed a transaction and sent a STOP condition.

The master can also generate a STOP condition by setting *I2Cn_MSTCTRL.stop* = 1.

If both START and RESTART conditions are enabled at the same time, a START condition is generated first. Then, at the end of the first transaction, a RESTART condition is generated.

If both RESTART and STOP conditions are enabled at the same time, a STOP condition is not generated. Instead, a RESTART condition is generated. After the RESTART condition is generated, both bits are cleared.

If START, RESTART, and STOP are all enabled at the same time, a START condition is first generated. At the end of the first transaction, a RESTART condition is generated. The *I2Cn_MSTCTRL.stop* bit is cleared and ignored.

A slave cannot generate START, RESTART, or STOP conditions. Therefore, when master mode is disabled, the *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* bits are all cleared to 0.

For master mode operation, the following registers should only be configured when either:

1. The I^2C peripheral is disabled,

   or
2. The I^2C bus is guaranteed to be idle/free.

If this peripheral is the only master on the bus, then changing the registers outside of a transaction (*I2Cn_MSTCTRL.start* = 0) satisfies this requirement:

- *I2Cn_CTRL.mst_mode*
- *I2Cn_CTRL.irxm_en*
- *I2Cn_CTRL.one_mst_mode*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL1.cnt*
- *I2Cn_MSTCTRL.ex_addr_en*
- *I2Cn_MSTCTRL.mcode*
- *I2Cn_CLKLO.lo*
- *I2Cn_CLKHI.hi*
- *I2Cn_HSCLK.hsclk_lo*
- *I2Cn_HSCLK.hsclk_hi*

In contrast to the above set of register fields, the register fields below can be safely (re)programmed at any time:

- All interrupt flags and interrupt enable bits
- *I2Cn_TXCTRL0.thd_val*
- *I2Cn_RXCTRL0.thd_lvl*
- *I2Cn_TIMEOUT*.scl_to_val
- *I2Cn_DMA.rx_en*
- *I2Cn_DMA.tx_en*
- *I2Cn_FIFO.data*
- *I2Cn_MSTCTRL.start*
- *I2Cn_MSTCTRL.restart*
- *I2Cn_MSTCTRL.stop*

13.4.6.1 I²C Master Mode Receiver Operation

When in master mode, initiating a master receiver operation begins with the following sequence:

1. Write the number of data bytes to receive to the I²C receive count field (*I2Cn_RXCTRL1.cnt*).
2. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 1.
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The slave address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The I²C controller receives data from the slave and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RXCTRL1.cnt* data bytes have been received, the I²C controller sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag (*I2Cn_INTFL0.done* = 1).
8. If *I2Cn_MSTCTRL.restart* or *I2Cn_M.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

13.4.6.2 I²C Master Mode Transmitter Operation

When in master mode, initiating a master transmitter operation begins with the following sequence:

1. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 0.
2. Write the desired data bytes to the *I2Cn_FIFO* register, up to the size of the transmit FIFO. (e.g., If the transmit FIFO size is 8 bytes, the software may write one address byte and seven data bytes prior to starting the transaction.)
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The controller transmits the slave address byte written to the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The *I2Cn_FIFO* register data bytes are transmitted on the SDA line.
   a. The I²C controller receives an ACK from the slave after each data byte.
   b. As the transfer proceeds, the software should refill the transmit FIFO by writing to the *I2Cn_FIFO* register as needed.
   c. If the transmit FIFO goes empty during this process, the controller pauses at the beginning of the byte and waits for the software to either write more data or instruct the controller to send a RESTART or STOP condition.
7. Once the software writes all the desired bytes to the *I2Cn_FIFO* register, the software should set either *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INTFL0.done* and proceeds to send out either a RESTART condition, if *I2Cn_MSTCTRL.restart* was set, or a STOP condition, if *I2Cn_MSTCTRL.stop* was set.

13.4.6.3 I²C Multi-Master Operation

The I²C protocol supports multiple masters on the same bus. When the bus is free, it is possible that two (or more) masters might try to initiate communication at the same time. This is a valid bus condition. If this occurs and the two masters want to transmit different data and/or address different slaves, only one master can remain in master mode and complete its transaction. The other master must back off transmission and wait until the bus is idle. This process by which the winning master is determined is called bus arbitration.

To determine which master wins the arbitration, for each address or data bit, the master compares the data being transmitted on SDA to the value observed on SDA. If a master attempts to transmit a 1 on SDA (that is, the master lets SDA float) but senses a 0 instead, then that master loses arbitration, and the other master that sent a zero continues with the transaction. The losing master cedes the bus by switching off its SDA and SCL drivers.

Note: This arbitration scheme works with any number of bus masters: if more than two masters begin transmitting simultaneously, the arbitration continues as each master cedes the bus until only one master remains transmitting. Data is not corrupted because as soon as each master realizes it has lost arbitration, it stops transmitting on SDA, leaving the following data bits sent on SDA intact.

If the I²C master peripheral detects it has lost arbitration, it stops generating SCL; sets *I2Cn_INTFL0.areri*; sets *I2Cn_INTFL0.tx_lockout*, flushing any remaining data in the transmit FIFO; and clears *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* to 0. So long as the peripheral is not itself addressed by the winning master, the I²C peripheral stays in master mode (*I2Cn_CTRL.mst_mode = 1)*. If *at any time* another master addresses this peripheral using the address programmed in I2Cn_SLAVE.*addr*, then the I²C peripheral clears *I2Cn_CTRL.mst_mode* to 0 and begins responding as a slave. This can even occur during the same address transmission during which the peripheral lost arbitration.

Note: Arbitration loss is considered an error condition, and like the other error conditions sets I2Cn_INTFL0.tx_lockout. Therefore, after an arbitration loss, the software needs to clear I2Cn_INTFL0.tx_lockout and reload the transmit FIFO.

Also, in a multi-master environment, the software does *not* need to wait for the bus to become free before attempting to start a transaction (writing 1 to *I2Cn_MSTCTRL.start*). If the bus is free when *I2Cn_MSTCTRL.start* is set to 1, the transaction begins immediately. If instead the bus is busy, then the peripheral will:

1. Wait for the other master to complete the transaction(s) by sending a STOP,
2. Count out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see *Equation 13-3), and then*
3. Send a START condition and begin transmitting the slave address byte(s) in the transmit FIFO, followed by the rest of the transfer.

The I²C master peripheral is compliant with all bus arbitration and clock synchronization requirements of the I²C specification; this operation is automatic, and no additional programming is required.

13.4.7 Slave Mode Operation

When in slave mode, the I²Cn peripheral operates as a slave device on the I²C bus and responds to an external master's requests to transmit or receive data. To configure the I²Cn peripheral as a slave, write the *I2Cn_CTRL.mst_mode* bit to zero. The I²Cn clock is driven by the master on the bus, so the SCL device pin is driven by the external master and *I2Cn_STATUS.mst_busy* remains a zero. The desired slave address must be set by writing to the *I2Cn_SLAVE.addr* register.

For slave mode operation, the following register fields should be configured with the I2Cn peripheral disabled:

- *I2Cn_CTRL.mst_mode* = 0 for slave operation.
- *I2Cn_CTRL.gc_addr_en*
- *I2Cn_CTRL.irxm_en*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with slave mode operation with clock stretching disabled (I2Cn_CTRL.clkstr_dis = 1).*
- *I2Cn_CTRL.clkstr_dis*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TXCTRL0.nack_flush_dis*
- *I2Cn_TXCTRL0.rd_addr_flush_dis*
- *I2Cn_TXCTRL0.wr_addr_flush_dis*
- *I2Cn_TXCTRL0.gc_addr_flush_dis*
- *I2Cn_TXCTRL0.preload_mode*
  - Recommended value is 0 for applications that can tolerate slave clock stretching (*I2Cn_CTRL.clkstr_dis* = 0).
  - Recommend value is 1 for applications that do not allow slave clock stretching (*I2Cn_CTRL.clkstr_dis* = 1).
- *I2Cn_CLKHI.hi*
  - Applies to slave mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 13-2* is >= $t_{SU;DAT(min)}$.
- *I2Cn_HSCLK.hsclk_hi*
  - Applies to slave mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 13-6* is >= $t_{SU;DAT(min)}$.
- I2Cn_SLAVE.*addr*
- *I2Cn_SLAVE.ext_addr_en*

In contrast to the above register fields, the following register fields can be safely (re)programmed at any time:

- All interrupt flags and interrupt enables.
- *I2Cn_TXCTRL0.thd_val* and *I2Cn_RXCTRL0.thd_lvl*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TXCTRL1.tx_rdy*
  - Transmit ready (can only be cleared by hardware).
- *I2Cn_TIMEOUT*.scl_to_val
  - Timeout control.
- *I2Cn_DMA.rx_en* and *I2Cn_DMA.tx_en*
  - Transmit and receive DMA enables.
- *I2Cn_FIFO.data*
  - FIFO access register.

13.4.7.1 Slave Transmitter

The device operates as a slave transmitter when the received address matches the device slave address with the R/W bit set to 1. The master is then reading from the device slave. There two main modes of slave transmitter operation: just-in-time mode and preload mode.

In just-in-time mode, the software waits to write the transmit data to the transmit FIFO until after the master addresses it for a READ transaction, “just in time” for the data to be sent to the master. This allows the software to defer the determination of what data should be sent until the time of the address match. As an example, the transmit data could be based off an immediately preceding I^2C write transaction that requests a certain block of data to be sent, or the data could

represent the latest, most up-to-date value of a sensor reading. Clock stretching *must* be enabled (*I2Cn_CTRL*.*clkstr_dis* = 0) for just-in-time mode operation.

Program flow for transmit operation in just-in-time mode is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL*. *clkstr_dis* = 0, *I2Cn_TXCTRL0*[5:2] = 0x8 and *I2Cn_TXCTRL0*.*preload_mode* = 0. Don't forget to program *I2Cn_CLKHI*.*hi* and *I2Cn_HSCLK*.*hsclk_hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL*.*en* = 1.
    a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral responds to its address with an ACK.
    b. When the address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*.
3. The software waits for *I2Cn_INTFL0*.*addr_match* =1, either through polling the interrupt flag or setting *I2Cn_INTEN0*.*addr_match* to interrupt the CPU.
4. After reading *I2Cn_INTFL0*.*addr_match* =1, the software reads *I2Cn_CTRL*.*read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume read = 1, indicating transmit.
    a. The hardware holds SCL low until the software clears *I2Cn_INTFL0*.*tx_lockout* and loads data into the FIFO.
5. The software clears *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*. Now that *I2Cn_INTFL0*.*tx_lockout* is 0, the software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, the hardware releases SCL (after counting out *I2Cn_CLKHI*.*hi*) and sends out the data on the bus.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*ddone* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
    a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TXCTRL0*.*thd_val* and setting the *I2Cn_INTEN0*.*tx_thd* interrupt.
    b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The master ends the transaction by sending a NACK. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
    a. If the software needs to know how many data bytes were transmitted to the master, it should check the transmit FIFO level as soon as *I2Cn_INTFL0*.*done* = 1 and use it to determine how many data bytes were successfully sent.
        1) *Note: Any data remaining in the transmit FIFO is discarded prior to the next transmit operation; it is NOT necessary for the software to manually flush the transmit FIFO.*
9. The transaction is complete, the software should clear the *I2Cn_INTFL0*.*done* interrupt flag, and clear the *I2Cn_INTFL0*.*tx_thd* interrupt flag. Return to step 3, waiting on an address match.

The other mode of operation for slave transmit is preload mode. In this mode, it is assumed that the application firmware knows prior to the transmit operation what data it should send to the master. This data is then “preloaded” into the transmit FIFO. Once the address match occurs, this data can be sent out without any software intervention. Preload mode can be used with clock stretching either enabled or disabled, but it is the only option if clock stretching must be disabled.

To use slave transmit preload mode:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL*.*cl_clk_stretch_dis* = 1, *I2Cn_TXCTRL0*[5:2] = 0xF and *I2Cn_TXCTRL0*.*preload_mode* = 1.
2. The software sets *I2Cn_CTRL*.*en* = 1.
    a. Even though the controller is enabled, at this point it does not ACK an address match with R/W = 1 until the software sets *I2Cn_TXCTRL1*.*preload_rdy* = 1.

3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TXCTRL0.thd_val* and setting *I2Cn_INTEN0.tx_thd* interrupt, etc.
   a. If clock stretching is disabled, an empty transmit FIFO during the transmit operation causes a transmit underrun error. Therefore, the software should take any necessary steps to avoid an underrun *prior* to setting *I2Cn_TXCTRL1.preload_rdy* = 1.
   b. If clock stretching is enabled, then an empty transmit FIFO does not cause a transmit underrun error. However, it is recommended to follow the same preparation steps to minimize the amount of time spent clock stretching, which lets the transaction complete as quickly as possible.
4. Once the software has prepared for the transmit operation, it sets *I2Cn_TXCTRL1.preload_rdy* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. The hardware sets *I2Cn_INTFL0.addr_match* when an address match occurs. *I2Cn_INTFL0.tx_lockout* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INTFL0.addr_match* = 1, either through polling the interrupt flag or setting *I2Cn_INTEN0.amie* to interrupt the CPU.
6. After seeing *I2Cn_INTFL0.addr_match* =1, the software reads *I2Cn_CTRL.read* to determine if the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume *I2Cn_CTRL.read*, indicating a transmit.
   a. The hardware begins sending out the data that was preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TXCTRL1.preload_rdy* to 0.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INTFL0.done* remains 0 and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TXCTRL0.thd_val* and setting *I2Cn_INTEN0.tx_thd* interrupt.
   b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INTFL1.tx_un* = 1 and sends 0xFF for all following data bytes requested by the master.
8. The master ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INTFL0.done* interrupt flag.
   a. If the transmit FIFO empties at the same time that the master indicates the transaction is complete by sending a NACK, this is not considered an underrun event, and the *I2Cn_INTFL1.tx_un* flag remains 0.
   b. If the software needs to know how many data bytes were transmitted to the master, check the transmit FIFO level when the *I2Cn_INTFL0.done* flag is set to 1.
9. The transaction is complete, the software should "clean up", which should include clearing *I2Cn_INTFL0.done*. Return to step 3 and prepare for the next transaction.
   a. Any data remaining in the transmit FIFO is not discarded, it is reused for the next transmit operation.
      1) If this is not desired, the software can flush the transmit FIFO. Flush the transmit and receive FIFOs by writing 0 to *I2Cn_CTRL.en* and the writing 1 to *I2Cn_CTRL.en*.

Once a slave starts transmitting from the *I2Cn_FIFO*, detection of out of sequence STOP, START, or RESTART conditions terminate the current transaction. When a transaction is terminated as a result of an out of sequence error, *I2Cn_INTFL0.start_err* or *I2Cn_INTFL0.stop_err* is set to 1.

If the transmit FIFO is not ready (*I2Cn_TXCTRL1.preload_rdy* = 0) and the I²C controller receives a data read request from the master, the hardware automatically sends a NACK at the end of the first address byte. The setting of the do not respond field is ignored by the hardware in this case because the only opportunity to send a NACK for an I²C read transaction is after the address byte.

13.4.7.2 Slave Receivers

The device operates as a slave receiver when the received address matches the device slave address with the R/W bit set to 0. The external master is writing to the slave.

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers.
2. Set *I2Cn_CTRL*.*en* = 1.
   a. If an address match with R/W=0 occurs, and the receive FIFO is empty, the peripheral responds with an ACK and the *I2Cn_INTFL0*.*addr_match* flag is set.
   b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RXCTRL0*.*dnr*, the peripheral NACKs either the address byte (*I2Cn_RXCTRL0*.*dnr* = 1) or the first data byte (*I2Cn_RXCTRL0*.*dnr* = 0).
3. Wait for *I2Cn_INTFL0*.*addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* = 1.
4. Read *I2Cn_CTRL*.*read* to determine if the transaction is a transmit (*I2Cn_CTRL*.*read* = 1) or a receive (*I2Cn_CTRL*.*read* = 0) operation. In this case, assume *I2Cn_CTRL*.*read* = 0, indicating receive. The device begins receiving data into the receive FIFO.
5. Clear *I2Cn_INTFL0*.*addr_match*, and while the master keeps sending data, *I2Cn_INTFL0*.*done* remains 0 and the software should continue to monitor the receive FIFO and empty it as needed.
   a. The FIFO level can be monitored synchronously through the receive FIFO status/interrupt flags, or asynchronously by setting *I2Cn_RXCTRL0*.*thd_lvl* and enabling the *I2Cn_INTFL0*.*rx_thd* interrupt.
   b. If the receive FIFO ever fills up during the transaction, then the hardware sets *I2Cn_INTFL1*.*rx_ov* and then either:
      i. If *I2Cn_CTRL*.*clkstr_dis* = 0, start clock stretching and wait for software to read from the receive FIFO, *or*
      ii. If *I2Cn_CTRL*.*clkstr_dis* = 1, respond to the master with a NACK and the last byte is discarded.
6. The master ends the transaction by sending a RESTART or STOP. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the receive FIFO.
7. Once a slave starts receiving into its receive FIFO, detection of an out of sequence STOP, START, or RESTART condition releases the I²C bus to the Idle state and the hardware sets the *I2Cn_INTFL0*.*start_err* field or *I2Cn_INTFL0*.*stop_err* field to 1 based on the specific condition.

If software has not emptied the data in the receive FIFO from the previous transaction by the time that a master addresses it for another write (i.e., receive) transaction, then the controller will *not* participate in the transaction, and no additional data is written into the FIFO. Although a NACK *is* sent to the master, the software can control if the NACK is sent with the initial address match, or if instead it is sent at the end of the first data byte. Setting *I2Cn_RXCTRL0*.*dnr* to 1 chooses the former, while setting *I2Cn_RXCTRL0*.*dnr* to 0 chooses the latter.

13.4.8 Interrupt Sources

The I²C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I²C interrupt, the software determines the cause of the interrupt by reading the I²C interrupt flags registers *I2Cn_INTFL0* and *I2Cn_INTFL1*. Interrupts can be generated for the following events:

- Transaction Complete (master/slave).
- Address NACK received from slave (master).
- Data NACK received from slave (master).
- Lost arbitration (master).
- Transaction timeout (master/slave).
- FIFO is empty, not empty, full to configurable threshold level (master/slave).
- Transmit FIFO locked out because it is being flushed (master/slave)
- Out of sequence START and STOP conditions (master/slave).
- Sent a NACK to an external master because the transmit or receive FIFO was not ready (slave).
- Address ACK or NACK received (master).
- Incoming address match (slave).
- Transmit Underflow or receive Overflow (slave).

Interrupts for each event can be enabled or disabled by setting or clearing the corresponding bit in the *I2Cn_INTEN0* or *I2Cn_INTEN1* interrupt enable register.

Note: Disabling the interrupt does not prevent the corresponding flag from being set by the hardware but does prevent an IRQ when the interrupt flag is set.

Note: Prior to enabling an interrupt, the status of the corresponding interrupt flag should be checked and, if necessary, serviced or cleared. This prevents a previous interrupt event from interfering with a new I^2C communications session.

13.4.9 Transmit FIFO and Receive FIFO

There are separate transmit and receive FIFOs. Both are accessed using the FIFO data register *I2Cn_FIFO*. Writes to this register enqueue data into the transmit FIFO. Writes to a full transmit FIFO have no effect. Reads from *I2Cn_FIFO* dequeue data from the receive FIFO. Writes to a full transmit FIFO have no effect and reads from an empty receive FIFO return 0xFF.

The transmit and receive FIFO only read or write one byte at a time. Transactions larger than 8 bits can still be performed, however. A 16- or 32-bit write to the transmit FIFO stores just the lowest 8 bits of the write data. A 16- or 32-bit read from the receive FIFO has the valid data in the lowest 8 bits and 0's in the upper bits. In any case, the transmit and receive FIFOs only accept 8 bits at a time for either reads or writes.

To offload work from the CPU, the DMA can read and write to each FIFO. See *DMA Control* for more information on configuring the DMA.

During a receive transaction (which during master operation is a READ, and during slave operation is a WRITE), received bytes are automatically written to the receive FIFO. The software should monitor the receive FIFO level and unload data from it as needed by reading *I2Cn_FIFO*. If the receive FIFO becomes full during a master mode transaction, then the hardware sets the *I2Cn_INTFL1.rx_ov* the *I2Cn_INTFL1.rx_ov* bit and one of two things happen depending on the value of *I2Cn_CTRL.clkstr_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0), then the hardware stretches the clock until the software makes space available in the receive FIFO by reading from *I2Cn_FIFO*. Once space is available, the hardware moves the data byte from the shift register into the receive FIFO, the SCL device pin is released, and the master is free to continue the transaction.
- If clock stretching is disabled (*I2Cn_CTRL.clkstr_dis* = 1), then the hardware responds to the master with a NACK and the data byte is lost. The master can return the bus to idle with a STOP condition or start a new transaction with a RESTART condition.

During a transmit transaction (which during master operation is a WRITE, and during slave operation is a READ), either the software or the DMA can provide data to be transmitted by writing to the transmit FIFO. Once the peripheral finishes transmitting each byte, it removes it from the transmit FIFO and, if available, begins transmitting the next byte.

Interrupts can be generated for the following FIFO status:

- Transmit FIFO level less than or equal to threshold.
- Receive FIFO level greater than or equal to threshold.
- Transmit FIFO underflow.
- Receive FIFO overflow.
- Transmit FIFO locked for writing.

Both the receive FIFO and transmit FIFO are flushed when the I^2Cn port is disabled by clearing *I2Cn_CTRL.en*=0. While the peripheral is disabled, writes to the transmit FIFO have no effect and reads from the receive FIFO return 0xFF.

The transmit FIFO and receive FIFO can be flushed by setting the transmit FIFO flush bit (*I2Cn_TXCTRL0.flush*=1) or the receive FIFO flush bit (*I2Cn_RXCTRL0.flush*=1), respectively. In addition, under certain conditions, the transmit FIFO is

automatically locked by the hardware and flushed so stale data is not unintentionally transmitted. The transmit FIFO is automatically flushed, and writes locked out from the software under the following conditions:

- General Call Address Match: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.gc_addr_flush_dis*.
- Slave Address Match Write: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.wr_addr_flush_dis*.
- Slave Address Match Read: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.rd_addr_flush_dis*.
- During operation as a slave transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.nack_flush_dis*.
- Any of the following interrupts: arbitration error, timeout error, master mode address NACK error, master mode data NACK error, start error, and stop error. Automatic flushing cannot be disabled for these conditions.

When the above conditions occur, the transmit FIFO is flushed so that data intended for a previous transaction is not transmitted unintentionally for a new transaction. In addition to flushing the transmit FIFO, the transmit lockout flag is set (*I2Cn_INTFL0.tx_lockout* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INTFL0.tx_lockout*.

13.4.10 Transmit FIFO Preloading

There may be situations during slave mode operation where the software wants to preload the transmit FIFO prior to a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external master requesting data with an ACK and clock stretching while the software writes the data to the transmit FIFO, the hardware responds with a NACK until the software has preloaded the requested data into the transmit FIFO.

When transmit FIFO preloading is enabled, the software controls ACKs to the external master using the transmit ready (*I2Cn_TXCTRL1.preload_rdy*) bit. When *I2Cn_TXCTRL1.preload_rdy* is set to 0, the hardware automatically NACKs all read transactions from the master. Setting *I2Cn_TXCTRL1.preload_rdy* to 1 sends an ACK to the master on the next read transaction and transmits the data in the transmit FIFO. Preloading the transmit FIFO should be complete prior to setting the *I2Cn_TXCTRL1.preload_rdy* field to 1.

The required steps for implementing transmit FIFO preloading in software are as follows:

1. Enable the transmit FIFO preloading by setting the field *I2Cn_TXCTRL0.preload_mode* to 1. The hardware automatically clears the *I2Cn_TXCTRL1.preload_rdy* field to 0.
2. If the transmit FIFO lockout flag (*I2Cn_INTFL0.tx_lockout*) is set to 1, write 1 to clear the flag and enable writes to the transmit FIFO.
3. Enable DMA or interrupts if required.
4. Load the transmit FIFO with the data to send when the master sends the next read request.
5. Set *I2Cn_TXCTRL1.preload_rdy* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a master.
6. *I2Cn_TXCTRL1.preload_rdy* is cleared by the hardware once it finishes transmitting the first byte, and data is transmitted from the transmit FIFO. Once cleared, the application firmware may repeat the preloading process or disable transmit FIFO preloading.

Note: To prevent the preloaded data from being cleared when the master tries to read it, the software must at least set I2Cn_TXCTRL0.rd_addr_flush_dis to 1, disabling auto flush on READ address match. The software determines if the other auto flush disable bits should be set. For example, if a master uses I^2C WRITE transactions to determine what data the slave should send in the following READ transactions, then the software can clear I2Cn_TXCTRL0.wr_addr_flush_dis to 0. Then when a WRITE occurs, the transmit FIFO is flushed, giving the software time to load the new data. For the READ transaction, the external master can poll the slave address until the new data has been loaded and I2Cn_TXCTRL1.preload_rdy is set, at which point the peripheral responds with an ACK.

13.4.11 Interactive Receive Mode (IRXM)

In some situations, the I2Cn might want to inspect and respond to each byte of received data. In this case, interactive receive mode (IRXM) can be used. IRXM is enabled by setting *I2Cn_CTRL.irxm_en* = 1. If IRXM is enabled, it must occur before any I^2C transfer is initiated.

When IRXM is enabled, after every data byte received, the I2Cn peripheral automatically holds SCL low before the ACK bit. Additionally, after the 8th SCL falling edge, the I2Cn peripheral sets the IRXM interrupt status flag (*I2Cn_INTFL0.irxm* = 1). Application firmware must read the data and generate a response (ACK or NACK) by setting the IRXM Acknowledge (*I2Cn_CTRL.irxm_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL.irxm_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL.irxm_ack* bit to 1.

After setting the *I2Cn_CTRL.irxm_ack* bit, clear the IRXM interrupt flag. Write 1 to *I2Cn_INTFL0.irxm* to clear the interrupt flag. When the IRXM interrupt flag is cleared, the I2Cn peripheral hardware releases the SCL line and sends the *I2Cn_CTRL.irxm_ack* on the SDA line.

While the I2Cn peripheral is waiting for the application firmware to clear the *I2Cn_INTFL0.irxm* flag, the software can disable IRXM and, if operating as a master, load the remaining number of bytes to be received for the transaction. This allows the software to examine the initial bytes of a transaction, which might be a command, and then disable IRXM to receive the remaining bytes in normal operation.

During IRXM, received data is not placed in the receive FIFO. Instead, the *I2Cn_FIFO* address is repurposed to directly read the receive shift register, bypassing the receive FIFO. Therefore, before disabling IRXM, the software must first read the data byte from *I2Cn_FIFO.data.* If the IRXM byte is not read, the byte is lost and the next read from the receive FIFO returns 0xFF.

Note: IRXM only applies to data bytes and does not apply to address bytes, general call address responses or START byte responses.

Note: When enabling IRXM and operating as a slave, clock stretching must remain enabled (I2Cn_CTRL.clkstr_dis = 0).

13.4.12 Clock Stretching

When the I2Cn peripheral requires some response or intervention from the software to continue with a transaction, it holds SCL low, preventing the transfer from continuing. This is called 'clock stretching' or 'stretching the clock'. While the I^2C Bus Specification defines the term 'clock stretching' to only apply to a slave device holding the SCL line low, this section describes situations where the I2Cn peripheral holds the SCL line low in either slave *or* master mode, and refers to *both* as clock stretching.

When the I2Cn peripheral stretches the clock, it typically does so in response to either a full receive FIFO during a receive operation, or an empty transmit FIFO during a transmit operation. Necessarily, this occurs before the next data byte begins, either between the ACK bit and the first data bit or, if at the beginning of a transaction, immediately after a START or RESTART condition. However, when operating in IRXM (*I2Cn_CTRL.irxm_en* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing the software to decide if to send an ACK or NACK.

For a transmit operation (as either master or slave), when the transmit FIFO is empty, SCL is automatically held low after the ACK bit and before the next data byte begins. To stop clock stretching and continue the transaction, the software must write data to *I2Cn_FIFO.data*. If operating in master mode, however, instead of sending more data, the software may also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.restart* to send a STOP or RESTART condition, respectively.

For a receive operation (as either master or slave), when both the receive FIFO and the receive shift register are full, SCL is automatically held low until at least one data byte is read from the receive FIFO. To stop clock stretching and continue the transaction, the software must read data from *I2Cn_FIFO.data*. If operating in master mode and this is the final byte of the transaction, as determined by *I2Cn_RXCTRL1.cnt*, then the software must also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.restart* to send a STOP or RESTART condition, respectively. This must be done in addition to reading from the receive FIFO, since the peripheral cannot start sending the STOP or RESTART until the last data byte has been moved from the receive shift register into the receive FIFO. This occurs automatically once there is space in the receive FIFO.

Note: Since some masters do not support other devices stretching the clock, it is possible to completely disable all clock stretching during slave mode by setting I2Cn_CTRL.clkstr_dis to 1 and clearing I2Cn_CTRL.irxm_en to 0. In this case, instead of clock stretching the I2Cn peripheral sends a NACK if receiving data or sends 0xFF if transmitting data.

Note: The clock synchronization required to support other I2C master or slave devices stretching the clock is built into the peripheral and requires no intervention from the software to operate correctly.

13.4.13 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 13-8* and *Equation 13-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 13-8: I²C Timeout Maximum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 3\right)$$

Due to clock synchronization, the timeout is guaranteed to meet the following minimum time calculation shown in *Equation 13-9*.

Equation 13-9: I²C Timeout Minimum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 2\right)$$

The timeout feature is disabled when *I2Cn_TIMEOUT.scl_to_val* = 0 and is enabled for any non-zero value. When the timeout is enabled, the timeout timer starts counting when the I2Cn peripheral hardware drives SCL low and is reset by the I2Cn peripheral hardware when the SCL line is released.

The timeout counter only monitors if the I2Cn peripheral hardware is driving the SCL line low. It does not monitor if an external I2Cn device is actively holding the SCL line low. The timeout counter also does not monitor the status of the SDA line.

If the timeout timer expires, a bus error condition has occurred. When a timeout error occurs, the I2Cn peripheral hardware releases the SCL and SDA lines, and sets the timeout error interrupt flag to 1 (*I2Cn_INTFL0.to_err* = 1).

For applications where the device may hold the SCL line low longer than the maximum timeout supported, the timeout can be disabled by setting the timeout field to 0 (*I2Cn_TIMEOUT.scl_to_val* = 0).

13.4.14 DMA Control

There are independent DMA channels for each transmit FIFO and each receive FIFO. DMA activity is triggered by the transmit FIFO (*I2Cn_TXCTRL0.thd_val*) and receive FIFO (*I2Cn_RXCTRL0.thd_lvl*) threshold levels.

When the transmit FIFO byte count (*I2Cn_TXCTRL1.lvl*) is less than or equal to the transmit FIFO threshold level *I2Cn_TXCTRL0.thd_val*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

To ensure the DMA does not overflow the transmit FIFO, the DMA burst size should be set as follows:

Equation 13-10: DMA Burst Size Calculation for I²C Transmit

$$DMA\ Burst\ Size\ \leq TX\ FIFO\ Depth - I2Cn\_TXCTRL0.thd\_val = 8 - I2Cn\_TXCTRL0.thd\_val$$

$$where\ 0\ \leq I2Cn\_TXCTRL0.thd\_val\ \leq 7$$

Applications trying to avoid transmit underflow and/or clock stretching should use a smaller burst size and higher *I2Cn_TXCTRL0.thd_val* setting. This fills up the FIFO more frequently but increases internal bus traffic.

When the receive FIFO count (*I2Cn_RXCTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RXCTRL0.thd_lvl*, the DMA transfers data out of the receive FIFO according to the DMA configuration. To ensure the DMA does not underflow the receive FIFO, the DMA burst size should be set as follows:

Equation 13-11: DMA Burst Size Calculation for I²C Receive

$$DMA\ Burst\ Size \leq I2Cn\_RXCTRL0.thd\_lvl$$

$$where\ 1 \leq I2Cn\_RXCTRL0.thd\_lvl \leq 8$$

Applications trying to avoid receive overflow and/or clock stretching should use a smaller burst size and lower *I2Cn_RXCTRL0.thd_lvl*. This results in reading from the Receive FIFO more frequently but increases internal bus traffic.

Note for receive operations, the length of the DMA transaction (in bytes) must be an integer multiple of I2Cn_RXCTRL0.thd_lvl. Otherwise, the receive transaction ends with some data still in the receive FIFO, but not enough to trigger an interrupt to the DMA, leaving the DMA transaction incomplete. One easy way to ensure this for all transaction lengths is to set burst size to 1 (I2Cn_RXCTRL0.thd_lvl = 1).

To enable DMA transfers, enable the transmit DMA channel (*I2Cn_DMA.tx_en*) and/or the receive DMA channel (*I2Cn_DMA.rx_en*).

13.5 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 13-1*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 13-5: Register Summary

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	I²C Control Register
[0x0004]	*I2Cn_STATUS*	I²C Status Register
[0x0008]	*I2Cn_INTFL0*	I²C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INTEN0*	I²C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INTFL1*	I²C Interrupt Flags 1 Register
[0x0014]	*I2Cn_INTEN1*	I²C Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFOLEN*	I²C FIFO Length Register
[0x001C]	*I2Cn_RXCTRL0*	I²C Receive Control 0 Register
[0x0020]	*I2Cn_RXCTRL1*	I²C Receive Control 1 Register
[0x0024]	*I2Cn_TXCTRL0*	I²C Transmit Control 0 Register
[0x0028]	*I2Cn_TXCTRL1*	I²C Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	I²C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MSTCTRL*	I²C Master Control Register
[0x0034]	*I2Cn_CLKLO*	I²C Clock Low Time Register
[0x0038]	*I2Cn_CLKHI*	I²C Clock High Time Register
(0x003C)	*I2Cn_HSCLK*	I²C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I²C Timeout Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable Register
[0x004C]	*I2Cn_SLAVE*	I²C Slave Address Register

13.5.1 Register Details

Table 13-6: I²C Control Register

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	hs_en	R/W	0	**Hs-Mode Enable** I²C high speed mode operation 0: Disable 1: Enable
14	-	RO	0	**Reserved**
13	one_mst_mode	R/W	0	**Single Master Only** When set to 1, the device MUST ONLY be used in a single master application with slave devices that are NOT going to hold SCL low (i.e., the slave devices will never clock stretch)
12	clkstr_dis	R/W	0	**Slave Mode Clock Stretching** 0: Enabled 1: Disabled
11	read	R	0	**Slave Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INTFL0.addr_match* = 1) or general call match (*I2Cn_INTFL0.gc_addr_match* = 1). This bit is valid three system clock cycles after the address match status flag is set.
10	bb_mode	R/W	0	**Software Output Control Enabled** Setting this field to 1 enables software bit-bang control of the I2Cn Bus. 0: The I2C controller manages the SDA and SCL pins in the hardware. 1: SDA and SCL are controller by the software using the *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out* fields.
9	sda	R	-	**SDA Status** 0: SDA pin is logic low. 1: SDA pin is logic high.
8	scl	R	-	**SCL Status** 0: SCL pin is logic low. 1: SCL pin is logic high.
7	sda_out	R/W	0	**SDA Pin Output Control** Set the state of the SDA hardware pin (actively pull low or float). 0: Pull SDA Low 1: Release SDA *Note: Only valid when I2Cn_CTRL.bb_mode=1*
6	scl_out	R/W	0	**SCL Pin Output Control** Set the state of the SCL hardware pin (actively pull low or float). 0: Pull SCL low 1: Release SCL *Note: Only valid when I2Cn_CTRL.bb_mode =1*
5	-	RO	0	**Reserved**
4	irxm_ack	R/W	0	**IRXM Acknowledge** If IRXM is enabled (*I2Cn_CTRL.rx_mode* = 1), this field determines if the hardware sends an ACK or a NACK to an IRXM transaction. 0: Respond to IRXM with ACK 1: Respond to IRXM with NACK

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
3	irxm_en	R/W	0	**IRXM Enable** When receiving data, allows for an IRXM interrupt event after each received byte of data. The I2Cn peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See *Interactive Receive Mode* section for detailed information. 0: Disable 1: Enable *Note: Only set this field when the I²C bus is inactive.*
2	gc_addr_en	R/W	0	**General Call Address Enable** 0: Ignore General Call Address 1: Acknowledge General Call Address
1	mst_mode	R/W	0	**Master Mode Enable** 0: Slave mode enabled. 1: Master mode enabled.
0	en	R/W	0	**I²C Peripheral Enable** 0: Disabled 1: Enabled

Table 13-7: I²C Status Register

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	mst_busy	RO	0	**Master Mode I²C Bus Transaction Active** The peripheral is operating in master mode and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as master and actively driving SCL clock cycles.
4	tx_full	RO	0	**Transmit FIFO Full** 0: Not full 1: Full
3	tx_em	RO	1	**Transmit FIFO Empty** 0: Not empty 1: Empty
2	rx_full	RO	0	**Receive FIFO Full** 0: Not full 1: Full
1	rx_em	RO	1	**Receive FIFO Empty** 0: Not empty 1: Empty
0	busy	RO	0	**Master or Slave Mode I²C Busy Transaction Active** The peripheral is operating in master or slave mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the peripheral acting as a master or an external master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: I²C bus is idle. 1: I²C bus transaction in progress.

Table 13-8: I²C Interrupt Flag 0 Register

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_add_match	R/W1C	0	**Slave Write Address Match Interrupt Flag** If set, the device has been accessed for a write (i.e., receive) transaction in slave mode and the address received matches the device slave address. 0: No address match. 1: Address match.
22	rd_addr_match	R/W1C	0	**Slave Read Address Match Interrupt Flag** If set, the device has been accessed for a read (i.e., transmit) transaction in slave mode and the address received matches the device slave address. 0: No address match. 1: Address match.
21:17	-	RO	0	**Reserved**
16	-	R/W1C	0	**MAMI Interrupt Flag**
15	tx_lockout	R/W1C	0	**Transmit FIFO Locked Interrupt Flag** If set, the transmit FIFO is locked and writes to the transmit FIFO are ignored. When set, the transmit FIFO is automatically flushed. Writes to the transmit FIFO are ignored until this flag is cleared. Write 1 to clear. 0: transmit FIFO not locked. 1: transmit FIFO is locked and all writes to the transmit FIFO are ignored.
14	stop_err	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.
13	start_err	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.
12	dnr_err	R/W1C	0	**Slave Mode Do Not Respond Interrupt Flag** This occurs if an address match is made, but the transmit FIFO or receive FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I²C address match has occurred and either the transmit or receive FIFO is not configured.
11	data_err	R/W1C	0	**Master Mode Data NACK from External Slave Interrupt Flag** This flag is set by the hardware if a NACK is received from a slave. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a slave.
10	addr_nack_err	R/W1C	0	**Master Mode Address NACK from Slave Error Flag** This flag is set by the hardware if an Address NACK is received from a slave bus. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a slave.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
9	to_err	R/ W1C	0	**Timeout Error Interrupt Flag** This occurs when this device holds SCL low longer than the programmed timeout value. Applies to both master and slave mode. Write 1 to clear. Write 0 has no effect. 0: Timeout error has not occurred. 1: Timeout error occurred.
8	arb_err	R/ W1C	0	**Master Mode Arbitration Lost Interrupt Flag** Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
7	addr_ack	R/ W1C	0	**Master Mode Address ACK from External Slave Interrupt Flag** This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The slave device ACK for the address was received.
6	stop	R/ W1C	0	**Slave Mode STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
5	tx_thd	RO	1	**Transmit FIFO Threshold Level Interrupt Flag** This field is set by the hardware if the number of bytes in the Transmit FIFO is less than or equal to the Transmit FIFO threshold level. Write 1 to clear. This field is automatically cleared by the hardware when the transmit FIFO contains fewer bytes than the transmit threshold level. 0: transmit FIFO contains more bytes than the transmit threshold level. 1: transmit FIFO contains transmit threshold level or fewer bytes (Default).
4	rx_thd	R/W1C	1	**Receive FIFO Threshold Level Interrupt Flag** This field is set by the hardware if the number of bytes in the Receive FIFO is greater than or equal to the Receive FIFO threshold level. This field is automatically cleared when the receive FIFO contains fewer bytes than the receive threshold setting. 0: receive FIFO contains fewer bytes than the receive threshold level. 1: receive FIFO contains at least receive threshold level of bytes (Default).
3	addr_match	R/W1C	0	**Slave Mode Incoming Address Match Status Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: Slave address match has not occurred. 1: Slave address match occurred.
2	gc_addr_match	R/W1C	0	**Slave Mode General Call Address Match Received Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: General call address match has not occurred. 1: General call address match occurred.
1	irxm	R/W1C	0	**Interactive Receive Mode Interrupt Flag** Write 1 to clear. Writing 0 is ignored. 0: Interrupt condition has not occurred. 1: Interrupt condition occurred.
0	done	R/W1C	0	**Transfer Complete Interrupt Flag** This flag is set for both master and slave mode once a transaction completes. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.

Table 13-9: I²C Interrupt Enable 0 Register

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W	0	**Slave Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode and the address received matches the device slave addressed for a write transaction. 0: Disabled. 1: Enabled.
22	rd_addr_match	R/W	0	**Slave Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode and the address received matches the device slave addressed for a read transaction. 0: Disabled. 1: Enabled.
21:17	-	RO	0	**Reserved**
16	mami	R/W	0	**MAMI Interrupt Enable**
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
14	stop_err	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
13	start_err	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
12	dnr_err	R/W	0	**Slave Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in slave mode when the “Do Not Respond” condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_err	R/W	0	**Master Mode Received Data NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.
10	addr_nack_err	R/W	0	**Master Mode Received Address NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.
9	to_err	R/W	0	**Timeout Error Interrupt Enable** 0: Disabled. 1: Enabled.
8	arb_err	R/W	0	**Master Mode Arbitration Lost Interrupt Enable** 0: Disabled. 1: Enabled.
7	addr_ack	R/W	0	**Received Address ACK from Slave Interrupt Enable** Set this field to enable interrupts for master mode slave device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
5	tx_thd	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
4	rx_thd	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
3	addr_match	R/W	0	**Slave Mode Incoming Address Match Interrupt Enable** 0: Disabled. 1: Enabled.
2	gc_addr_match	R/W	0	**Slave Mode General Call Address Match Received Interrupt Enable** 0: Disabled. 1: Enabled.
1	irxm	R/W	0	**Interactive Receive Interrupt Enable** 0: Disabled. 1: Enabled.
0	done	R/W	0	**Transfer Complete Interrupt Enable** 0: Disabled. 1: Enabled.

Table 13-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition has been detected. 0: START condition not detected. 1: START condition detected.
1	tx_un	R/W1C	0	**Slave Mode Transmit FIFO Underflow Status Flag** In slave mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the master requests more data by sending an ACK after the previous byte transferred. 0: Slave mode transmit FIFO underflow condition has not occurred. 1: Slave mode transmit FIFO underflow condition occurred.
0	rx_ov	R/W1C	0	**Slave Mode Receive FIFO Overflow Status Flag** In slave mode operation, the hardware sets this flag automatically when an receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Slave mode receive FIFO overflow event has not occurred. 1: Slave mode receive FIFO overflow condition occurred (data lost).

Table 13-11: I²C Interrupt Enable 1 Register

I²C Interrupt Enable 1			I2Cn_INTEN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W	0	**START Condition Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_un	R/W	0	**Slave Mode Transmit FIFO Underflow Interrupt Enable** 0: Disabled. 1: Enabled.

I²C Interrupt Enable 1			I2Cn_INTEN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
0	rx_ov	R/W	0	**Slave Mode Receive FIFO Overflow Interrupt Enable** 0: Disabled. 1: Enabled.

Table 13-12: I²C FIFO Length Register

I²C FIFO Length			I2Cn_FIFOLEN	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	tx_depth	RO	8	**Transmit FIFO Length** Returns the depth of the transmit FIFO. 8: 8-bytes
7:0	rx_depth	RO	8	**Receive FIFO Length** Returns the depth of the receive FIFO. 8: 8-bytes

Table 13-13: I²C Receive Control 0 Register

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	thd_lvl	R/W	0	**Receive FIFO Threshold Level** Set this field to the required number of bytes to trigger a receive FIFO threshold event. When the number of bytes in the receive FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INTFL0.rx_thd* bit indicating an receive FIFO threshold level event. 0: 0 bytes or more in the receive FIFO causes a threshold event. 1: 1+ bytes in the receive FIFO triggers a receive threshold event (recommended minimum value). … 8: Receive FIFO threshold event only occurs when the receive FIFO is full.
7	flush	R/W1O	0	**Flush Receive FIFO** Write 1 to this field to initiate a receive FIFO flush, clearing all data in the receive FIFO. This field is automatically cleared by the hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Receive FIFO flush complete or not active. 1: Flush the receive FIFO
6:1	-	RO	0	**Reserved**
0	dnr	R/W	0	**Slave Mode Do Not Respond** Slave mode operation only. If the device has been addressed for a write operation, and there is still data in the receive FIFO then: 0: Always respond to an address match with an ACK but then always respond to data bytes with a NACK. 1: NACK the address.

Table 13-14: I²C Receive Control 1 Register

I²C Receive Control 1			I2Cn_RXCTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**

I²C Receive Control 1			I2Cn_RXCTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	lvl	R	0	**Receive FIFO Byte Count Status** This field returns the number of bytes in the receive FIFO. 0: 0 bytes (No data) 1: 1 byte 2: 2 bytes 3: 3 bytes 4: 4 bytes 5: 5 bytes 6: 6 bytes 7: 7 bytes 8: 8 bytes
7:0	cnt	R/W	1	**Receive FIFO Transaction Byte Count Configuration** When in master mode, write the number of bytes to be received in a transaction from 1 to 256. 0x00 represents 256. 0: 256 byte receive transaction. 1: 1 byte receive transaction. 2: 2 byte receive transaction. … 255: 255 byte receive transaction. *This field is ignored when I2Cn_CTRL.irxm_en = 1. To receive more than 256 bytes, use I2Cn_CTRL.irxm_en = 1*

Table 13-15: I²C Transmit Control 0 Register

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	thd_val	R/W	0	**Transmit FIFO Threshold Level** Sets the level for a Transmit FIFO threshold event interrupt. If the number of bytes remaining in the transmit FIFO falls to this level or lower the interrupt flag *I2Cn_INTFL0.thd_val* is set indicating a transmit FIFO Threshold Event occurred. 0: 0 bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event. 1: 1 byte or fewer remaining in the transmit FIFO triggers a transmit FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event
7	flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: transmit FIFO flush is complete or not active. 1: Flush the transmit FIFO *Note: The hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INTFL0.tx_lockout* = 1, then *I2Cn_TXCTRL0.flush* = 1.
6	-	RO	0	**Reserved**

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
5	nack_flush_dis	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Received NACK at end of slave transmit operation enabled 1: Received NACK at end of slave transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 0).*
4	rd_addr_flush_dis	R/W	0	**Transmit FIFO Slave Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bitfield to 1).*
3	wr_addr_flush_dis	R/W	0	**Transmit FIFO Slave Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*
2	gc_addr_flush_dis	R/W	0	**Transmit FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: The hardware controls *I2Cn_TXCTRL1.preload_rdy.* 1: Software control of *I2Cn_TXCTRL1.preload_rdy.*
0	preload_mode	R/W	0	**Transmit FIFO Preload Mode Enable** 0: Normal operation. An address match in slave mode, or a general call address match, flushes and locks the transmit FIFO so it cannot be written and set *I2Cn_INTFL0.tx_lockout.* 1: Transmit FIFO preload mode. An address match in slave mode, or a general call address match, does not lock the transmit FIFO and does not set *I2Cn_INTFL0.tx_lockout*. This allows the software to preload data into the transmit FIFO. The status of the I²C is controllable at *I2Cn_TXCTRL1.preload_rdy.*

Table 13-16: I²C Transmit Control 1 Register

I²C Transmit Control Register 1			I2Cn_TXCTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	lvl	R	0	**Transmit FIFO Byte Count Status** 0: 0 bytes (No data) 1: 1 byte 2: 2 bytes 3: 3 bytes 4: 4 bytes 5: 5 bytes 6: 6 bytes 7: 7 bytes 8: 8 bytes (max value)
7:1	-	RO	0	**Reserved**
0	preload_rdy	R/W1O	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a slave address match a NACK is sent. Once the I2Cn hardware is ready (the software has preloaded the transmit FIFO, configured the DMA, etc.), the software must set this bit to 1 so the I2Cn hardware sends an ACK on a slave address match. When transmit FIFO preload mode is disabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is forced to 1 and the I2Cn hardware behaves normally.

Table 13-17: I²C Data Register

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	R/W	0xFF	**FIFO Data** Reads from this register pop data off the receive FIFO. Writes to this register push data onto the transmit FIFO. Reading from an empty receive FIFO returns 0xFF. Writes to a full transmit FIFO are ignored.

Table 13-18: I²C Master Control Register

I²C Master Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10:8	mcode	R/W	0	**MCODE** These bits set the master code used in Hs-mode operation
7	ex_addr_en	R/W	0	**Slave Extended Addressing Enable** 0: Send a 7-bit address to the slave 1: Send a 10-bit address to the slave
6:3	-	RO	0	**Reserved**
2	stop	R/W1O	0	**Send STOP Condition** 1: Send a STOP Condition at the end of the current transaction *Note: This bit is automatically cleared by the hardware when the STOP condition begins.*

I²C Master Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a slave, the master may send another START to retain control of the bus. 1: Send a Repeated START condition to slave instead of sending a STOP condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Master Mode Transfer** 1: Start master mode transfer *Note: This bit is automatically cleared by the hardware when the transfer is completed or aborted.*

Table 13-19: I²C SCL Low Control Register

I²C Clock Low Control			I2Cn_CLKLO	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	lo	R/W	0x001	**Clock Low Time** In master mode, this configures the SCL low time. $t_{SCL\_LO} = f_{I2C\_CLK} \times (lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 13-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLKHI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	hi	R/W	0x001	**Clock High Time** In master mode, this configures the SCL high time. $t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (hi + 1)$ In both master and slave mode, this also configures the time SCL is held low after new data is loaded from the transmit FIFO or after the software clears *I2Cn_INTFL0.irxm* during IRXM. *Note: 0 is not a valid setting for this field.*

Table 13-21: I²C Hs-Mode Clock Control Register

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:8	hsclk_hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In slave mode, this is the time SCL is held high after data is output on SDA. *Note: See SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	hsclk_lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In slave mode, this is the time SCL is held low after data is output on SDA. *Note: See SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*

Table 13-22: I²C Timeout Register

I²C Timeout			I2Cn_TIMEOUT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	scl_to_val	R/W	0	**Bus Error SCL Timeout Period** Set this value to the number of I2C clock cycles desired to cause a bus timeout error. The peripheral timeout timer starts when it pulls SCL low. After the peripheral releases the line, if the line is not pulled high prior to the timeout number of I2C clock cycles, a bus error condition is set (*I2Cn_INTFL0.to_err* = 1) and the peripheral releases the SCL and SDA lines 0: Timeout disabled. All other values result in a timeout calculation of: $t_{BUS\_TIMEOUT} = {}^{1}/_{f_{I2C\_CLK}} \times scl\_to\_val$ *Note: The timeout counter monitors the I2Cn peripheral's driving of the SCL pin, not an external I2C device driving the SCL pin.*

Table 13-23: I²C DMA Register

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rx_en	R/W	0	**Receive DMA Channel Enable** 0: Disable 1: Enable
0	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disable 1: Enable

Table 13-24: I²C Slave Address Register

I²C Slave Address			I2Cn_SLAVE	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Slave Mode Extended Address Length Select** 0: 7-bit addressing 1: 10-bit addressing
14:10	-	RO	0	**Reserved**

I²C Slave Address			I2Cn_SLAVE	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
9:0	addr	R/W	0	**Slave Mode Slave Address** In slave mode operation, (*I2Cn_CTRL.mstr* = 0), set this field to the slave address for the I2Cn port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bit and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

14. Inter-Integrated Sound Interface (I²S)

I²S is a serial audio interface for communicating pulse-code modulation (PCM) encoded streams between devices. The peripheral supports both master and slave modes.

Key features:

- Stereo (2 channel) and mono (left or right channel option) formats.
- Separate DMA channels for transmit and receive.
- Flexible timing
  - Configurable sampling rate from $^{1}/_{65536}$ to 1 of the I²S input clock.
- Flexible data format
  - Number of bits per data word can be selected from 1 to 32, typically 8, 16, 24, or 32-bit width.
  - Feature enhancement not in the I²S Specification
    - Word/Channel select polarity control.
    - First bit position selection.
    - Selectable FIFO data alignment to the MSB or the LSB of the sample.
    - Sample size less than the word size with adjustment to MSB or LSB of the word.
    - Optional sign extension.
- Full-duplex serial communication with separate I²S serial data input and serial data output pins.

14.1 Instances

Table 14-1: MAX32655 I²S Instances

Instance	Supported Channels	I2S_CLK Clock Options		Receive FIFO Depth	Transmit FIFO Depth
I2S0	Stereo	ERFO	PCLK	8 × 32-bits	8 × 32-bits

Note: ERFO must be enabled for master operation; in slave operation external clocking is used for the LRCLK and BCLK input pins.

14.1.1 I²S Bus Lines and Definitions

The I²S peripheral includes support for the following signals:

1. Bit clock line
   - Continuous serial clock (SCK) referred to as bit clock (BCLK) in this document.
2. Word clock line
   - Word select (WS) referred to as left right clock (LRCLK) in this document.
3. Serial data input (SDI)
4. Serial data output (SDO)
5. ERFO is required for operation in master mode and must be enabled.

Detailed pin and alternate function mapping is shown in *Table 14-2*.

Table 14-2: MAX32655 I²S Pin Mapping

Instance	I²S Signal	Pin Description	81 CTBGA Pin Number	Alternate Function Number	Notes
I2S0	BCLK (SCK)	I²S bit clock	P1.2	AF1	Also referred to as serial clock
	LRCLK (WS)	I²S left/right clock	P1.3	AF1	Also referred to as word select
	SDI	I²S serial data input	P1.4	AF1	
	SDO	I²S serial data output	P1.5	AF1	

14.2 Details

The I²S supports full-duplex serial communication with separate SDI and SDO pins. *Figure 14-1* shows an interconnect between a peripheral configured in host mode, communicating with an external I²S slave receiver and an external I²S transmitter. In master mode, the peripheral hardware generates the BCLK and LRCLK, and are output to each slave device.

Note: Master operation requires the use of the ERFO to generate the LRCLK and BCLK signals.

Figure 14-1: I²S Master Mode

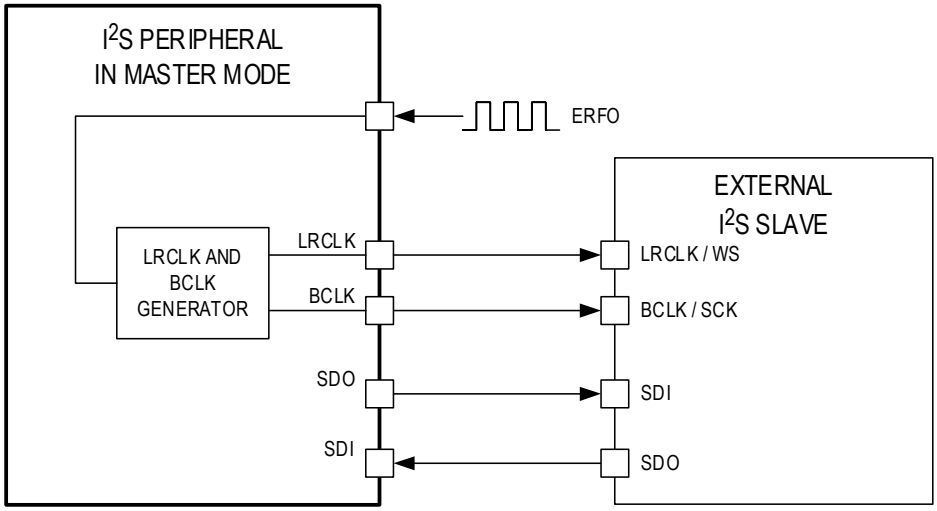
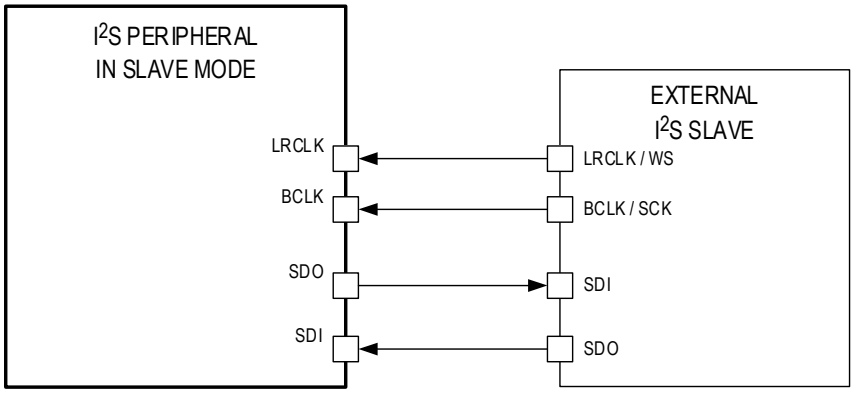


Figure 14-2 shows the I²S peripheral configured for slave operation. The LRCLK and BCLK signals are generated externally, and are inputs to the I²S peripheral.

Figure 14-2: I²S Slave Mode



14.3 Master and Slave Mode Configuration

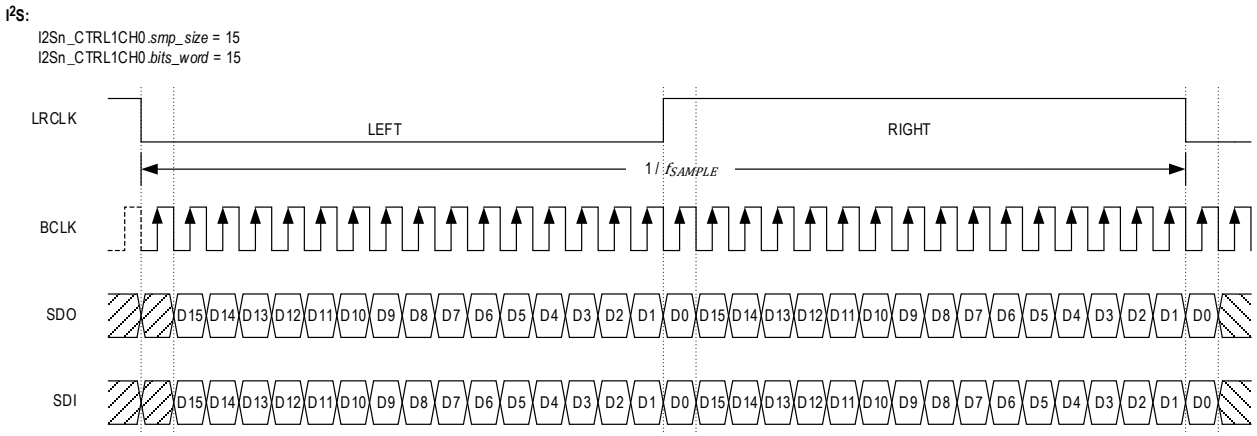
The device supports master and slave modes. In master mode, the BCLK and LRCK signals are generated internally, and output on the BCLK and LRCLK pins. In slave mode, the BCLK and LRCLK pins are configured as inputs, and the peripheral timing is controlled by the external clock source.

Table 14-3: I²S Mode Configuration

Device Mode	*I2Sn_CTRL0CH0.ch_mode*	LRCLK	BCLK
Master	0	Output to slave	Output to slave
Slave	3	Input from master	Input from master

14.4 Clocking

Figure 14-3:Audio Interface I²S Signal Diagram



I²S communication is synchronized using two signals, the LRCLK and the BCLK. When the I²S peripheral is configured as a master, the BCLK and LRCLK signals are generated internally by the peripheral using the ERFO. See *Table 14-2* for details of the I²S pin mapping and alternate function selection. If using the I²S peripheral in master mode, the ERFO must be enabled to generate the BCLK and LRCLK signals.

When the I²S peripheral is configured in slave mode, the BCLK and LRCLK pins must be configured as inputs. An external master generates the BCLK and LRCLK signals, which the peripheral uses to synchronize itself to the I²S bus. *Figure 14-3* shows the default I²S signals and timing for I²S communication.

The BCLK frequency is the product of the sample rate, the number of bits per channel (left and right), and the number of channels. For CD audio sampled at a frequency of 44.1kHz, with 16-bit sample width, and stereo audio (left and right) the bit clock frequency, f_{BCLK}, is 1.4112MHz as shown in *Equation 14-1*.

Equation 14-1: CD Audio Bit Frequency Calculation

$$f_{BCLK} = 44.1\ kHz \times 16 \times 2 = 1.4112MHz$$

14.4.1 BCLK Generation for Master Mode

As indicated by Equation 14-1, the requirements for determining the BCLK frequency are:

1. Audio sample frequency
2. Number of bits per sample, referred to as sample width

Using the above requirements, Equation 14-2 shows the formula to calculate the bit clock frequency for a given audio file.

Equation 14-2: Calculating the Bit Clock Frequency for Audio

$$f_{BCLK} = f_{SAMPLE} \times Sample\ Width \times 2$$

In master mode, the I²S external clock input is used to generate the BCLK frequency. The I²S external clock is divided by the *I2Sn_CTRL1CH0.clkdiv* field to achieve the target BCLK frequency as shown in Equation 14-3.

Equation 14-3: Master Mode BLCK Generation Using the I²S External Clock

$$f_{BCLK} = \frac{f_{ERFO}}{(I2Sn\_CTRL1CH0.clkdiv + 1) \times 2}$$

Use Equation 14-4 to determine the I²S clock divider for a target BCLK frequency.

Equation 14-4: Master Mode Clock Divisor Calculation for a Target Bit Clock Frequency

$$I2Sn\_CTRL1CH0.clkdiv = \frac{f_{ERFO}}{2 \times f_{BCLK}} - 1$$

14.4.2 LRCLK Period Calculation

An I²S data stream can carry mono (either left or right channel) or stereo (left and right channel) data. The LRCLK signal indicates which channel is currently being sent, either left or right channel data, as shown in Figure 14-3. The LRCLK is a 50% duty cycle signal and is the same frequency as the audio sampling frequency, f_{SAMPLE}.

The I²S Peripheral uses the bits per word field, *I2Sn_CTRL1CH0.bits_word*, to define the sample width of the audio, equivalent to the number of bit clocks per channel. This value should be set to the sample width of the audio minus 1. For example, the software should set the *I2Sn_CTRL1CH0.bits_word* field to 15 for audio sampled using a 16-bit width.

Equation 14-5: Bits Per Word Calculation

$$I2Sn\_CTRL1CH0.bits\_word = Sample\ Width - 1$$

The LRCLK frequency, or word select frequency, is automatically generated by the I²S peripheral hardware when it is set to operate as a master. The LRCLK frequency calculation is shown in Equation 14-6.

Equation 14-6: LRCLK Frequency Calculation

$$f_{LRCLK} = f_{BCLK} \times (I2Sn\_CTRL1CH0.bits\_word + 1)$$

14.5 Data Formatting

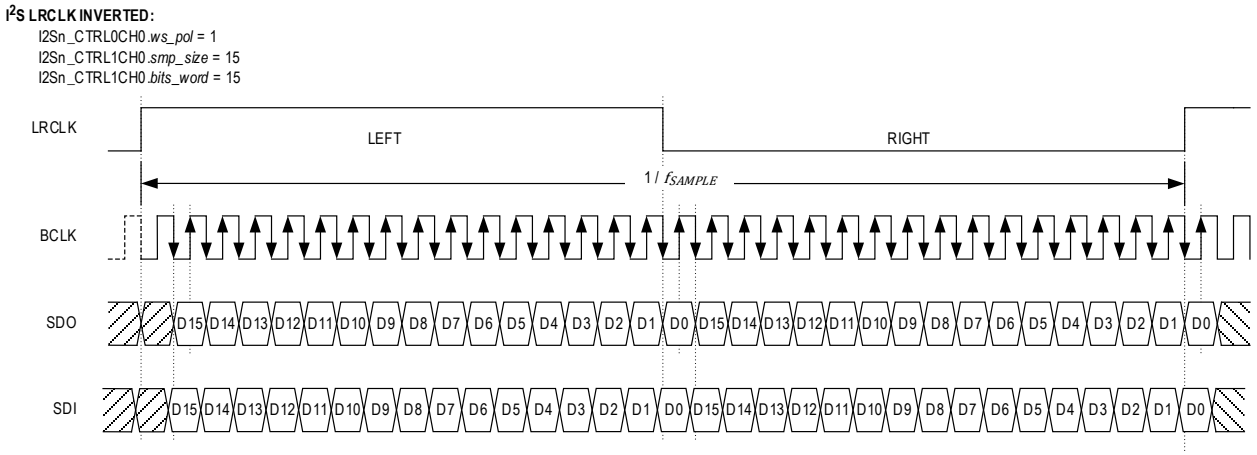
14.5.1 Sample Size

The sample size field, *I2Sn_CTRL1CH0.smp_word*, defines the number of desired samples within each channel, left, right or mono, for the peripheral. This field can be less than or equal to the *I2Sn_CTRL1CH0.bits_word* field. For example, for 16-bit sample width audio, the *I2Sn_CTRL1CH0.bits_word* field must be set to 15. However, the sample size field can be set from 0 to 15. Setting the sample size to 0 is equivalent to setting it to the value of the bits per word field. The sample size field determines how many of the bits per word are transmitted or saved per channel. The sample size field is a 0 based field, therefore, setting *I2Sn_CTRL1CH0.smp_word* to 15 collects 16 samples. See Figure 14-6 for an example of the bits per word field's setting compared to the sample size field's setting.

14.5.2 Word Select Polarity

Left channel data, by default, is transferred when the LRCLK signal is low, and right channel data is transferred when the LRCLK signal is high. The polarity of the LRCLK is programmable allowing left and right data to be swapped. The LRCLK polarity is controlled using the word select polarity field, *I2Sn_CTRL0CH0.ws_pol*. By default, LRCLK low is for the left channel, high is for the right channel as shown in *Figure 14-3*. Setting *I2Sn_CTRL0CH0.ws_pol* to 1 inverts the LRCLK polarity, using LRCLK high for the left channel and LRCLK low for the right channel as shown in *Figure 14-4*.

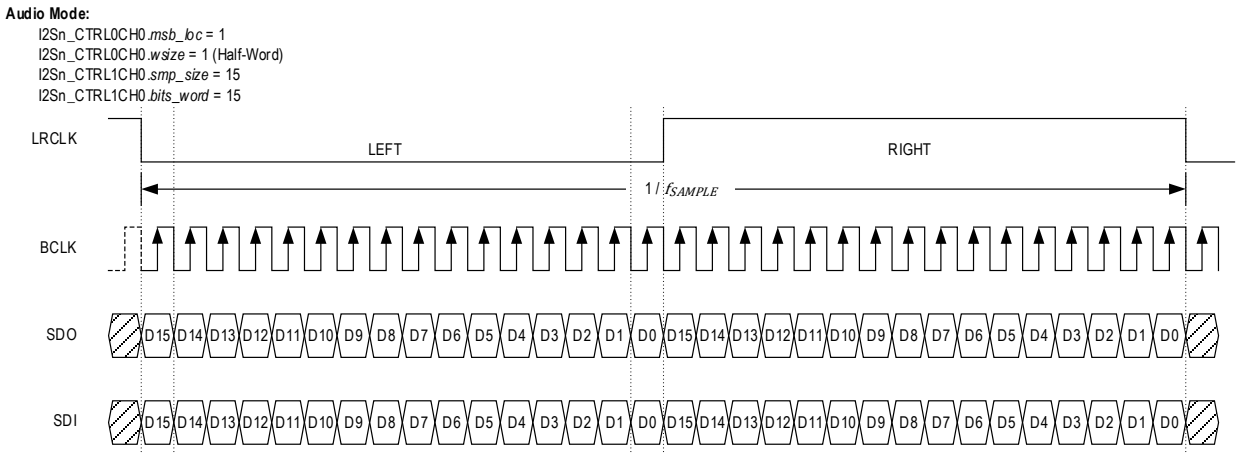
Figure 14-4: Audio Mode with Inverted Word Select Polarity



14.5.3 First Bit Location Control

The default setting is for the first bit of I²S data to be located at the second complete BCLK cycle after the LRCLK transition as required by the I²S specification. See *Figure 14-3* for the standard data sampling configuration. Optionally, the first bit location can be left justified, resulting in the first bit of data being sampled on the first BCLK cycle after the LRCLK signal transitions as shown in *Figure 14-5*. Set *I2Sn_CTRL0CH0.msb_loc* to 1 to left justify the data with respect to the LRCLK.

Figure 14-5: Audio Master Mode Left-Justified First Bit Location



14.5.4 Sample Adjustment

When the sample size field, *I2Sn_CTRL1CH0.smp_word*, is less than the bits per word field, *I2Sn_CTRL1CH0.bits_word*, use the *I2Sn_CTRL1CH0.adjst* field to set which bits are stored in the receive FIFO or transmitted from the transmit FIFO, either from the first sample of the SDI/SDO line or the last sample of the SDI/SDO line for the left and right channels. *Figure 14-6* shows an example of the default adjustment, MSB, where *I2Sn_CTRL1CH0.smp_word* = 7 and *I2Sn_CTRL1CH0.bits_word* = 15. *Figure 14-7* shows the adjustment set to the LSB of the SDI/SDO data.

Figure 14-6: MSB Adjustment when Sample Size is Less Than Bits Per Word

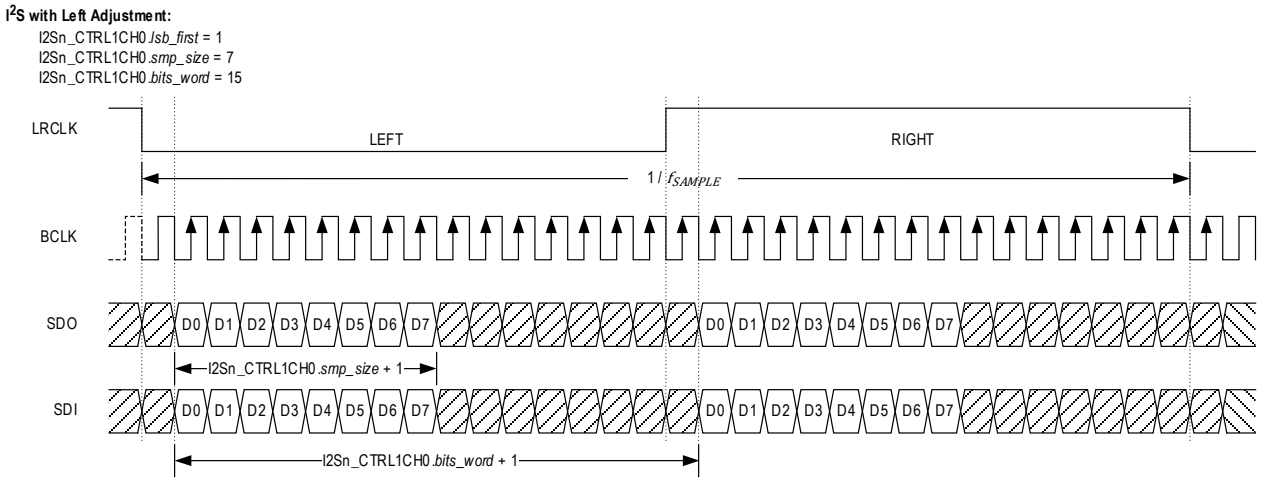
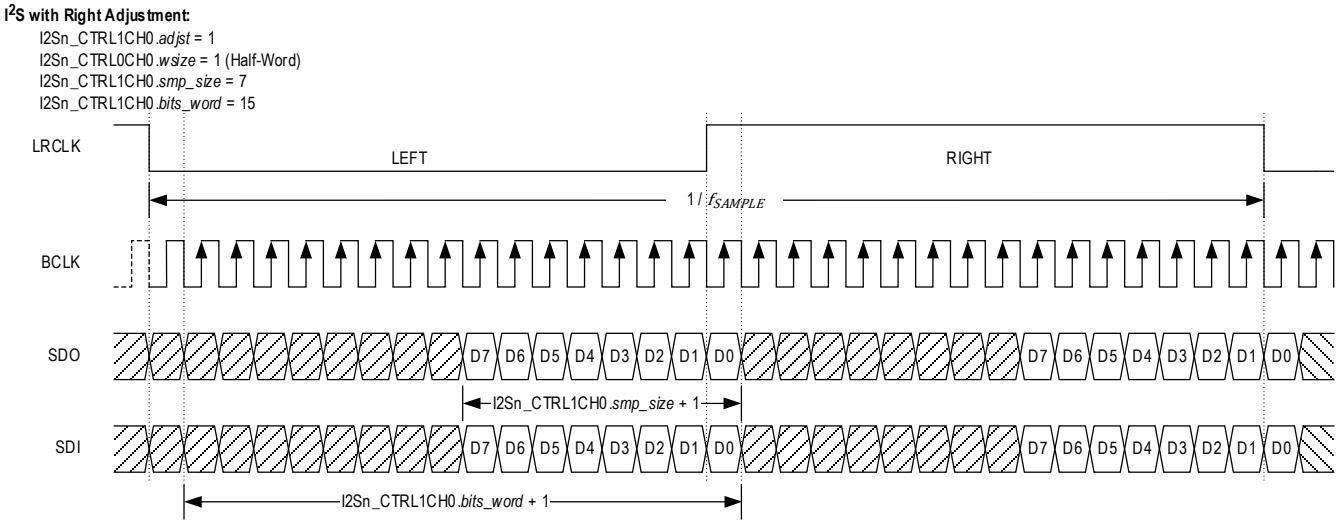


Figure 14-7: LSB Adjustment when Sample Size is Less Than Bits Per Word



14.5.5 Stereo/Mono Configuration

The I²S can transfer stereo or mono data based on the *I2Sn_CTRL0CH0.stereo* field. In stereo mode, the default mode, both the left and right channels hold data. In mono mode, only the left or right channel contain data. For stereo mode, set *I2Sn_CTRL0CH0.stereo* to 0. Set *I2Sn_CTRL0CH0.stereo* field to 2 for left channel mono. Set *I2Sn_CTRL0CH0.stereo* field to 3 for right channel mono.

Figure 14-8: I²S Mono Left Mode

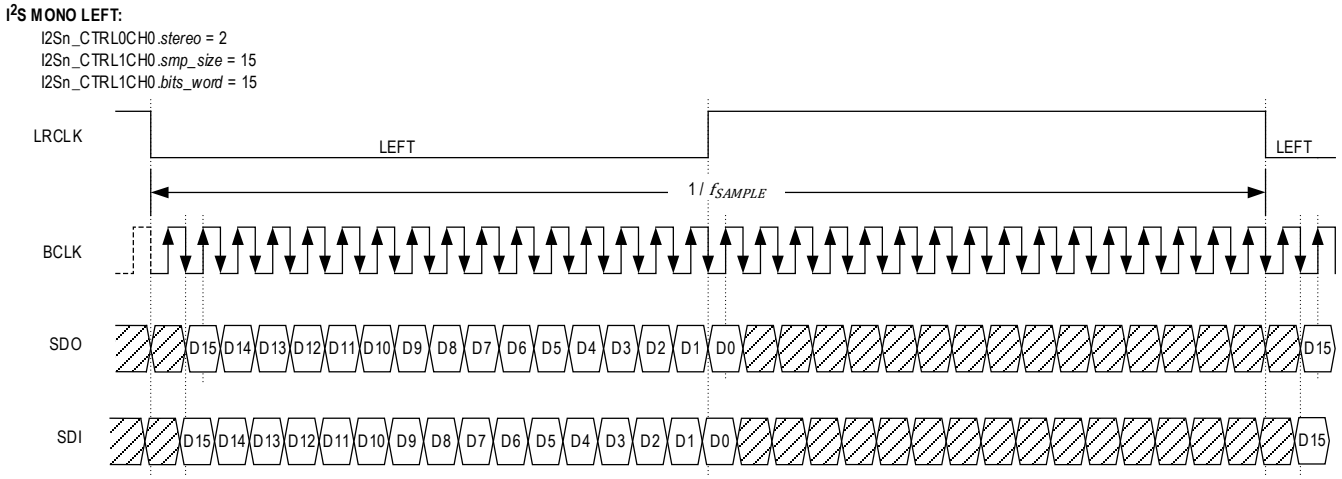
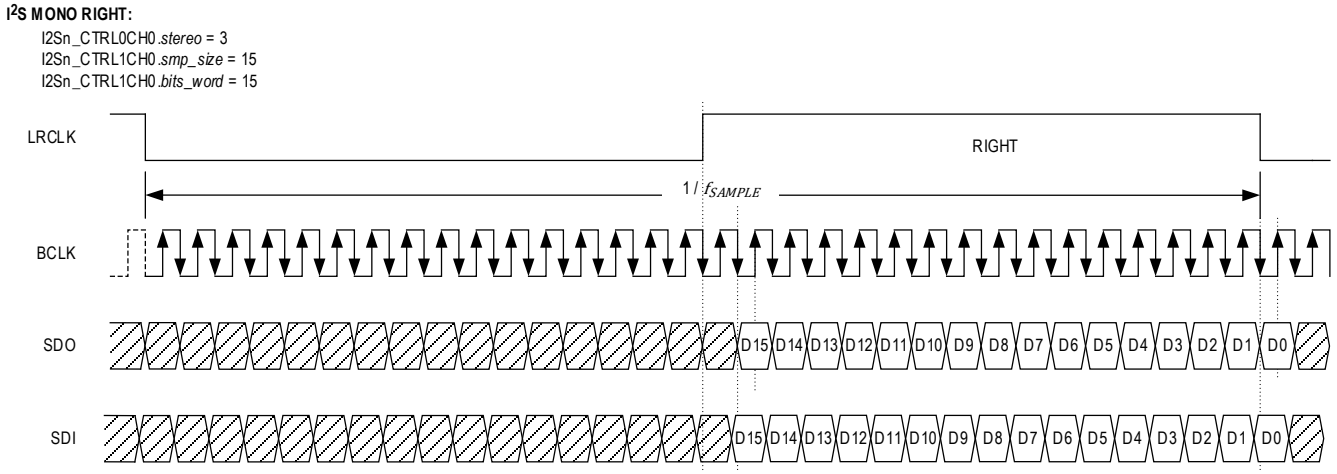


Figure 14-9: I²S Mono Right Mode



14.6 Transmit and Receive FIFOs

14.6.1 FIFO Data Width

I²S audio data is programmable from 1 to 32 bits using the *I2Sn_CTRL1CH0.bits_word* field. The software can set the FIFO width to either 8-bits (byte), 16-bits (half-word), or 32-bits (word). Set the FIFO width using the *I2Sn_CTRL0CH0.wsize* field. For FIFO word sizes less than 32-bits, the data frame, comprising a full LRCLK cycle, may still be 64 bits; the unused bits are transmitted as zero by the hardware.

14.6.2 Transmit FIFO

An I²S transaction is started by writing data to the transmit FIFO using the *I2Sn_FIFOCH0.data* register, either directly or using a DMA channel. The data written is automatically transmitted out by the hardware, a FIFO word, as defined using the I2Sn_CTRL0CH0.*wsize* field, at a time, in the order it was written to the transmit FIFO. Use the I²S interrupt flags to monitor the transmit FIFO status and determine when the transfer cycle(s) have completed.

If the transmit FIFO becomes empty an error condition occurs and results in undefined behavior.

14.6.3 Receive FIFO

The received data is loaded into the receive FIFO and it can then be unloaded by reading from the *I2Sn_FIFOCH0.data* register. An overrun event occurs if the receive FIFO is full and another word is shifted into the FIFO.

14.6.4 FIFO Word Control

The data width of the transmit and receive FIFOs can be configured using the *I2Sn_CTRL0CH0.wsize* field. The following tables describe the data ordering based on the *I2Sn_CTRL0CH0.wsize* setting.

The transmit and receive FIFOs must be flushed, and the peripheral reset by the software prior to reconfiguration. The software resets the peripheral by setting the *I2Sn_CTRL0CH0.rst* field to 1.

Table 14-4: Data Ordering for Byte Data Size (Stereo Mode)

Byte Data Width (*I2Sn_CTRL0CH0.wsize* = 0)				
FIFO Entry	**MSByte**			**LSByte**
FIFO 0	Right Channel Byte 1	Left Channel Byte 1	Right Channel Byte 0	Left Channel Byte 0
FIFO 1	Right Channel Byte 3	Left Channel Byte 3	Right Channel Byte 2	Left Channel Byte 2
…	…	…	…	…
FIFO *7*	Right Channel Byte 14	Left Channel Byte 14	Right Channel Byte 13	Left Channel Byte 13

Table 14-5: Data Ordering for Half-Word Data Size (Stereo Mode)

Half-Word Data Width (*I2Sn_CTRL0CH0.wsize* = 1)		
FIFO Entry	**MS Half-Word**	**LS Half-Word**
FIFO 0	Right Channel Half-Word 0	Left Channel Half-Word 0
FIFO 1	Right Channel Half-Word 1	Left Channel Half-Word 1
…	…	…
FIFO *7*	Right Channel Half Word *7*	Left Channel Half-Word *7*

Table 14-6: Data Ordering for Word Data Size (Stereo Mode)

Word Data Width (*I2Sn_CTRL0CH0.wsize* = 2 or 3)	
FIFO Entry	**Word**
FIFO 0	Left Channel Word 0
FIFO 1	Right Channel Word 0
FIFO 2	Left Channel Word 1
FIFO 3	Right Channel Word 1
…	…
FIFO 6	Left Channel Word 3
FIFO 7	Right Channel Word 3

14.6.5 FIFO Data Alignment

The I²S data can be left aligned (reset default), or right aligned using the I2Sn_CTRL0CH0.*align* field. The following conditions apply to each setting:

Left aligned: I2Sn_CTRL0CH0.*align* = 0

- If the number of bits per word is greater than the FIFO data width:
  - Receive: All bits after the LSB of the FIFO data width is discarded.
  - Transmit: All bits after the LSB of the FIFO data width are sent as 0.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is stored starting at the MSB of the FIFO entry up to the number of bits per word plus one bit.
  - Transmit: The data in the transmit FIFO is sent from the LSB to the number of bits plus 1.

Right aligned: I2Sn_CTRL0CH0.*align* = 1

- If the number of bits per word is greater than the FIFO data width:
  - Receive: The data received is stored in the receive FIFO starting with the LSB up to the FIFO data width and any additional bits are discarded.
  - Transmit: 0 bits are transmitted for all bits greater than the FIFO data width. For example, if the bits per word field is set to 12 and the FIFO data width is 8, the first 4 bits are transmitted as 0 and then the 8-bits of data in the FIFO are transmitted.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is sign extended and saved to the receive FIFO.
  - Transmit: The data in the transmit FIFO is sent from the LSB to the number of bits plus 1.

14.6.6 Typical Audio Configurations

Table 14-7 shows the relationship between the bits per word field and the sample size field. Equation 14-7 shows the required relationship between the sample size field and the bits per word field.

Equation 14-7: Sample Size Relationship Bits per Word

$$I2Sn\_CTRL1CH0.smp\_size \leq I2Sn\_CTRL1CH0.bits\_word$$

The I2Sn_CTRL1CH0.*bits_word* column in Table 14-7 is set by the equation $\frac{\#\ BCLK}{Channel} - 1$. The I2Sn_CTRL1CH0.*smp_size* column is the number of samples per word captured from the I²S bus and is calculated by the equation $\frac{\#\ Samples}{Channel} - 1$. Channel refers to the left and right channels of audio.

Table 14-7: Configuration for Typical Audio Width and Samples per WS Clock Cycle

Audio Sample Width/ Samples per WS Cycle	$\frac{\#\ BCLK}{Channel}$	$\frac{\#\ Samples}{Channel}$	I2Sn_CTRL1CH0			Sign extension (align = 1)[†]
			bits_word	*smp_size*	*wsize*	
8-bit / 16	8	8	7	7	0	
16-bit / 32	16	16	15	15	1	
20-bit / 40	20	20	19	19	2	sign
24-bit / 48	24	24	23	23	2	sign
24-bit / 64	32	24	31	23	2	sign
32-bit / 64	32	32	31	31	2	

[†] *Sign Extension applies only when I2Sn_CTRL0CH0.align is set to 1 and I2Sn_CTRL0CH0.smp_size is less than the FIFO width size setting.*

14.7 Interrupt Events

The I²S peripheral generates interrupts for the events shown in *Table 14-8*. An interrupt is generated if the corresponding interrupt enable field is set. The interrupt flags stay set until cleared by the software by writing 1 to the interrupt flag field.

Table 14-8. I²S Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Receive FIFO overrun	*I2Sn_INTFL.rx_ov_ch0*	*I2Sn_INTEN.rx_ov_ch0*
Receive threshold	*I2Sn_INTFL.rx_thd_ch0*	*I2Sn_INTEN.rx_thd_ch0*
Transmit FIFO half-empty	*I2Sn_INTFL.tx_he_ch0*	*I2Sn_INTEN.tx_he_ch0*
Transmit FIFO one byte remaining	*I2Sn_INTFL.tx_ob_ch0*	*I2Sn_INTEN.tx_ob_ch0*

14.7.1 Receive FIFO Overrun

A receive FIFO overrun event occurs if the number of data words in the receive FIFO, *I2Sn_DMACH0.rx_lvl* is equal to the RX_FIFO_DEPTH and another word has been shifted into the FIFO. The hardware automatically sets the *I2Sn_INTFL.rx_ov_ch0* field to 1 when this event occurs.

14.7.2 Receive FIFO Threshold

A receive FIFO threshold event occurs when a word is shifted in and the number of words in the receive FIFO, *I2Sn_DMACH0.rx_lvl*, exceeds the *I2Sn_CTRL0CH0.rx_thd_val*. The event does not occur if the opposite transition occurs. When this event occurs, hardware automatically sets the *I2Sn_INTFL.rx_thd_ch0* field to 1.

14.7.3 Transmit FIFO Half-Empty

A transmit FIFO half-empty event occurs when the number of words in the transmit FIFO, *I2Sn_DMACH0.tx_lvl*, is less than ½ of the TX_FIFO_DEPTH as shown in *Equation 14-8*. When this event occurs, the *I2Sn_INTFL.tx_he_ch0* flag is set to 1 by hardware.

Note: The transmit FIFO half empty interrupt flag is set by the hardware one BCLK cycle prior to the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software may receive the interrupt while the actual transmit FIFO level is still equal to ½ of the TX_FIFO_DEPTH. The software should always read the transmit FIFO level prior to filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2Sn_DMACH0.tx_lvl field.

Equation 14-8: Transmit FIFO Half-Empty Condition

$$I2Sn\_DMACH0.tx\_lvl < \left(\frac{TX\ FIFO\ DEPTH}{2}\right)$$

14.7.4 Transmit FIFO One Entry Remaining

A transmit FIFO one entry remaining event occurs when the number of entries in the transmit FIFO is 1, *I2Sn_DMACH0.tx_lvl* = 1. When this event occurs, the *I2Sn_INTFL.tx_ob_ch0* flag is set to 1 by the hardware.

Note: The transmit FIFO one entry remaining interrupt flag is set by the hardware one BCLK cycle prior to the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software may receive the interrupt while the actual transmit FIFO level is still equal to 2. The software should always read the transmit FIFO level prior to filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2Sn_DMACH0.tx_lvl field.

14.8 Direct Memory Access

The I²S supports DMA for both transmit and receive; separate DMA channels can be connected to the receive and transmit FIFOs. The following describe the behavior of the receive and transmit DMA requests.

- A receive DMA request is asserted when the number of words in the receive FIFO is greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of valid bytes in the transmit FIFO is less than ½ of the transmit FIFO's depth.

14.9 Block Operation

After exiting a power-on reset, the IP is disabled by default. It must be enabled and configured by the software to establish the I²S serial communication. A typical firmware sequence is shown below.

1. Set *GCR_PCLKDIS1*.*i2s0* to 0 to enable the I²S peripheral clock source shown in *Table 14-1*.
2. Disable the I²S clock by setting *I2Sn_CTRL1CH0*.*en* to 0.
3. Set *I2Sn_CTRL0CH0*.*rst* to 1 to reset the I²S configuration.
4. Set *I2Sn_CTRL1CH0*.*flush* to 1 to flush the FIFO buffers.
5. Configure the *I2Sn_CTRL0CH0*.*ch_mode* to select the master or slave configuration.
   a. For master mode, configure the baud rate by programming the *I2Sn_CTRL1CH0*.*clkdiv* field to achieve the required bit rate, set the *I2Sn_CTRL1CH0*.*smp_size* field to the desired sample size of the data, and the *I2Sn_CTRL1CH0*.*adjst* field if the Sample Size is smaller than the number of bits per word.
6. Configure the threshold of the receive FIFO by programming the *I2Sn_CTRL1CH0*.*rx_thd*. The threshold of the transmit FIFO is a fixed value, which is half of the transmit FIFO depth.
7. If desired, configure DMA operation, see section *Direct Memory Access* for details.
8. Enable interrupt functionality by configuring the *I2Sn_INTEN* register if desired.
9. Program the *clkdiv* bits in *I2Sn_CTRL1CH0* register for the new bit clock frequency.
10. For master operation, load data in the transmit FIFO for transmit.
11. Re-enable the bit clock by setting *I2Sn_CTRL1CH0*.*en* to 1.

14.10 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 14-9*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 14-9: I²S Register Summary

Offset	Register Name	Description
[0x0000]	*I2Sn_CTRL0CH0*	I²S Global Mode Control 0 Register
[0x0010]	*I2Sn_CTRL1CH0*	I²S Master Mode Configuration Register
[0x0030]	*I2Sn_DMACH0*	I²S DMA Control Channel Register
[0x0040]	*I2Sn_FIFOCH0*	I²S FIFO Register
[0x0050]	*I2Sn_INTFL*	I²S Interrupt Status Register
[0x0054]	*I2Sn_INTEN*	I²S Interrupt Enable Register

14.10.1 Register Details

Table 14-10: I²S Control 0 Register

I²S Control 0 Register			I2Sn_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	rx_thd_val	R/W	0	**Receive FIFO Interrupt Threshold** This field specifies the level of the receive FIFO for the threshold interrupt generation. Values of 0 or greater than the RX_FIFO_DEPTH are ignored.
23:21	-	RO	0	**Reserved**
20	fifo_lsb	R/W	0	**FIFO Bit Field Control** Only used if the FIFO size is larger than the sample size and *I2Sn_CTRL0CH0.align* = 0. For transmit, the LSB part is sent from the FIFO. For receive, store the LSB part in the FIFO without sign extension. 0: Disabled 1: Enabled
19	rst	R/W1O	0	**Reset** Write 1 to reset the I²S peripheral. The hardware automatically clears this field to 0 when the reset is complete. 0: Reset not in process. 1: Reset peripheral.
18	flush	R/W1O	0	**FIFO Flush** Write 1 to start a flush of the receive FIFO and the transmit FIFO. The hardware automatically clears this field when the operation is complete. 0: Flush complete or not in process. 1: Flush receive and transmit FIFOs.
17	rx_en	R/W	0	**Receive Enable** Enable receive mode for the I²S peripheral. 0: Disabled 1: Enabled
16	tx_en	R/W	0	**Transmit Enable** Enable transmit mode for the I²S peripheral. 0: Disabled 1: Enabled
15:14	wsize	R/W	0x3	**Data Size When Reading/Writing FIFO** Set this field to the desired width for data writes and reads from the FIFO. 0: Byte 1: Half-word (16 bits) 2-3: Word (32 bits)
13:12	stereo	R/W	0	**I²S Mode** Select the mode for the I²S to stereo, mono left channel only, or mono right channel only. 0-1: Stereo 2: Mono left channel 3: Mono right channel
11	-	RO	0	**Reserved**
10	align	R/W	0	**FIFO Data Alignment** Set this field to control the alignment of the data in the FIFOs. This field is only used if the FIFO data width, *I2Sn_CTRL0CH0.wsize*, is not equal to the bits per word field. 0: MSB 1: LSB

I²S Control 0 Register				I2Sn_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
9	msb_loc	R/W	0	**First Bit Location Sampling** This field controls when the first bit is transmitted/received in relation to the LRCLK. By default, the first bit is transmitted/received on SDO/SDI on the second complete LRCLK cycle. Set this field to 1 to transmit/receive the first bit of data on the first complete LRCLK cycle. 0: Second complete LRCLK cycle is the first bit of the data 1: First complete LRCLK cycle is the first bit of the data	
8	ws_pol	R/W	0	**LRCLK Polarity Select** This field determines the polarity of the LRCLK signal associated with the left channel data. Set this field to 1 to associate the left channel with the LRCLK high state. The default setting is the standard I²S association. 0: LRCLK low for left channel 1: LRCLK high for left channel	
7:6	ch_mode	R/W	0	**Mode** Set this field to indicate master or slave I²S operation. When using master mode, the ERFO must be used to generate the LRCLK/BCLK signals. 0: Master mode, internal generation of LRCLK/BCLK using the ERFO. 1-2: Reserved for Future Use 3: Slave mode, external generation of LRCLK/BCLK	
5:2	-	DNM	0	**Reserved, Do Not Modify**	
1	lsb_first	R/W	0	**LSB First** Setting this field to 1 indicates the least significant bit of the data is transmitted/received first on the SDI/SDO pins. The default setting, 0, indicates the most significant bit of the data is received first. 0: Disabled 1: Enabled	
0	-	RO	0	**Reserved**	

Table 14-11: I²S Master Mode Configuration Register

I²S Master Mode Configuration				I2Sn_CTRL1CH0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:16	clkdiv	R/W	0	**I²S Frequency Divisor** Set this field to the required divisor to achieve the desired frequency for the I²S BCLK. See *BCLK Generation for Master Mode* for detailed information. *Note: This field only applies when the I²S peripheral is set to master mode, I2Sn_CTRL0CH0.ch_mode = 0.*	
15	adjst	R/W	0	**Data Justification When Sample Size is Less than Bits Per Word** This field is used to determine which bits are used if the sample size is less than the bits per word. 0: Left adjustment 1: Right adjustment	
14	-	RO	0	**Reserved**	

I²S Master Mode Configuration			I2Sn_CTRL1CH0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
13:9	smp_size	R/W	0	**Sample Size** This field is the desired sample size of the data received or transmitted with respect to the Bits per Word field. In most use cases, the sample size is equal to the bits per word. However, in some situations fewer number of bits are required by the application and this field allows flexibility. An example use case would be for 16-bit audio being received and the application only needs 8-bits of resolution. See *Sample Size* for additional details. *Note: The sample size is equal to I2Sn_CTRL1CH0.bits_word when I2Sn_CTRL1CH0.smp_size = 0 or I2Sn_CTRL1CH0.smp_size > I2Sn_CTRL1CH0.bits_word.*
8	en	R/W	0	**I²S Enable** For master mode operation, this field is used to start the generation of the I²S LRCLK and BCLK outputs. In slave mode, this field enables the peripheral to begin receiving signals on the I²S interface. 0: Disabled. 1: Enabled
7:5	-	RO	0	**Reserved**
4:0	bits_word	R/W	0	**I²S Word Length** This field is defined as the I²S data bits per left and right channel. *Example: If the bit clocks is 16 per half frame, bits_word is 15.*

Table 14-12: I²S DMA Control Register

I²S DMA Control			I2Sn_DMACH0	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	rx_lvl	RO	0	**Receive FIFO Level** This field is the number of data words in the receive FIFO.
23:16	tx_lvl	RO	0	**Transmit FIFO Level** This field is the number of data words in the transmit FIFO.
15	dma_rx_en	RW	0	**DMA Receive Channel Enable** 0: Disabled 1: Enabled
14:8	dma_rx_thd_val	RW	0	**DMA Receive FIFO Event Threshold** If the receive FIFO level is greater than this value, then the receive FIFO DMA interface sends a signal to the system DMA indicating the receive FIFO has characters to transfer to memory.
7	dma_tx_en	RW	0	**DMA Transmit Channel Enable** 0: Disabled 1: Enabled
6:0	dma_tx_thd_val	RO	0	**DMA Transmit FIFO Event Threshold** If the transmit FIFO level is less than this value, then the transmit FIFO DMA interface sends a signal to system DMA indicating the transmit FIFO is ready to receive data from memory.

Table 14-13: I²S FIFO Register

I²S FIFO Register			I2Sn_FIFOCH0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**I²S FIFO** Writing to this field loads the next character into the transmit FIFO and increments the *I2Sn_DMACH0.tx_lvl*. Writes are ignored if the transmit FIFO is full. Reads of this field return the next character available from the receive FIFO and decrements the *I2Sn_DMACH0.rx_lvl*. The value 0 is returned if *I2Sn_DMACH0.rx_lvl* = 0.

Table 14-14: I²S Interrupt Flag Register

I²S Interrupt Flag			I2Sn_INTFL	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	DNM	0	**Reserved, Do Not Modify**
3	tx_he_ch0	W1C	0	**Transmit FIFO Half-Empty Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event 1: Event occurred
2	tx_ob_ch0	W1C	0	**Transmit FIFO One Entry Remaining Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event 1: Event occurred
1	rx_thd_ch0	W1C	0	**Receive FIFO Threshold Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event 1: Event occurred
0	rx_ov_ch0	W1C	0	**Receive FIFO Overrun Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event 1: Event occurred

Table 14-15: I²S Interrupt Enable Register

I²S Interrupt Enable			I2Sn_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	DNM	0	**Reserved, Do Not Modify**
3	tx_he_ch0	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled 1: Enabled
2	tx_ob_ch0	R/W	0	**Transmit FIFO One Entry Remaining Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled 1: Enabled
1	rx_thd_ch0	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled 1: Enabled

I²S Interrupt Enable				I2Sn_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**	
0	rx_ov_ch0	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled 1: Enabled	

15. 1-Wire Master (OWM)

The device provides a 1-Wire master that software can use to communicate with one or more external 1-Wire slave devices using a single-signal, combined clock, data protocol. The OWM is contained in the OWM module. The OWM module manages the lower-level details (including timing and drive modes) required by the 1-Wire protocol, allowing the CPU to communicate over the 1-Wire bus at a logical data level.

15.1 1-Wire Master Features

The OWM provides the following features:

- Flexible, 1-Wire timing generation (required 1MHz timing base) using the OWM module clock frequency, which is in turn derived from the current system clock source. The OWM module clock can be pre-scaled to allow proper 1-Wire timing generation using a range of base frequencies.
- Automatic generation of proper 1-Wire time slots for both standard and overdrive timing modes.
- Flexible configuration for 1-Wire line pullup modes: options for internal pullup, external fixed pullup, and optional external strong pullup are available.
- Long-line compensation and bit banging (direct firmware drive) modes.
- 1-Wire reset generation and presence-pulse detection.
- Generation of 1-Wire read and write time slots for single-bit and eight-bit byte transmissions.
- Search ROM Accelerator (SRA) mode, which simplifies the generation of multiple-bit time slots and discrepancy resolution required when completing the Search ROM function to determine the IDs of multiple, unknown 1-Wire slaves on the bus.
- Transmit data completion, received data available, presence pulse detection, and 1-Wire line-error condition interrupts.

For more information about the Maxim 1-Wire protocol and supporting devices, refer to the following resources:

- *AN937: The Book of iButton Standards*
  - *www.maximintegrated.com/AN937*
- *AN1796: Overview of 1-Wire Technology and its Use*
  - *www.maximintegrated.com/AN1796*
- *AN187: 1-Wire Search Algorithm*
  - *www.maximintegrated.com/AN187*

15.2 1-Wire Pins and Configuration

The one instance of the peripheral shown in *Table 15-1* lists the locations of the OWM_IO and OWM_PE signals for the OWM peripheral per package.

Table 15-1: MAX32655 1-Wire Master Peripheral Pins

OWMn	ALTERNATE FUNCTION	ALT FUNCTION #	81-CTBGA
OWM	OWM_IO	AF1	P0.6
	OWM_PE	AF1	P0.7

15.2.1 1-Wire I/O (OWM_IO)

The OWM_IO pin is a bidirectional I/O that is used to directly drive the external 1-Wire bus. As described in the *Book of iButton Standards*, this I/O is generally driven as an open-drain output. The 1-Wire bus requires a common pullup to return the 1-Wire bus line to an idle high state when no master or slave device is actively driving the line low. This pullup can consist of a fixed resistor pullup (connected to the 1-Wire bus outside the microcontroller), an internal pullup enabled by setting *OWM_CFG.int_pullup_enable* to 1, or an OWM module controlled external pullup enabled by setting *OWM_CFG.ext_pullup_mode* to 1.

15.2.2 Pullup Enable (OWM_PE)

The 1-Wire pullup enable (PE) signal is an active high output used to enable an optional external pullup on the 1-Wire bus. This pullup is intended to provide a stronger (lower impedance) pullup on the 1-Wire bus under certain circumstances, such as during overdrive mode.

15.2.3 Clock Configuration

To correctly generate the timing required by the 1-Wire protocol in Standard or Overdrive timing modes, the OWM clock must be set to achieve $f_{owmclk} = 1\text{MHz}$. This clock generates both the Standard and Overdrive timing, so it does not need adjustment when transitioning from Standard to Overdrive mode or vice versa.

The OWM peripheral uses the system peripheral clock, PCLK, divided by the value in the *OWM_CLK_DIV_1US.divisor* field as shown in *Equation 15-1* where $f_{PCLK} = {f_{SYSCLK}}/{2}$.

Equation 15-1: OWM 1MHz Clock Frequency

$$f_{owmclk} = 1MHz = \frac{f_{PCLK}}{OWM\_CLK\_DIV\_1US.divisor}$$

15.3 1-Wire Protocol

The general timing and communication protocols used by the OWM interface are those standardized for the 1-Wire network.

Because the 1-Wire interface is a master interface, it initiates and times all communication on the 1-Wire bus. Except for the present pulse generation when a device first connects to the 1-Wire bus, 1-Wire slave devices complete 1-Wire bus communication only as directed by the 1-Wire bus master. From a firmware perspective, the lowest-level timing and electrical details of how the 1-Wire network operates are unimportant. The application can configure the OWM module properly and direct it to complete low-level operations such as reset, read, and write bit/byte operations. Thus, the OWM module on the microcontroller is designed to interface to the 1-Wire bus at a low level.

15.3.1 Networking Layers

In the *Book of iButton Standards*, the 1-Wire communication protocol is described in terms of the ISO-OSI model (International Organization of Standardization (ISO) Open System Interconnection (OSI) network layer model). Network layers that apply to this description are the Physical, Link, Network, and Transport layers. The Transport layer consists of the

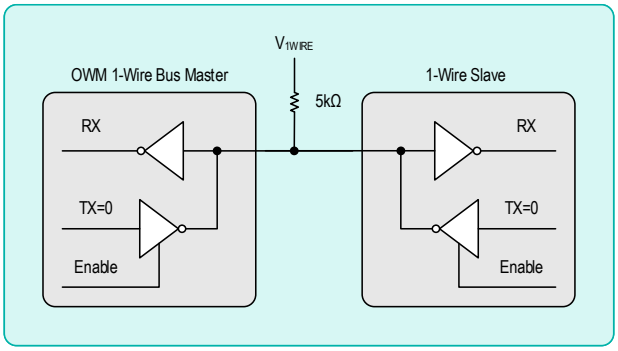
software that transfers memory data other than ROM ID contents to and from the individual 1-Wire network nodes. The Presentation layer corresponds to higher-level application software functions (such as library layers) that implement communication protocols using the 1-Wire layers as a foundation. This document describes the details of the Physical, Link, and Network layers with regards to the OSI network layer model. The Transport and Presentation layers are beyond the scope of this document.

15.3.1.1 Physical Layer

The 1-Wire communication bus consists of a single data/power line plus ground. Devices (either master or slave) that interface to the 1-Wire communication bus using an open-drain (active low) connection, which means that the 1-Wire bus normally idles in a high state.

An external pullup resistor is used to pull the 1-Wire line high when no master or slave device is driving the line. This means that 1-Wire devices do not actively drive the 1-Wire line high. Instead, they either drive the line low or release it (set their output to high impedance) to allow the external resistor to pull the line high. This allows the 1-Wire bus to operate in a wired-AND manner as shown in *Figure 15-1* and avoids bus contention if more than one device attempts to drive the 1-Wire bus at the same time.

Figure 15-1: 1-Wire Signal Interface



15.3.1.2 Link Layer

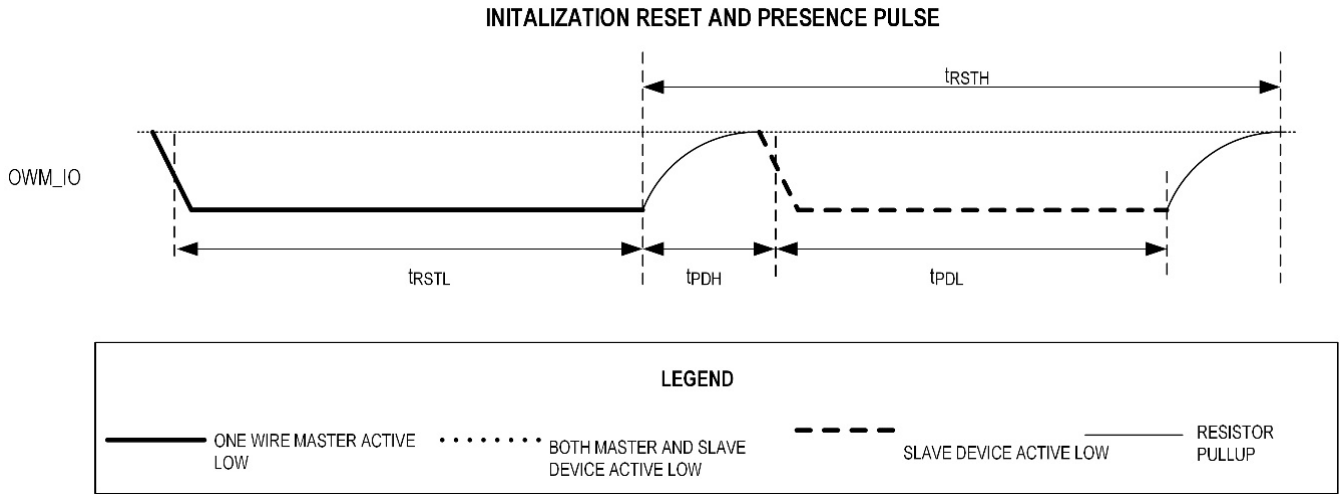
The 1-Wire Bus supports a single master and one or more slave devices (multidrop). slave devices can connect to and disconnect from the 1-Wire Bus dynamically (as is typically the case with iButton devices that operate using an intermittent touch contact interface), which means that it is the master's responsibility to poll the bus as needed to determine the number and types of 1-Wire devices that are connected to the bus.

All communication sequences on the 1-Wire Bus are initiated by the OWM. The OWM determines when 1-Wire data transmissions begin, as well as the overall communication speed that is used. There are three different communication speeds supported by the 1-Wire specification: standard speed, overdrive speed, and hyperdrive speed. However, only standard speed and overdrive speed are supported by the OWM peripheral in the devices.

15.3.1.2.1 OWM Reset and Presence Detect

The OWM begins each communication sequence by sending a reset pulse as shown in *Figure 15-2*. This pulse resets all 1-Wire slave devices on the line to their initial states and causes them all to begin monitoring the line for a command from the OWM. Each 1-Wire slave device on the line responds to the reset pulse by sending out a presence pulse. These pulses from multiple 1-Wire slave devices are combined in wired-AND fashion, resulting in a pulse whose length is determined by the slowest 1-Wire slave device on the bus.

Figure 15-2: 1-Wire Reset Pulse



In general, the 1-Wire line must idle in a high state when communication is not taking place. It is possible for the master to pause communication in between time slots. There is not an overall "timeout" period that causes a slave to revert to the reset state if the master takes too long between one time slot and the next time slot.

The 1-Wire communication protocol relies on the fact that the maximum allowable length for a bit transfer (write 0/1 or read bit) time slot is less than the minimum length for a 1-Wire reset. At any time, if the 1-Wire line is held low (by the master or by any slave device) for more than the minimum reset pulse time, all slave devices on the line interpret this as a 1-Wire reset pulse.

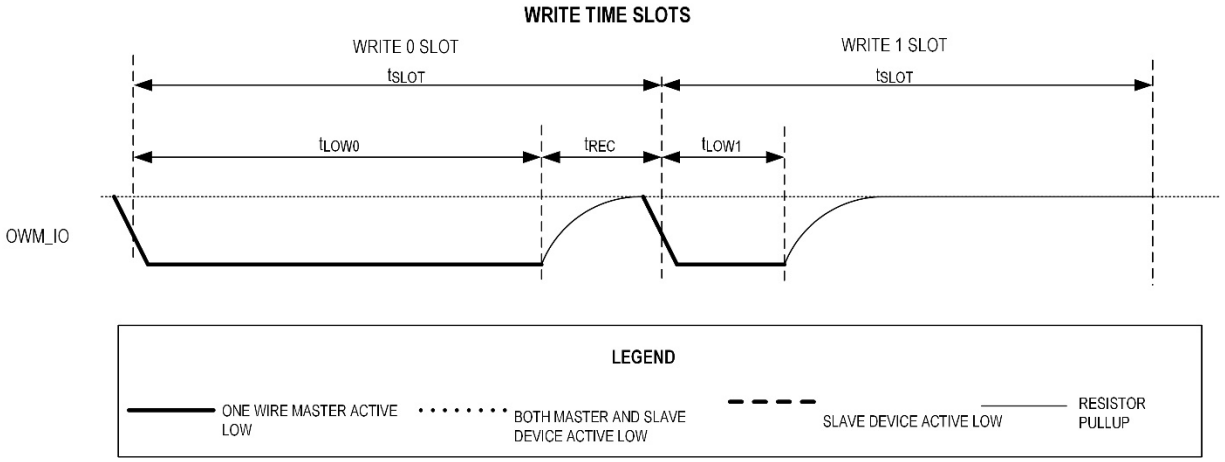
15.3.1.2.2 OWM Write Time Slot

All 1-Wire bit time slots are initiated by the 1-Wire bus master and begin with a single falling edge. There is no indication given by the beginning of a time slot if a read bit or write bit operation is intended, as the time slots all begin in the same manner. Rather, the 1-Wire command protocol enforces agreement between the OWM and slave as to which time slots are used for bit writes and which time slots are used for bit reads.

When multiple bits of a value are transmitted (or read) in sequence, the least significant bit of the value is always sent or received first. The 1-Wire bus is a half-duplex bus, so data is transmitted in only one direction (from master to slave or from slave to master) at any given time.

As shown in *Figure 15-3*, the time slots for writing a 0 bit and writing a 1 bit begin identically, with the falling edge and a minimum-width low pulse sent by the master. To write a one bit, the master releases the line after the minimum low pulse, allowing it to be pulled high. To write a zero bit, the master continues to hold the line low until the end of the time slot.

Figure 15-3: 1-Wire Write Time Slot



From the slave's perspective, the initial falling edge of the time slot triggers the start of an internal timer, and when the proper amount of time has passed, the slave samples the 1-Wire line that is driven by the master. This sampling point is in between the end of the minimum-width low pulse and the end of the time slot.

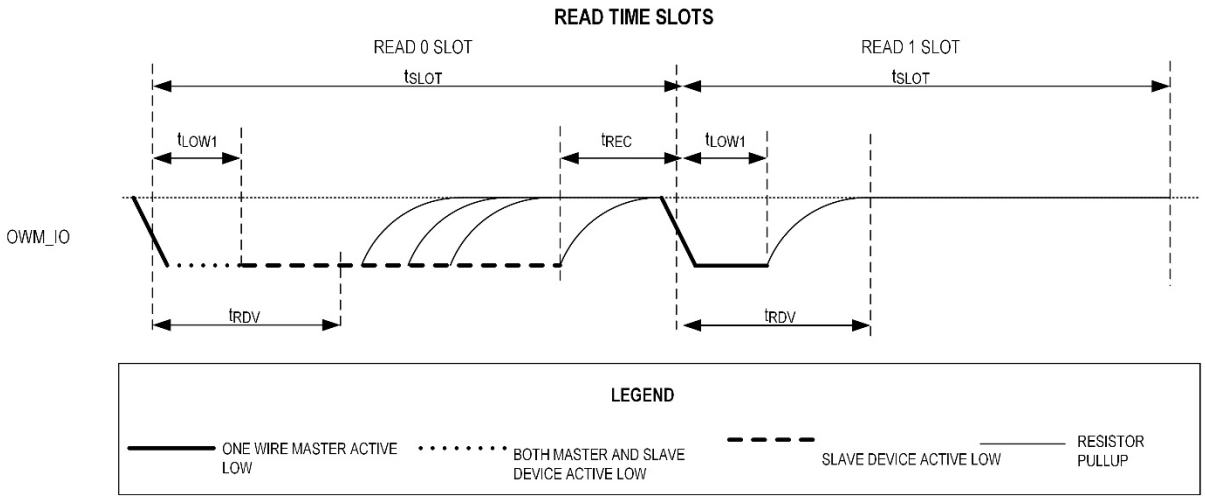
15.3.1.2.3 OWM Read Time Slot

As with all 1-Wire transactions, the master initiates all bit read time slots. Like the bit write time slots, the bit read time slot begins with a falling edge. From the master's perspective, this time slot is transmitted identically to the "Write 1 Bit" time slot shown in *Figure 15-3*. The master begins by transmitting a falling edge, holds the line low for a minimum-width period, and then releases the line.

The difference here is that instead of the slave sampling the line, the slave begins transmitting either a 0 (by holding the line low) or a 1 (by leaving the line to float high) after the initial falling edge. The master then samples the line to read the bit value that is transmitted by the slave device.

As an example, *Figure 15-4* shows a sequence in which the slave device transmits data back to the 1-Wire bus master upon request. Note that to transmit a 1 bit, the slave device does not need to do anything. It simply leaves the line alone (to float high) and waits for the next time slot. To transmit a 0 bit, the slave device holds the line low until the end of the time slot.

Figure 15-4: 1-Wire Read Time Slot



15.3.1.2.4 Standard Speed and Overdrive Speed

By default, all 1-Wire communications following reset begin at the lowest rate of speed (that is, standard speed). For 1-Wire devices that support it, it is possible for the OWM to increase the rate of communication from standard speed to overdrive speed by sending the appropriate command.

The protocols and time slots operate identically for standard and overdrive speeds. The difference comes in the widths of the time slots and pulses. The OWM automatically adjusts the timings based on the setting of the *OWM_CFG.overdrive* field.

If a 1-Wire slave device receives a standard speed reset pulse, it resets and reverts to standard speed communication. If the device is already communicating in overdrive mode, and it receives a reset pulse at the overdrive speed, it resets but remains in overdrive mode.

15.3.1.3 Network Layer

15.3.1.3.1 ROM Commands

Following the initial 1-Wire reset pulse on the bus, all slave 1-Wire devices are active, which means they are monitoring the bus for commands. Because the 1-Wire bus can have multiple slave devices present on the bus at any time, the OWM must go through a process (defined by the 1-Wire command protocol) to activate only the 1-Wire slave device it intends to communicate with and deactivate all others. This is the purpose of the ROM commands (network layer) shown in *Table 15-2*.

Table 15-2: 1-Wire ROM Commands

ROM Command	Hex Value
Read ROM	0x33
Match ROM	0x55
Search ROM	0xF0
Skip ROM	0xCC
Overdrive Skip ROM	0x3C
Overdrive Match ROM	0x69
Resume Communication	0xA5

The ROM command layer relies on the fact that all 1-Wire slave devices are assigned a globally unique, 64-bit ROM ID. This ROM ID value is factory programmed to ensure that no two 1-Wire slave devices have the same value.

15.3.1.3.2 ROM ID

Figure 15-5 is a visual representation of the 1-Wire ROM ID fields and shows the organization of the fields within the 64-bit ROM ID for a device.

Figure 15-5: 1-Wire ROM ID Fields

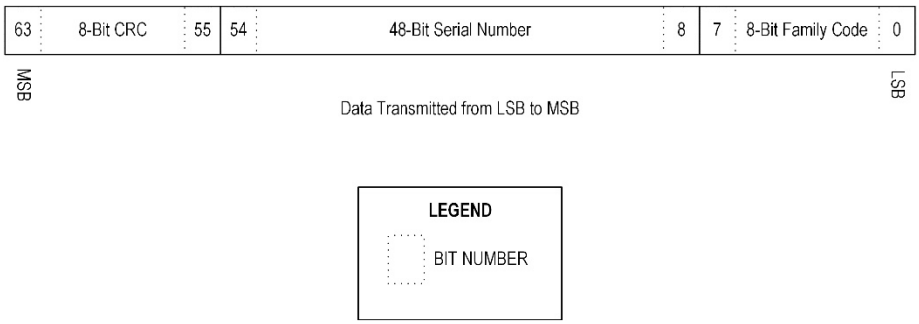


Table 15-3 provides a detailed description of each of the ROM ID fields.

Table 15-3: 1-Wire Slave Device ROM ID Field

Field	Bit Number	Description
Family code	0-7	This 8-bit value is used to identify the type of a 1-Wire slave device.
Unique ID	8-55	This 48-bit value is factory-programmed to give each 1-Wire slave device (within a given family code group) a globally unique identifier.
CRC	56-63	This is the 8-bit, 1-Wire CRC as defined in the *Book of iButton Standards*. The CRC is generated using the polynomial ($x^8 + x^5 + x^4 + 1$).

Note: For certain operations that consist only of writing from the OWM to the slave, it is technically possible for the master to communicate with more than one slave at a time on the same 1-Wire bus. For this to work, the exact same data must be transmitted to all slave devices, and any values read back from the slaves must either be identical as well or must be disregarded by the master device (because different slaves can attempt to transmit different values). The following descriptions assume, however, that the master is communicating with only one slave device at a time because this is the method normally used.

As explained above, the ROM ID contents play a key role in addressing and selecting devices on the 1-Wire bus. All devices except one are in an idle/inactive state after the Match ROM command or the Search ROM command is executed. They return to the active state only after receiving a 1-Wire reset pulse.

Devices with overdrive capability are distinguished from others by their family code and two additional ROM commands: Overdrive Skip ROM and Overdrive Match ROM. The first transmission of the ROM command itself takes place at the normal speed understood by all 1-Wire devices. After a device with overdrive capability is addressed and set into overdrive mode (that is, after the appropriate ROM command is received), further communication to that device must occur at overdrive speed. Because all deselected devices remain in the idle state if no reset pulse of regular duration is detected, even multiple overdrive components can reside on the same 1-Wire bus. A reset pulse of regular duration resets all 1-Wire devices on the bus and simultaneously sets all overdrive-capable devices back to the default standard speed.

15.3.2 Read ROM Command

The Read ROM command allows the OWM to obtain the 8-byte ROM ID of any slave device connected to the 1-Wire bus. Each slave device on the bus responds to this command by transmitting all eight bytes of its ROM ID value starting with the least significant byte (Family Code) and ending with the most significant byte (CRC).

Because this command is addressed to all 1-Wire devices on the bus, if more than one slave is present on the bus there is a data collision as multiple slaves attempt to transmit their ROM IDs at once. This condition is detectable by the OWM because the CRC value does not match the ROM ID value received. In this case, the OWM should reset the 1-Wire bus and select a single slave device on the bus to continue either by using the Match ROM command (if the ROM ID values are already known) or the Search ROM command (if the master has not yet identified some or all devices on the bus).

After the Read ROM command is complete, all slave devices on the 1-Wire bus are selected or active, and communication proceed to the Transport layer.

15.3.3 Skip ROM and Overdrive Skip ROM Commands

The Skip ROM command is used to activate all slave devices present on the 1-Wire bus regardless of their ROM ID. Normally, this command is used when only a single 1-Wire slave device is connected to the bus. After the Skip ROM command is complete, all slave devices on the 1-Wire bus are selected or active and communication proceeds to the Transport layer.

The Overdrive Skip ROM command operates in an identical manner except that running it also causes the receiving slave devices to shift communication speed from standard speed to overdrive speed. The Overdrive Skip ROM command byte itself (0x3C) is transmitted at standard speed. All subsequent communication is sent at overdrive speed.

15.3.4 Match ROM and Overdrive Match ROM Commands

The Match ROM command is used by the OWM to select one and only one slave 1-Wire device when the ROM ID of the device is already determined. When transmitting this command, the master sends the command byte (that is, 0x55 for

standard speed and 69h for overdrive speed) and then sends the entire 64-bit ROM ID for the device selected, least significant bit first.

During the transmission of the ROM ID by the master, all slave devices monitor the bus. As each bit is transmitted, each of the slave devices compares it against the corresponding bit of their ROM ID. If the bits match, the slave device continues to monitor the bus. If the bits do not match, the slave device transitions to the inactive state (waiting for a 1-Wire reset) and stops monitoring the bus.

At the end of the transmission, at most one slave device is active, which is the slave device whose ROM ID matched the ROM ID that was transmitted. All other slave devices are inactive. Communication then proceeds to the Transport layer for the device that was selected.

The Overdrive Match ROM command operates in an identical manner except that it also causes the slave device selected by the command to shift communication speed from standard speed to overdrive speed. The Overdrive Match ROM command byte (69h) and the 64-bit ROM ID bits are transmitted at standard speed. All subsequent communication is sent at overdrive speed.

15.3.5 Search ROM Command

The Search ROM command allows the OWM to determine the ROM ID values of all 1-Wire slave devices connected to the bus using an iterative search process. Each execution of the Search ROM command reveals the ROM ID of one slave device on the bus.

The operation of the Search ROM command resembles a combination of the Read ROM and Match ROM commands. First, all slaves on the bus transmit the least significant bit (Bit 0) of their ROM IDs. Next, all slaves on the bus transmit a complement of the same bit. By analyzing the two bits received, the master can determine if the Bit 0 values were 0 for all slaves, 1 for all slaves, or a combination of the two. Next, the master selects which slaves remain activated for the next step in the Search ROM process by transmitting the Bit 0 value for the slaves it selects. All slaves whose Bit 0 matches the value transmitted by the master remain active, while slaves with a different Bit 0 value go to the inactive state and do not participate in the remainder of the Search ROM command.

Next, the same process is followed for Bit 1, then Bit 2, and so on until the 63rd bit (most significant bit) of the ROM ID is transmitted. At this point, only one slave device remains active, and the master can either continue with communication at the Transport layer or issue a 1-Wire reset pulse to go back for another pass at the Search ROM command.

The *Book of iButton Standards* goes into more detail about the process used by the master to obtain ROM IDs of all devices on the 1-Wire bus using multiple executions of the Search ROM command. The algorithm resembles a binary tree search and is used regardless of how many devices are on the bus.

There is no overdrive equivalent version of the Search ROM command.

15.3.6 Search ROM Accelerator Operation

To allow the Search ROM command to process more quickly, the OWM module provides a special accelerator mode for use with the Search ROM command. This mode is activated by setting *OWM_CTRL_STAT.sra_mode* to 1.

When this mode is active, ROM IDs being processed by the Search ROM command are broken into 4-bit nibbles where the current 64-bit ROM ID varies with each pass through the search algorithm. Each 4-bit processing step is initiated by writing the 4-bit value to *OWM_DATA.tx_rx*. This causes the generation of twelve 1-Wire time slots by the OWM as each bit in the 4-bit value (starting with the LSB) results in a read of two bits (all active slaves transmitting bit N of their ROM IDs, then all active slaves transmitting the complement of bit N of their ROM ID), and then a write of a single bit by the OWM.

After the 4-bit processing stage is complete, the return value loaded into *OWM_DATA.tx_rx* consists of 8 bits. The low nibble (bits 0 through 3) contains the four discrepancy flags: one for each ID bit processed. If the discrepancy bit is set to 1, it means that either two slaves with differing ID bits in that position both responded (the 2 bits read were both zero), or no slaves responded (the 2 bits read were both 1). If the discrepancy bit is set to 0, then the 2 bits read were complementary (either 0, 1 or 1, 0) meaning there was no bus conflict.

In this way, at each step in the Search ROM command, the master either follows the ID of the responding slaves or deselects some of the slaves on the bus in case of a conflict. By the time the end of the 64-bit ROM ID is reached (the sixteenth 4-bit group processing step), the combination of all bits from the high nibbles of the received data are equal to the ROM ID of one of the slaves remaining on the bus. Subsequent passes through the Search ROM algorithm are used to determine additional slave ROM ID values until all slaves are identified. Refer to the *Book of iButton Standards* for a detailed explanation of the search function and possible variants of the search algorithm applicable to specific circumstances.

15.3.7 Resume Communication Command

If more than one 1-Wire slave device is on the bus, then the master must specify which one it wishes to communicate with each time a new 1-Wire command (starting with a reset pulse) begins. Using the commands discussed previously, this would normally involve sending the Match ROM command each time, which means the master must explicitly specify the full 64-bit ROM ID of the part it communicates with for each command.

The Resume Communication command provides a shortcut for this process by allowing the master to repeatedly select the same device for multiple commands without having to transmit the full ROM ID each time.

When the OWM selects a single device (using the Match ROM or Search ROM commands), an internal flag called the RC (for Resume Communication) flag is set in the slave device. (Only one device on the bus has this flag set at any one time; the Skip ROM command selects multiple devices, but the RC flag is not set by the Skip ROM command.)

When the master resets the 1-Wire bus, the RC flag remains set. At this point, it is possible for the master to send the Resume Communication command. This command does not have a ROM ID attached to it, but the device that has the RC flag set responds to this command by going to the active state while all other devices deactivate and drop off the 1-Wire bus.

Issuing any other ROM command clears the RC flag on all devices. So, for example, if a Match ROM command is issued for device A, its RC flag is set. The Resume Communication command can then be used repeatedly to send commands to device A. If a Match ROM command is then sent with the ROM ID of device B, the RC flag on device A will clear to 0, and the RC flag on device B is set.

15.4 1-Wire Operation

Once the OWM peripheral is correctly configured, then using the OWM peripheral to communicate with the 1-Wire network involves directing the OWM to generate the proper reset, read, and write operations to communicate with the 1-Wire slave devices used in a specific application.

The OWM manages the following 1-Wire protocol primitives directly in either Standard or Overdrive mode:

- 1-Wire bus reset (including detection of presence pulse from responding slave devices)
- Write single bit (a single write time slot)
- Write 8-bit byte, least significant bit first (eight write time slots)
- Read single bit (a single write-1 time slot)
- Read 8-bit byte, least significant bit first (eight write-1 time slots)
- Search ROM acceleration mode allowing the generation of four groups of three time slots (read, read, and write) from a single 4-bit register write to support the Search ROM command

15.4.1 Resetting the OWM

The first step in any 1-Wire communication sequence is to reset the 1-Wire bus. To direct the OWM module to complete a 1-Wire reset, write *OWM_CTRL_STAT.start_ow_reset* to 1. This generates a reset pulse and checks for a replying presence pulse from any connected slave devices.

Once the reset time slot is complete, the *OWM_CTRL_STAT.start_ow_reset* field is automatically cleared to zero. Then, the interrupt flag *OWM_INTFL.ow_reset_done* is set to 1 by the hardware. This flag must be cleared by writing a 1 bit to the flag.

If a presence pulse is detected on the 1-Wire bus during the reset sequence (that should normally be the case unless no 1-Wire slave devices are present on the bus), the *OWM_CTRL_STAT.presence_detect* flag is also set to 1. This flag does not result in the generation of an interrupt.

15.5 1-Wire Data Reads

15.5.1 Reading a Single Bit Value from the 1-Wire Bus

The procedure for reading a single bit is like the procedure for writing a single bit because the operation is completed by writing a 1 bit that the slave device either leaves unchanged (to transmit a 1 bit) or overrides by forcing the line low (to transmit a 0 bit).

To read a single bit value from the 1-Wire Bus, complete the following steps:

1. Set *OWM_CFG.single_bit_mode* to 1. This setting causes the OWM to transmit/receive a single bit of data at a time instead of the default 8 bits.
2. Write *OWM_DATA.tx_rx* to 1. Only bit 0 of this field is used in this instance; the other bits in the field are ignored. Writing to the *OWM_DATA* register initiates the read of the bit on the 1-Wire bus.
3. Once the single-bit transmission is complete, the hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.
4. As the hardware shifts the bit value out, it also samples the value returned from the slave device. Once this value is ready to read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.
5. Read *OWM_DATA.tx_rx* (only bit 0 is used) to determine the value returned by the slave device. Note that if no slave devices are present or the slaves are not communicating with the master, bit 0 remains set to 1.

15.5.2 Reading an 8-Bit Value from the 1-Wire Bus

The procedure for reading an 8-bit byte is like the procedure for writing an 8-bit byte because the operation is completed by writing eight 1 bits that the slave device either leaves unchanged (to transmit 1 bits) or overrides by forcing the line low (to transmit 0 bits).

1. Set *OWM_CFG.single_bit_mode* to 0. This setting causes the OWM to transmit/receive in the default 8-bit mode.
2. Write *OWM_DATA.tx_rx* to 0x0FFh.
3. Once the 8-bit transmission completes, the hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.
4. As the hardware shifts the bit values out, it also samples the values returned from the slave device. Once the full 8-bit value is ready to be read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.
5. Read *OWM_DATA.tx_rx* to determine the 8-bit value returned by the slave device. *Note that if no slave devices are present or the slave devices are not communicating with the master, the return value 0x0FF is the same as the transmitted value.*

15.6 Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 15-4: OWM Register Summary

Offset	Register	Description
[0x0000]	*OWM_CFG*	OWM Configuration Register
[0x0004]	*OWM_CLK_DIV_1US*	OWM Clock Divisor Register
[0x0008]	*OWM_CTRL_STAT*	OWM Control/Status Register
[0x000C]	*OWM_DATA*	OWM Data Buffer Register
[0x0010]	*OWM_INTFL*	OWM Interrupt Flag Register
[0x0014]	*OWM_INTEN*	OWM Interrupt Enable Register

15.6.1 Register Details

Table 15-5: OWM Configuration Register

OWM Configuration Register		OWM_CFG		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7	int_pullup_enable	R/W	0	**Internal Pullup Enable** Set this field to enable the internal pullup resistor. 0: Internal pullup disabled. 1: Internal pullup enabled.
6	overdrive	R/W	0	**Overdrive Enable** Set this field to 1 to enable overdrive mode for 1-Wire communications. Clearing this field sets 1-Wire communications to standard speed. 0: Overdrive mode disabled, standard speed mode. 1: Overdrive mode enabled.
5	single_bit_mode	R/W	0	**Bit Mode Enable** When set to 1, only a single bit at a time is transmitted and received (LSB of *OWM_DATA*) rather than the whole byte. 0: Byte mode enabled, single bit mode disabled. 1: Single bit mode enabled, byte mode disabled.
4	ext_pullup_enable	R/W	0	**External Pullup Enable** Enables external FET pullup when the 1-Wire master is idle. FET is designed to pull the wire high regardless of its enable state (that is, high or low). Idle means the 1-Wire master is idle, and there are no 1-Wire accesses in progress. 0: External pullup pin is not driven to high. 1: External pullup pin is driven high when the 1-Wire bus is idle, actively pulling the 1-Wire IO high.
3	ext_pullup_mode	R/W	0	**External Pullup Mode** Provides an extra output to control an external pullup. For long wires, a pullup resistor strong enough to pull the wire high in a reasonable amount of time might need to be so strong that it would be difficult to drive the line low. In this case, implement an external FET to actively drive the wire high for a brief amount of time. Then, let the resistor keep the line high.

OWM Configuration Register		OWM_CFG		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
2	bit_bang_en	R/W	0	**Bit-Bang Mode Enable** Enable bit-bang control of the OWM_IO pin. If this bit is set to 1, *OWM_CTRL_STAT.bit_bang_oe* controls the state of the OWM_IO pin. 0: Bit-bang mode disabled. 1: Bit-bang mode enabled.
1	force_pres_det	R/W	1	**Presence Detect Force** Setting this bit to 1 drives the OWM_IO pin low during presence detection. Use this bit field to prevent a large number of 1-Wire slaves on the bus from all responding at different times, which might cause ringing. When this bit is set to 1, the *OWM_CTRL_STAT.presence_detect* bit is always set as the result of a 1-Wire reset even if no slave devices are present on the bus. 0: OWM_IO pin floats during presence detection portion of a 1-Wire reset. 1: OWM_IO pin is driven low during presence detection portion of a 1-Wire reset.
0	long_line_mode	R/W	0	**Long Line Mode Enable** Selects alternate timings for 1-Wire communication. The recommended setting depends on the length of the wire. For lines less than 40 meters, 0 should be used. Setting this bit to 0 leaves the write one release, the data sampling, and the time-slot recovery times at approximately 5µs, 15µs, and 7µs, respectively. Setting this bit to 1 enables long line mode timings during standard mode communications. This mode moves the write one release, the data sampling, and the time-slot recovery times out to approximately 8µs, 22µs, and 14µs, respectively. 0: Standard operation for lines less than 40 meters. 1: Long Line mode enabled.

Table 15-6: OWM Clock Divisor Register

OWM Clock Divisor		OWM_CLK_DIV_1US		[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R	0	**Reserved**
7:0	divisor	R/W	0	**OWM Clock Divisor** Divisor for the OWM peripheral clock. The target is to achieve a 1MHz clock. See the *Clock Configuration* section for details. 0x00: OWM clock disabled. 0x01: $f_{owmclk} = \frac{f_{PCLK}}{1}$ 0x02: $f_{owmclk} = \frac{f_{PCLK}}{2}$ …: … 0xFF: $f_{owmclk} = \frac{f_{PCLK}}{255}$

Table 15-7: OWM Control Status Register

OWM Control Status		OWM_CTRL_STAT		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

OWM Control Status			OWM_CTRL_STAT	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
7	presence_detect	RO	0	**Presence Detect Flag** Set to 1 when a presence pulse is detected from one or more slaves during the 1-Wire reset sequence. 0: No presence detect pulse during previous 1-Wire reset sequence. 1: Presence detect pulse on bus during previous 1-Wire reset sequence.
6:5	-	RO	0	**Reserved**
4	od_spec_mode	RO	0	**Overdrive Spec Mode** Returns the version of the overdrive spec.
3	ow_input	RO	-	**OWM_IN State** Returns the current logic level on the OWM_IO pin. 0: OWM_IO pin is low. 1: OWM_IO pin is high.
2	bit_bang_oe	R/W	0	**OWM Bit-Bang Output** When bit-bang mode is enabled (*OWM_CFG.bit_bang_en* = 1), this bit sets the state of the OWM_IO pin. Setting this bit to 1 drives the OWM_IO pin low. Setting this bit to 0 releases the line, allowing the OWM_IO pin to be pulled high by the pullup resistor or held low by a slave device. 0: OWM_IO pin floating. 1: Drive OWM_IO pin to low state.
1	sra_mode	R/W	0	**Search ROM Accelerator Enable** Enable Search ROM Accelerator mode. This mode is used to identify slaves and their addresses attached to the 1-Wire bus. 0: Search ROM accelerator mode disabled. 1: Search ROM accelerator mode enabled.
0	start_ow_reset	R/W	0	**Start 1-Wire Reset Pulse** Write 1 to start a 1-Wire reset sequence. Automatically cleared by the OWM hardware when the reset sequence is complete. 0: 1-Wire reset sequence complete or inactive. 1: Start a 1-Wire reset sequence.

Table 15-8: OWM Data Buffer Register

OWM Data			OWM_DATA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7:0	tx_rx	R/W	0	**OWM Data Field** Writing to this field sets the transmit data and initiates a 1-Wire data transmit cycle. Reading from this field returns the data received by the master during the last 1-Wire data transmit cycle.

Table 15-9: OWM Interrupt Flag Register

OWM Interrupt Flag			OWM_INTFL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	R/W	0	**Reserved**
4	line_low	R/W1C	0	**Line Low Flag** If this flag is set, the OWM_IO pin was in a low state. Write 1 to clear this flag.

OWM Interrupt Flag		OWM_INTFL		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3	line_short	R/W1C	0	**Line Short Flag** The OWM hardware detected a short on the OWM_IO pin. Write 1 to clear this flag.
2	rx_data_ready	R/W1C	0	**RX Data Ready** Data received from the 1-Wire bus and is available in the *OWM_DATA.tx_rx* field. Write 1 to clear this flag. 0: RX data not available. 1: Data received and is available in the *OWM_DATA.tx_rx* field.
1	tx_data_empty	R/W1C	0	**TX Empty** The OWM hardware automatically sets this interrupt flag when the data transmit is complete. Write 1 to clear this flag. 0: Either no data was sent or the data in the *OWM_DATA.tx_rx* field has not completed transmission. 1: Data in the *OWM_DATA.tx_rx* field was transmitted.
0	ow_reset_done	R/W1C	0	**Reset Complete** This flag is set when a 1-Wire reset sequence completes. To start a 1-Wire reset sequence, see *OWM_CTRL_STAT.start_ow_reset*. Write 1 to clear this flag. 0: 1-Wire reset sequence not complete or bus idle. 1: 1-Wire reset sequence complete.

Table 15-10: OWM Interrupt Enable Register

OWM Interrupt Enable		OWM_INTEN		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	line_low	R/W	0	**Line Low Interrupt Enable** I/O pin low detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
3	line_short	R/W	0	**Line Short Interrupt Enable** I/O pin short detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
2	rx_data_ready	R/W	0	**Receive Data Ready Interrupt Enable** RX data ready interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
1	tx_data_empty	R/W	0	**Transmit Data Empty Interrupt Enable** TX data empty interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
0	ow_reset_done	R/W	0	**1-Wire Reset Sequence Complete Interrupt Enable** 1-Wire reset sequence completed. 0: Interrupt disabled. 1: Interrupt enabled.

16. Real-Time Clock (RTC)

16.1 Overview

The Real-Time Clock (RTC) is a 32-bit binary timer that keeps the time of day up to 136 years. It provides time-of-day and sub-second alarm functionality in the form of system interrupts.

The RTC operates on an external 32.768kHz time base. It can be generated from the internal crystal oscillator driving an external 32.768kHz crystal between the 32KIN and 32KOUT pins, or a 32.768kHz square wave driven directly into the 32KIN pin. Refer to the data sheet for the required electrical characteristics of the external crystal.

A user-configurable, digital frequency trim is provided for applications requiring higher accuracy.

The 32-bit seconds register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-seconds field.

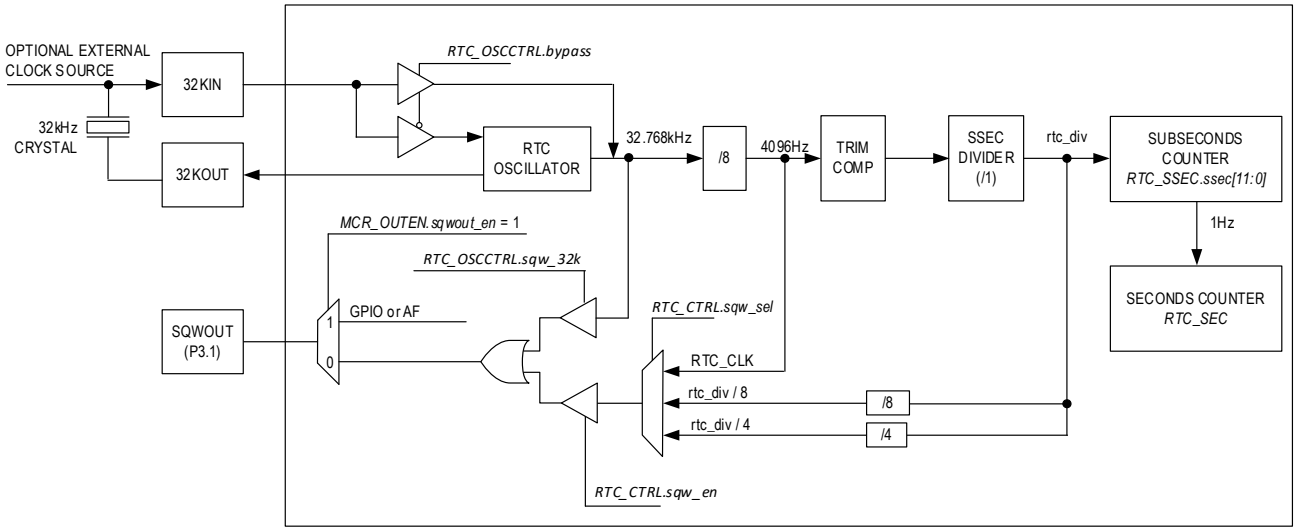
Two alarm functions are provided:

1. A programmable time-of-day alarm provides a single event, alarm timer using the *RTC_TODA* alarm register, *RTC_SEC* register, and *RTC_CTRL.tod_alarm_ie* field.
2. A programmable sub-second provides a recurring alarm using the RTC sub-second alarm register, *RTC_SSECA*, and is *RTC_CTRL.ssec_alarm* field

The RTC is powered in the always-on domain or if applicable, while there is a valid voltage on the V_{COREA} device pin.

Disabling the RTC, *RTC_CTRL.en* cleared to 0, stops incrementing the *RTC_SSEC*, *RTC_SEC*, and the internal RTC sub-second counter, but preserves their current values. The 32kHz oscillator is not affected by the *RTC_CTRL.en* field. While the RTC is enabled, *RTC_CTRL.en* set to 1, the *RTC_TRIM.vrtc_tmr* field is also incremented every 32 seconds.

Figure 16-1: MAX32655 RTC Block Diagram (12-bit Sub-Second Counter)



16.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm register details and description are shown in *Table 16-1*.

Table 16-1: RTC Seconds, Sub-Seconds, Time-of-Day Alarm and Sub-Seconds Alarm Register Details

Register	Width (bits)	Counter Increment	Minimum	Maximum	Description
RTC_SEC	32	1 second	1 second	136 years	Seconds Counter Register
RTC_SSEC	12	244 µs ($\frac{1}{4096Hz}$)	244 µs	1 second	Sub-Seconds Counter Register
RTC_TODA	20	1 second	1 second	12 days	Time-of-Day Alarm Register
RTC_SSECA	32	244 µs ($\frac{1}{4096Hz}$)	244 µs	12 days	Sub-Second Alarm Register

16.3 Register Access Control

Access protection mechanisms prevent the software from accessing critical registers and fields while RTC while the hardware is updating them. Monitoring the *RTC_CTRL.busy* and *RTC_CTRL.rdy* fields allows the software to determine when it is safe to write to registers and when registers return valid results.

Table 16-2: RTC Register Access

Register	Field	Read Access	Write Access	*RTC_CTRL.busy* = 1 during writes	Description
RTC_SEC	All	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Seconds Counter
RTC_SSEC	*ssec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Sub-Seconds Counter
RTC_TODA	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.tod_alarm_ie* = 0 *RTC_CTRL.en = 0*	Y	Time-of-Day Alarm
RTC_SSECA	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ssec_alarm_ie* = 0 *RTC_CTRL.en = 0*	Y	Sub-Second Alarm
RTC_TRIM	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	Trim
RTC_OSCCTRL	All	Always	*RTC_CTRL.wr_en* = 1	N	Oscillator Control
RTC_CTRL	*en*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	RTC Enable field
	All other bits	Always	††	Y	

† See the *RTC_SEC and RTC_SSEC Read Access Control* section for details.

†† See the *RTC_CTRL.busy* field description for limitations on specific bits.

16.3.1 RTC_SEC and RTC_SSEC Read Access Control

The software reads of the *RTC_SEC* and *RTC_SSEC* registers return invalid results if the read operation occurs on the same cycle that the register is being updated by the hardware (*RTC_CTRL.rdy* = 0). To avoid this, the hardware sets the *RTC_CTRL.rdy* field to 1 for 120µs when the *RTC_SEC* and *RTC_SSEC* registers are valid and readable by the software.

Alternately, the software can set the *RTC_CTRL.rd_en* field to 1 to allow asynchronous reads of both the *RTC_SEC* and *RTC_SSEC* registers.

Three methods are available to ensure valid results when reading *RTC_SEC* and *RTC_SSEC*:

1. The software clears the *RTC_CTRL.rdy* field to 0.
   a. The software polls the *RTC_CTRL.rdy* field until it reads 1 before reading the registers.
   b. The software must read the *RTC_SEC* and *RTC_SSEC* registers within 120µs to ensure valid register data.
2. The software sets the *RTC_CTRL.rdy_ie* field to 1 to generate an RTC interrupt when the hardware sets the *RTC_CTRL.rdy* field to 1.
   a. The software must service the RTC interrupt and read the *RTC_SEC* register and/or the *RTC_SSEC* register while the *RTC_CTRL.rdy* field is 1 to ensure valid data. This avoids the overhead associated with polling the *RTC_CTRL.rdy* field.
3. The software sets the *RTC_CTRL.rd_en* field to 1 enabling asynchronous reads of both the *RTC_SEC* register and the *RTC_SSEC* register.
   a. The software must read consecutive identical values of each of the *RTC_SEC* register and the *RTC_SSEC* register to ensure valid data.

16.3.2 RTC Write Access Control

The read-only status field *RTC_CTRL.busy* is set to 1 by the hardware following a software instruction that writes to specific registers. The bit remains 1 while the software updates are being synchronized into the RTC. The software should not write to any of the registers until the hardware indicates the synchronization is complete by clearing *RTC_CTRL.busy* to 0.

16.4 RTC Alarm Functions

The RTC provides time-of–day and sub-second interval alarm functions. The time-of-day alarm is implemented by matching the count values in the counter register with the value stored in the alarm register. The sub-second interval alarm provides an auto-reload timer driven by the trimmed RTC clock source.

16.4.1 Time-of-Day Alarm

Program the RTC time-of-day alarm register (*RTC_TODA*) to configure the time-of-day-alarm. The alarm triggers when the value stored in *RTC_TODA.tod_alarm* matches the lower 20 bits of the *RTC_SEC* seconds count register. This allows programming the time-of-day-alarm to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. Disable the time-of-day alarm before changing the *RTC_TODA.tod* field.

When the alarm occurs, a single event sets the time-of-day alarm interrupt flag (*RTC_CTRL.tod_alarm*) to 1.

Setting the *RTC_CTRL.tod_alarm* bit to 1 in the software results in an interrupt request to the processor if the alarm time-of-day interrupt enable (*RTC_CTRL.tod_alarm_ie*) bit is set to 1, and the RTC's system interrupt enable is set.

16.4.2 Sub-Second Alarm

The *RTC_SSECA* and *RTC_CTRL.ssec_alarm_ie* field control the sub-second alarm. Writing *RTC_SSECA* sets the starting value for the sub-second alarm counter. Writing the sub-second alarm enable (*RTC_CTRL.ssec_alarm_ie*) bit to 1 enables the sub-second alarm. Once enabled, an internal alarm counter begins incrementing from the *RTC_SSECA* value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, the hardware sets the *RTC_CTRL.ssec_alarm* bit triggering the alarm. At the same time, the hardware also reloads the counter with the value previously written to *RTC_SSECA.rssa*.

Disable the sub-second interval alarm, *RTC_CTRL.ssec_alarm_ie*, prior to changing the interval alarm value, *RTC_SSECA*.

The delay (uncertainty) associated with enabling the sub-second alarm is up to one period of the sub-second clock. This uncertainty is propagated to the first interval alarm. Thereafter, if the interval alarm remains enabled, the alarm triggers after each sub-second interval as defined without the first alarm uncertainty because the sub-second alarm is an auto-

reload timer. Enabling the sub-second alarm with the sub-second alarm register set to 0 (*RTC_SSECA* = 0) results in the maximum sub-second alarm interval.

16.4.3 RTC Interrupt and Wakeup Configuration

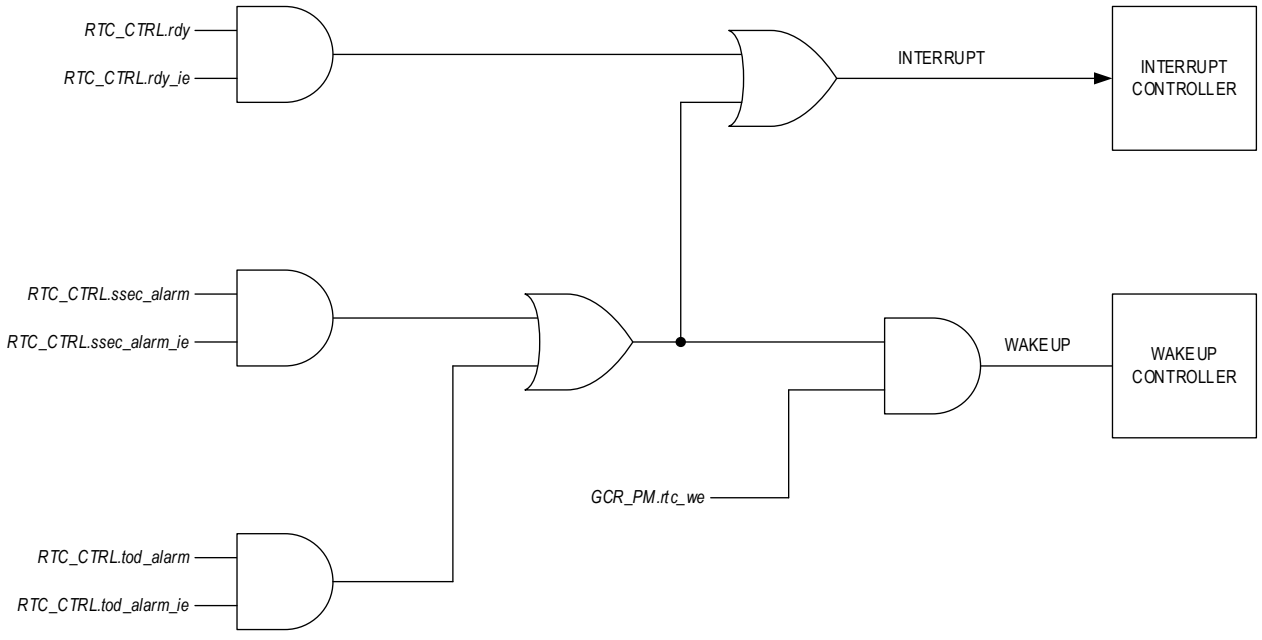
The following are a list of conditions that, when enabled, generate an RTC interrupt.

1. Time-of-day alarm
2. Sub-second alarm
3. *RTC_CTRL.rdy* field asserted high, signaling read access permitted

The RTC can be configured so the time-of-day and sub-second alarms are a wakeup source for exiting the following low-power modes:

1. *BACKUP*
2. *DEEPSLEEP*
3. *UPM*
4. *SLEEP*

Figure 16-2: RTC Interrupt/Wakeup Diagram Wakeup Function



Use this procedure to enable the RTC as a wakeup source:

1. Configures the RTC interrupt enable bits so one or more interrupt conditions generate an RTC interrupt.
2. Create a RTC interrupt handler function and register the address of the RTC_IRQHandler using the NVIC.
3. Set the *GCR_PM.rtc_we* field to 1 to enable system wakeup by the RTC.
4. Enter the desired low-power mode. See *Operating Modes* for details.

16.4.4 Square Wave Output

The RTC can output a 50% duty cycle square wave signal derived from the 32kHz oscillator on a selected device pin. See *Table 16-3* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 16-3: MAX32655 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin	P3.1: SQWOUT	*MCR_OUTEN.sqwout_en* = 1
Frequency Selection	1Hz (Compensated)	*RTC_CTRL.sqw_sel* = 0
	512Hz (Compensated)	*RTC_CTRL.sqw_sel* = 1
	4kHz	*RTC_CTRL.sqw_sel* = 2
	32kHz	*RTC_OSCCTRL.32k_out* = 1
Enable Frequency Output	1Hz (Compensated)	*RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.32k_out* = 0
	512Hz (Compensated)	*RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.32k_out* = 0
	4kHz	*RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.32k_out* = 0
	32kHz	*RTC_OSCCTRL.32k_out* = 1

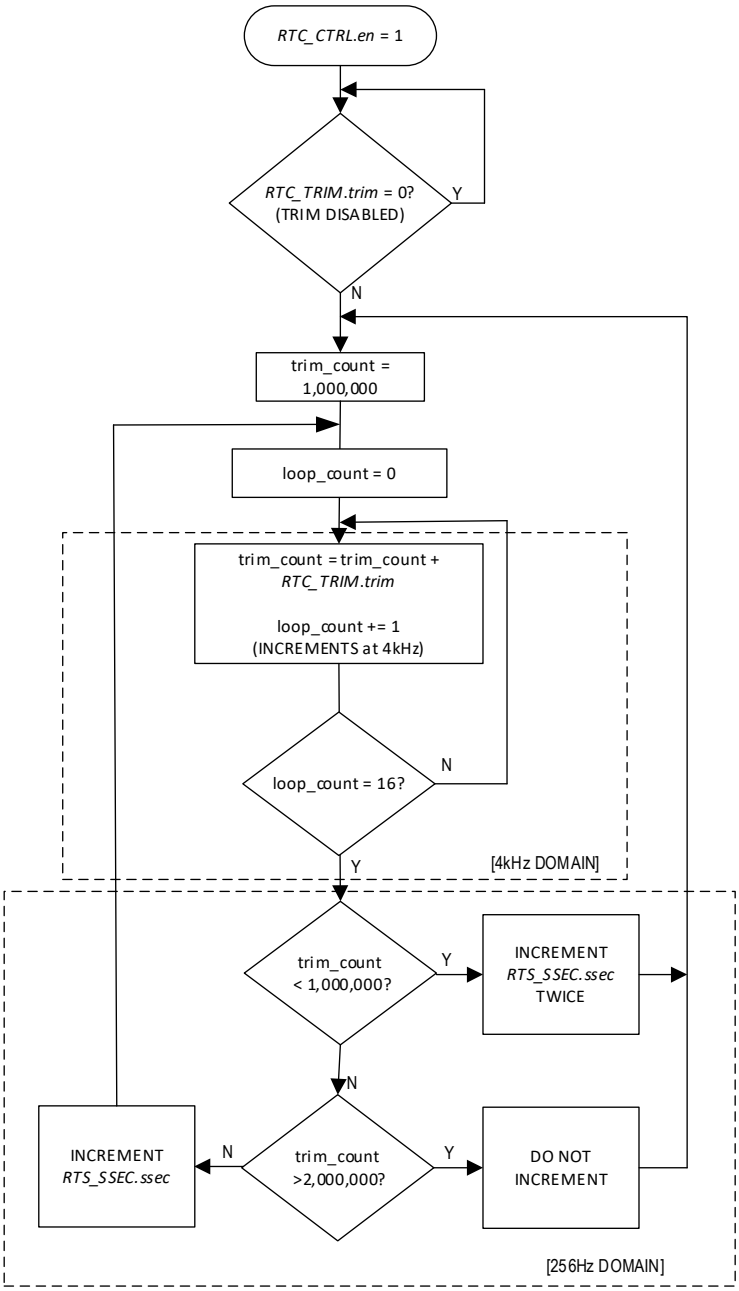
Use the following software procedure to generate and output the square wave:

1. Select the desired frequency to output:
   a. Set the field *RTC_CTRL.sqw_sel* to 0 for a 1Hz compensated output frequency, or
   b. Set the field *RTC_CTRL.sqw_sel* to 1 for a 512Hz compensated output frequency, or
   c. Set the field *RTC_CTRL.sqw_sel* to 2 for a 4kHz output frequency, or
   d. Set the field *RTC_OSCCTRL.32k_out* to 1 for the 32kHz frequency output
2. Enable the system level output pin by setting the Output Pin as shown in *Table 16-3*.
3. If the selected frequency is 1Hz, 512Hz, or 4kHz, set the *RTC_CTRL.sqw_en* field to 1 to output the selected output frequency.

16.5 RTC Calibration

A digital trim facility provides the ability to compensate for RTC inaccuracies of up to ± 127ppm when compared against an external reference clock. The trimming function utilizes an independent dedicated timer that increments the sub-second register based on a user-supplied, twos compliment value in the *RTC_TRIM* register as shown in *Figure 16-3*.

Figure 16-3: Internal Implementation of 4kHz Digital Trim



Complete the following steps to perform an RTC calibration:

1. The software must configure and enable one of the compensated calibration frequencies as described in section *Square Wave Output*.
2. Measure the frequency on the square wave output pin and compute the deviation from an accurate reference clock.
3. Clear the *RTC_CTRL.rdy* field to 0.
4. Wait for the *RTC_CTRL.rdy* to be set to 1 by the hardware:
   a. Set the *RTC_CTRL.rdy_ie* to 1 to generate an interrupt when the *RTC_CTRL.rdy* field is set to 1, or

b. Poll the *RTC_CTRL.rdy* field until it reads 1.

5. Poll the *RTC_CTRL.busy* field until it reads 0 to allow any active operations to complete.
6. Set the *RTC_CTRL.wr_en* field to 1 to allow access to the *RTC_TRIM.trim* field.
7. Write a trim value to the *RTC_TRIM.trim* field to correct for measured inaccuracy.
8. Poll the *RTC_CTRL.busy* field until it reads 0
9. Clear the *RTC_CTRL.wr_en* field to 0.
10. Repeat the process as needed until the desired accuracy is achieved.

16.6 Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise all fields are reset on a system reset, soft reset, POR, and the peripheral-specific reset.

Table 16-4: RTC Register Summary

Offset	Register	Description
[0x0000]	*RTC_SEC*	RTC Seconds Counter Register
[0x0004]	*RTC_SSEC*	RTC Sub-Second Counter Register
[0x0008]	*RTC_TODA*	RTC Time-of-Day Alarm Register
[0x000C]	*RTC_SSECA*	RTC Sub-Second Alarm Register
[0x0010]	*RTC_CTRL*	RTC Control Register
[0x0014]	*RTC_TRIM*	RTC 32KHz Oscillator Digital Trim Register
[0x0018]	*RTC_OSCCTRL*	RTC 32KHz Oscillator Control Register

16.6.1 Register Details

Table 16-5: RTC Seconds Counter Register

RTC Seconds Counter			RTC_SEC	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	sec	R/W	0	**Seconds Counter** This register is a binary count of seconds.

Table 16-6: RTC Sub-Second Counter Register (12-bit)

RTC Sub-Seconds Counter			RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:0	ssec	R/W	0	**Sub-Seconds Counter (12-bit)** *RTC_SEC* increments when this field rolls from 0x0FFF to 0x0000

Table 16-7: RTC Time-of-Day Alarm Register

RTC Time-of-Day Alarm			RTC_TODA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	tod_alarm	R/W	0	**Time-of-Day Alarm** Sets the time-of-day alarm from 1 second up to 12-days. When this field matches *RTC_SEC[19:0]*, an RTC system interrupt is generated.

Table 16-8: RTC Sub-Second Alarm Register

RTC Sub-Second Alarm			RTC_SSECA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ssec_alarm	R/W	0	**Sub-second Alarm (4KHz)** Sets the starting and reload value of the internal sub-second alarm counter. The internal counter increments and generates an alarm when the internal counter rolls from 0xFFFF FFFF to 0x0000 0000.

Table 16-9: RTC Control Register

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	wr_en	R/W	0*	**Write Enable** This field controls access to the *RTC_TRIM* register, the RTC enable (*RTC_CTRL.en*) fields. 1: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are allowed. 0: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are ignored. **Note: Reset on System Reset, Soft Reset, and GCR_RST0.rtc assertion.*
14	rd_en	R/W	0	**Asynchronous Counter Read Enable** Set this field to 1 to allow direct read access of the *RTC_SEC* and *RTC_SSEC* registers without waiting for *RTC_CTRL.rdy.* Multiple consecutive reads of *RTC_SEC* and *RTC_SSEC* must be executed until two consecutive reads are identical to ensure data accuracy. 0: *RTC_SEC* and *RTC_SSEC* registers are synchronized and should only be accessed while *RTC_CTRL.rdy*= 1. 1: *RTC_SEC* and *RTC_SSEC* registers are asynchronous and requires software interaction to ensure data accuracy.
13:11	-	RO	0	**Reserved**
10:9	sqw_sel	R/W	0*	**Frequency Output Select** Selects the RTC-derived frequency to output on the square wave output pin. See *Table 16-3* for configuration details. 0: 1Hz (Compensated) 1: 512Hz (Compensated) 2: 4kHz **Note: Reset on POR only.*
8	sqw_en	R/W	0*	**Square Wave Output Enable** Enables the square wave output. See *Table 16-3* for configuration details. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*
7	ssec_alarm	R/W	0*	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wakeup source for the device. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. **Note: Reset on POR only.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
6	tod_alarm	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by the hardware when a time-of-day alarm occurs. 0: No time-of-day alarm interrupt pending. 1: Time-of-day interrupt pending. **Note: Reset on POR only.*
5	rdy_ie	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. **Note: Reset on system reset, soft reset, and GCR_RST0.rtc assertion.*
4	rdy	R/W0	0*	**RTC Ready** This bit is set to 1 for 120µs by the hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers has occurred. The software should read *RTC_SEC* and *RTC_SSEC* while this hardware bit is set to 1. The software can clear this bit at any time. An RTC interrupt is generated if *RTC_CTRL.rdy_ie* = 1. 0: Software reads of *RTC_SEC* and *RTC_SSEC* are invalid. 1: Software reads of *RTC_SEC* and *RTC_SSEC* are valid. **Note: Reset on System Reset, Soft Reset, and GCR_RST0.rtc assertion.*
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by the hardware while a register update is in progress when the software writes to the following registers: • *RTC_SEC* • *RTC_SSEC* • *RTC_TRIM* The following fields cannot be written when this field is set to 1: • *RTC_CTRL.en* • *RTC_CTRL.tod_alarm_ie* • *RTC_CTRL.ssec_alarm_ie* • *RTC_CTRL.rdy_ie* • *RTC_CTRL.tod_alarm* • *RTC_CTRL.ssec_alarm* • *RTC_CTRL.sqw_en* • *RTC_CTRL.ft* • *RTC_CTRL.rd_en* This field is automatically cleared by the hardware when the update is complete. Software should poll this field until it reads 0 after changing the *RTC_SEC*, *RTC_SSEC*, or *RTC_TRIM* register, prior to making any other RTC register modifications. 0: RTC not busy 1: RTC busy **Note: Reset on POR only.*
2	ssec_alarm_ie	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
1	tod_alarm_ie	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
0	en	R/W	0*	**Real-Time Clock Enable** The RTC write enable (*RTC_CTRL.wr_en*) bit must be set and RTC Busy (*RTC_CTRL.busy*) must read 0 before writing to this field. After writing to this bit, check the *RTC_CTRL.busy* flag for 0 to determine when the RTC synchronization is complete. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*

Table 16-10: RTC 32KHz Oscillator Digital Trim Register

RTC 32KHz Oscillator Digital Trim			RTC_TRIM	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vrtc_tmr	R/W	0*	**VRTC Time Counter** The hardware increments this field every 32 seconds while the RTC is enabled. **Note: Reset on POR only.*
7:0	trim	R/W	0*	**RTC Trim** This field specifies the 2s complement value of the trim resolution. Each increment or decrement of the field adds or subtracts 1ppm at each 4kHz clock value with a maximum correction of ± 127ppm. **Note: Reset on POR only.*

Table 16-11: RTC 32KHz Oscillator Control Register

RTC Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved**
5	sqw_32k	R/W	0	**RTC Square Wave Output** 0: Disabled. 1: Enables the 32kHz oscillator output or the external clock source is output on square wave output pin. See *Table 16-3* for configuration details. **Note: Reset on POR only.*
4	bypass	R/W	0	**RTC Crystal Bypass** This field disables the RTC oscillator and allows an external clock source to drive the 32KIN pin. 0: Disable bypass. RTC time base is external 32kHz crystal. 1: Enable bypass. RTC time base is external square wave driven on 32KIN. **Note: Reset on POR only.*
3:0	-	DNM	9	**Reserved Do Not Modify**

17. Timers (TMR/LPTMR)

Multiple 32-bit and dual 16-bit reloadable timers are provided.

The features include:

- Operation as a single 32-bit counter or single/dual 16-bit counter(s).
- Programmable clock prescaler with values from 1 to 4096.
- Non-overlapping Pulse Width Modulated (PWM) output generation with configurable off-time.
- Capture, compare, and capture/compare capability.
- Timer input and output signals available, mapped as alternate functions.
- Configurable input pin for event triggering, clock gating, or capture signal.
- Timer output pin for event output and PWM signal generation.
- Multiple clock source options.

Instances denoted as LPTMR, shown in *Table 17-1*, are configurable to operate in any of the low-power modes and wake the device from the low-power modes to *ACTIVE*.

Each timer supports multiple operating modes:s

- One-shot: the timer counts up to terminal value then halts.
- Continuous: the timer counts up to terminal value then repeats.
- Counter: the timer counts input edges received on the timer input pin.
- PWM / PWM differential.
- Capture: the timer captures a snapshot of the current timer count when the timer's input edge transitions.
- Compare: the timer pin toggles when the timer's count exceeds the terminal count.
- Gated: the timer increments only when the timer's input pin is asserted.
- Capture/Compare: the timer counts when the timer input pin is asserted; the timer captures the timer's count when the input pin is deasserted.

17.1 Instances

Instances of the peripheral are listed in *Table 17-1*. Both the TMR and LPTMR are functionally similar, so for convenience all timers are referenced as TMR. The LPTMR instances can function while the device is in certain low-power modes.

Refer to the device data sheet for frequency limitations for external clock sources, if available. Refer to the data sheet for I/O signal configurations and alternate functions for each timer instance.

Table 17-1: MAX32655 TMR/LPTMR Instances

Instance	Register Access Name	Cascade 32-Bit Mode	16-Bit Mode	Operating Modes	CLK0	CLK1	CLK2	CLK3
TMR0	TMR0	Yes	Dual	*ACTIVE SLEEP LPM*	PCLK	ISO	IBRO	ERTCO
TMR1	TMR1							
TMR2	TMR2							
TMR3	TMR3							
LPTMR0	TMR4	No	Single	*ACTIVE SLEEP LPM*	IBRO	ERTCO	INRO	LPTMR0_CLK P2.6 (AF1)
				UPM	N/A	N/A	ERTCO	INRO
LPTMR1	TMR5	No	Single	*ACTIVE SLEEP LPM*	IBRO	$\frac{IBRO}{8}$	INRO	LPTMR1_CLK P2.7 (AF1)
				UPM	N/A	N/A	ERTCO	INRO

Table 17-2: MAX32655 TMR/LPTMR Instances Capture Events

Instance	Capture Event 0	Capture Event 1	Capture Event 2	Capture Event 3
TMR0	Timer Input Pin	TMR0A_IOA	TMR0B_IOA	Software Event
TMR1	Timer Input Pin	TMR1A_IOA	TMR1B_IOA	Software Event
TMR2	-	-	-	-
TMR3	-	-	-	-
LPTMR0	LPTMR0B_IOA	LPCMP0 Interrupt	LPCMP1 Interrupt	-
LPTMR1	LPTMR1B_IOA	LPCMP0 Interrupt	LPCMP1 Interrupt	-

17.2 Basic Timer Operation

The timer modes operate by incrementing the *TMRn_CNT.count* register, driven by either the timer clock, an external stimulus on the timer pin, or a combination of both. The *TMRn_CNT.count* register is always readable, even while the timer is enabled and counting.

Each timer mode has a user-configurable timer period, which terminates on the timer clock cycle following the end of the timer period condition. Each timer mode has a different response at the end of a timer period, which can include changing the state of the timer pin, capturing a timer value, reloading *TMRn_CNT.count* with a new starting value, or disabling the counter. The end of a timer period always sets the corresponding interrupt bit and can generate an interrupt, if enabled.

In most modes, the timer peripheral automatically sets *TMRn_CNT.count* to 0x0000 0001 at the end of a timer period, but *TMRn_CNT.count* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset is one timer clock longer than subsequent timer periods if *TMRn_CNT.count* is not initialized to 0x0000 0001 during the timer configuration step.

17.3 32-Bit Single / 32-Bit Cascade / Dual 16-Bit

Most instances contain two 16-bit timers, which may support combinations of single or cascaded 32-bit modes, and single or dual 16-bit modes as shown in *Table 17-1*. In most cases, the two 16-bit timers have the same functionality.

The terminology TimerA and TimerB are used to differentiate the organization of the 32-bit registers shown in *Table 17-3*. Most of the other registers have the same fields duplicated in the upper and lower 16-bits and are differentiated with the _a and _b suffixes.

In the 32-bit modes, the fields and controls associated with TimerA are used to control the 32-bit timer functionality. In single 16-bit timer mode, the TimerA fields are used to control the single 16-bit timer and the TimerB fields are ignored. In dual 16-bit timer modes, both TimerA and TimerB fields are used to control the dual timers; TimerB fields control the upper 16-bit timer and TimerA fields control the lower 16-bit timer. In dual-16 bit timer modes, TimerB can be used as a single 16-bit timer.

Table 17-3: TimerA/TimerB 32-Bit Field Allocations

Register	Cascade 32-Bit Mode	Dual 16-Bit Mode		Single 16-Bit Mode
Timer Counter	TimerA Count = *TMRn_CNT.count[31:0]*	TimerA Compare = *TMRn_CNT.count[15:0]*	TimerB Count = *TMRn_CNT.count[31:16]*	TimerA Compare = *TMRn_CNT.count[15:0]*
Timer Compare	TimerA Compare = *TMRn_CMP.compare[31:0]*	TimerA Compare = *TMRn_CMP.compare[15:0]*	TimerB Compare = *TMRn_CMP.compare[31:16]*	TimerA Compare = *TMRn_CMP.compare[15:0]*
Timer PWM	TimerA Count = *TMRn_PWM.pwm[31:0]*	TimerA Count = *TMRn_PWM.pwm[15:0]*	TimerB Count = *TMRn_PWM.pwm[31:16]*	TimerA Count = *TMRn_PWM.pwm[15:0]*

17.4 Timer Clock Sources

Clocking of timer functions is driven by the timer clock frequency, f_{CNT_CLK}, which is a function of the selected clock source shown in *Table 17-1*. Most modes support multiple clock sources and prescaler values, which can be chosen independently for TimerA and TimerB when the peripheral is operating in dual 16-bit mode. The prescaler can be set from 1 to 4096 using the *TMRn_CNT.pres* field.

Equation 17-1: Timer Peripheral Clock Equation

$$f_{CNT\_CLK} = \frac{f_{CLK\_SOURCE}}{prescaler}$$

The software configures and controls the timer's by reading and writing to the timer registers. External events on timer pins are asynchronous events to the timer's clock frequency. The external events are latched on the next rising edge of the timer's clock. Since it is not possible to externally synchronize to the timer's internal clock, input events may require up to 50% of the timer's internal clock cycle before the hardware recognizes the event.

The software must configure the timer's clock source by performing the following steps:

1. Disable the timer peripheral.
   a. Clear *TMRn_CTRL0.en* to 0 to disable the timer:
   b. Read the *TMRn_CTRL1.clken* field until it returns 0 confirming the timer peripheral is disabled.
2. Set *TMRn_CTRL1.clksel* to the new desired clock source.
3. Configure the timer for the desired operating mode. See *Operating Modes* for details on mode configuration.
4. Enable the timer clock source:
   a. Set the *TMRn_CTRL0.clken* field to 1 to enable the timer's clock source.
   b. Read the *TMRn_CTRL1.clkrdy* field until it returns 1 confirming the timer clock source is enabled.

5. Enable the timer:
   a. Set *TMRn_CTRL0.en* to 1 to enable the timer.
   b. Read the *TMRn_CNT.clken* field until it returns 1 to confirm the timer is enabled.

The timer peripheral should be disabled while changing any of the registers in the peripheral.

17.5 Timer Pin Functionality

Each timer instance may have an input signal and/or output signal depending on the operating mode. Not all instances of the peripheral are available in all packages. The number of input and output signals per peripheral instance may vary as well. Refer to the device data sheet for I/O signal configurations and alternate functions for each Timer instance.

The physical pin location of the timer input and/or output signals may vary between packages. The timer functionality, however, is always expressed on the same GPIO pin in the same alternate function mode.

The timer pin functionality is mapped as an alternate function that is shared with a GPIO. When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc., as the GPIO mode settings for that pin. The pin characteristics must be configured before enabling the timer. When configured as an output, the corresponding bit in the GPIO_OUT register should be configured to match the inactive state of the timer pin for that mode. Consult the GPIO section for details on how to configure the electrical characteristics for the pin.

The TimerA output controls for modes 0, 1, 3, and 5 output signals are shown in *Figure 17-1*. The TimerA input controls for modes 2, 4, 6, 7, 8, and 14 input signals are shown in *Figure 17-2*.

Figure 17-1: MAX32655 TimerA Output Functionality, Modes 0/1/3/5

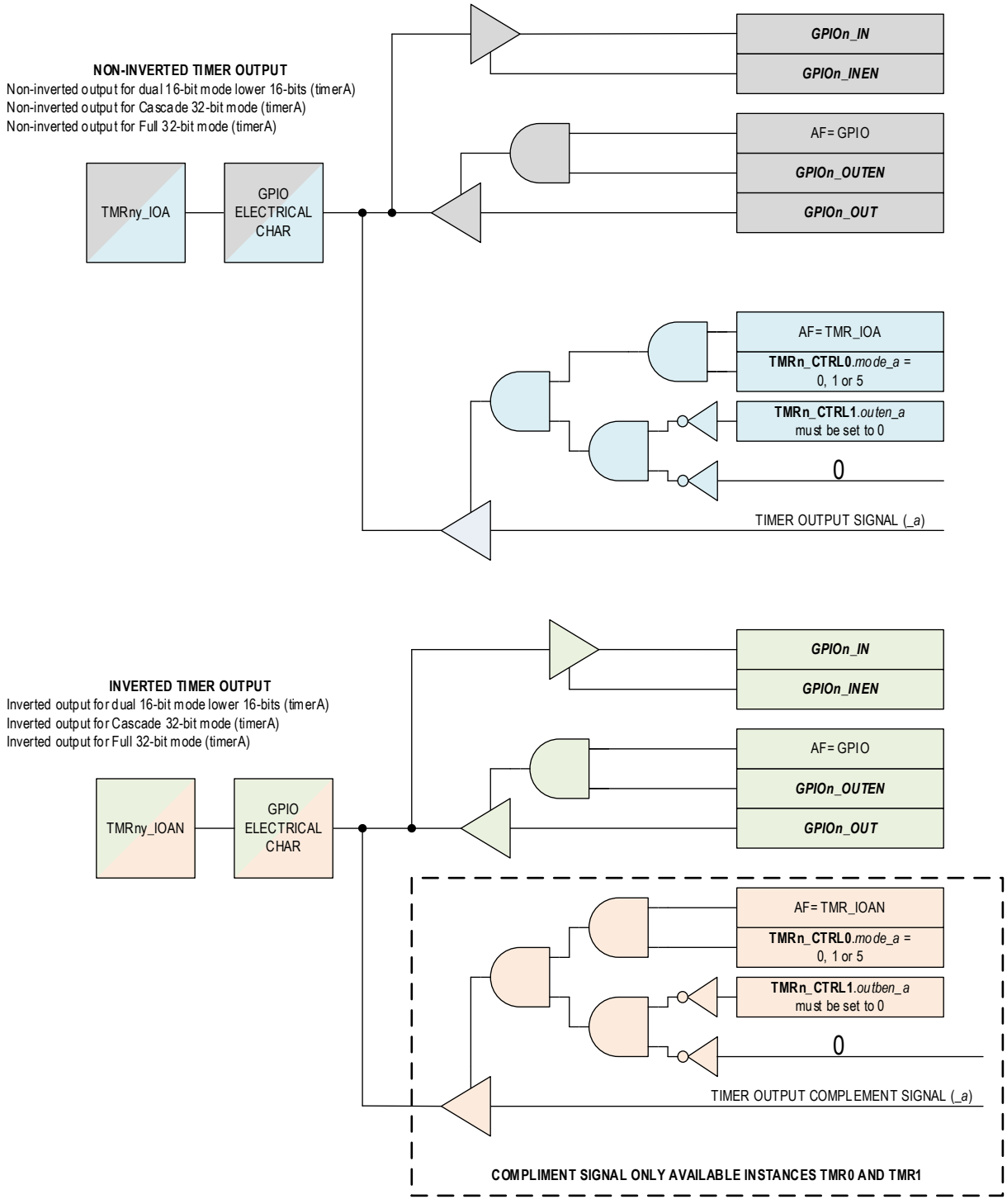
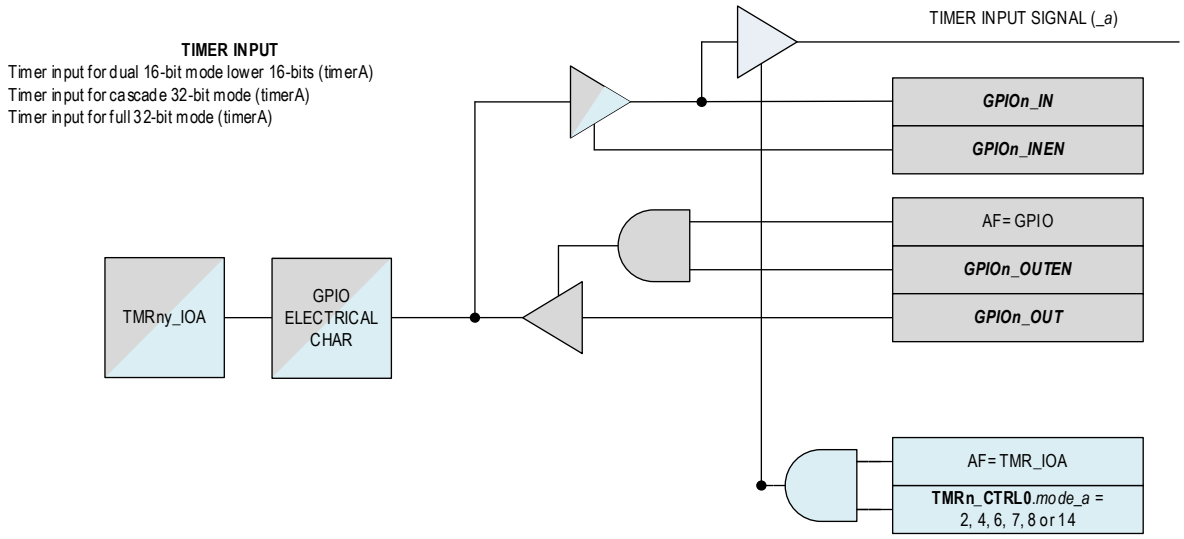


Figure 17-2: MAX32655 TimerA Input Functionality, Modes 2/4/6/7/8/14



17.6 Wakeup Events

In low-power modes, the system clock may be turned off to conserve power. LPTMR instances can continue to run from the clock sources shown in *Table 17-1*. In this case, a wakeup event can be configured to wake up the clock control logic and re-enable the system clock.

Programming sequence example:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Configure the timer operating mode as described in the section *Operating Modes*.
3. Enable the timer by setting *TMRn_CTRL0.en* to 1.
4. Poll *TMRn_CTRL1.clkrdy* until it reads 1.
5. Set the *TMRn_CTRL1.we* field to 1 to enable wakeup events for the timer.
6. If desired, enable the timer interrupt and provide a TMRn_IRQHandler for the timer.
6. Enter a low-power mode as described in the device *Operating Modes* section.
8. When the device wakes up from the low-power mode, check the *TMRn_WKFL* register to determine if the timer is the result of the wakeup event.

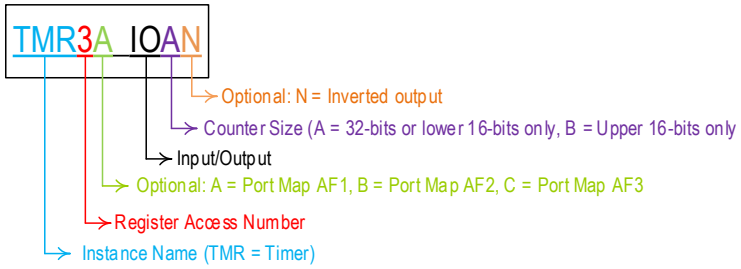
Table 17-4: MAX32655 Wakeup Events

Condition	Peripheral Wakeup Flag *TMRn_INTFL*	Peripheral Wakeup Enable	Low-Power Peripheral Wakeup Flag	Low-Power Peripheral Wakeup Enable	Power Management Wakeup Enable
Any event for LPTMR0	*irq_a*	N/A	*PWRSEQ_LPPWST .lptmr0*	*PWRSEQ_LPPWEN .lptmr0*	N/A
Any event for LPTMR1	*irq_a*	N/A	*PWRSEQ_LPPWST .lptmr1*	*PWRSEQ_LPPWEN .lptmr1*	N/A

17.7 Operating Modes

Multiple operating modes are supported. The availability of some operating modes is dependent on the device and package-specific implementation of the external input and output signals. Refer to the device data sheet for I/O signal configurations and alternate functions for each Timer instance.

Figure 17-3: Timer I/O Signal Naming Conventions



In *Table 17-5*, *Table 17-6*, and *Table 17-7*, the timer's signal name is generically shown where *n* is the timer number (0, 1, 2, 3, etc.) and *y* is the port mapping alternate function. See *Figure 17-3* for details of the timer's naming convention for I/O signals.

Table 17-5: MAX32655 Operating Mode Signals for Timer 0 and Timer 1

Timer Mode	TMR0/TMR1 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name[†]	Pin Required
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes

Timer Mode	TMR0/TMR1 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name†	Pin Required
Dual Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See* Figure 17-3 *for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 17-6: MAX32655 Operating Mode Signals for Timer 2 and Timer 3

Timer Mode	TMR2/TMR3 *TMRn_CTRL1*.*outen_a* = 0 *TMRn_CTRL1*.*outben_a* = 0	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (0 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See* Figure 17-3 *for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 17-7: MAX32655 Operating Mode Signals for Low-Power Timer 0 and Low-Power Timer 1

Timer mode	TMR4/TMR5 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IOB	Yes

Timer mode	TMR4/TMR5 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name†	Required?
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See* Figure 17-3 *for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

17.7.1 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the timer's *TMRn_CNT*.*count* field until it reaches the timer's *TMRn_CMP*.*compare* field and the timer is then disabled. If the timer's output is enabled, the output signal is driven active for one timer clock cycle. One-shot mode provides exactly one timer period and is automatically disabled.

The timer period ends on the timer clock following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*. The timer peripheral hardware automatically performs the following actions at the end of the timer period:

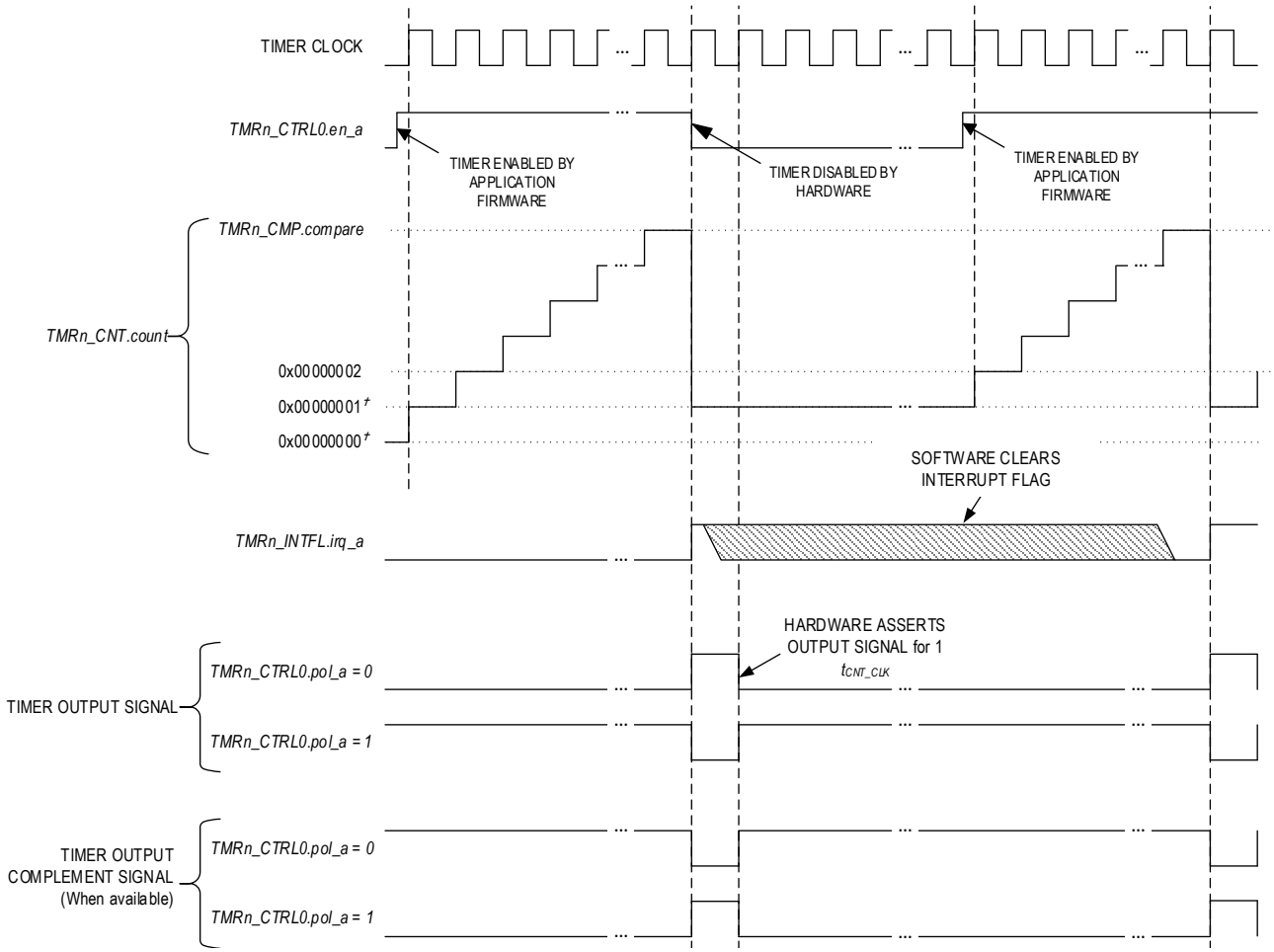
- The *TMRn_CNT*.*count* field is set to 0x0000 0001.
- The timer is disabled (*TMRn_CTRL0*.*en* = 0).
- The timer output, if enabled, is driven to its active state for one timer clock period.
- The *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The timer period is calculated using *Equation 17-2*.

Equation 17-2: One-shot Mode Timer Period

$$One-shot\ mode\ timer\ period\ in\ seconds = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}(Hz)}$$

Figure 17-4: One-Shot Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 0 (One-shot)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for one-shot mode by performing the following steps:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0*.*mode* field to 0 to select one-shot mode.
3. Set the *TMRn_CTRL0*.*pres* field to set the prescaler for the required timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.

6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP*.*compare* field.
8. If desired, write an initial value to *TMRn_CNT*.*count* field.
   a. The initial value only effects the first period; subsequent timer periods always reset the *TMRn_CNT*.*count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

17.7.2 Continuous Mode (1)

In continuous mode, the *TMRn_CNT*.*count* field increments until it matches the *TMRn_CMP*.*compare* field; the *TMRn_CNT*.*count* field is then set to 0x0000 0001 and the count continues to increment. Optionally, the software can configure continuous mode to toggle the timer output pin at the end of each timer period. A continuous mode timer period ends when the timer count field reaches the timer compare field (*TMRn_CNT*.*count* = *TMRn_CMP*.*compare*).

The timer peripheral hardware automatically performs the following actions on the timer clock cycle after the period ends:

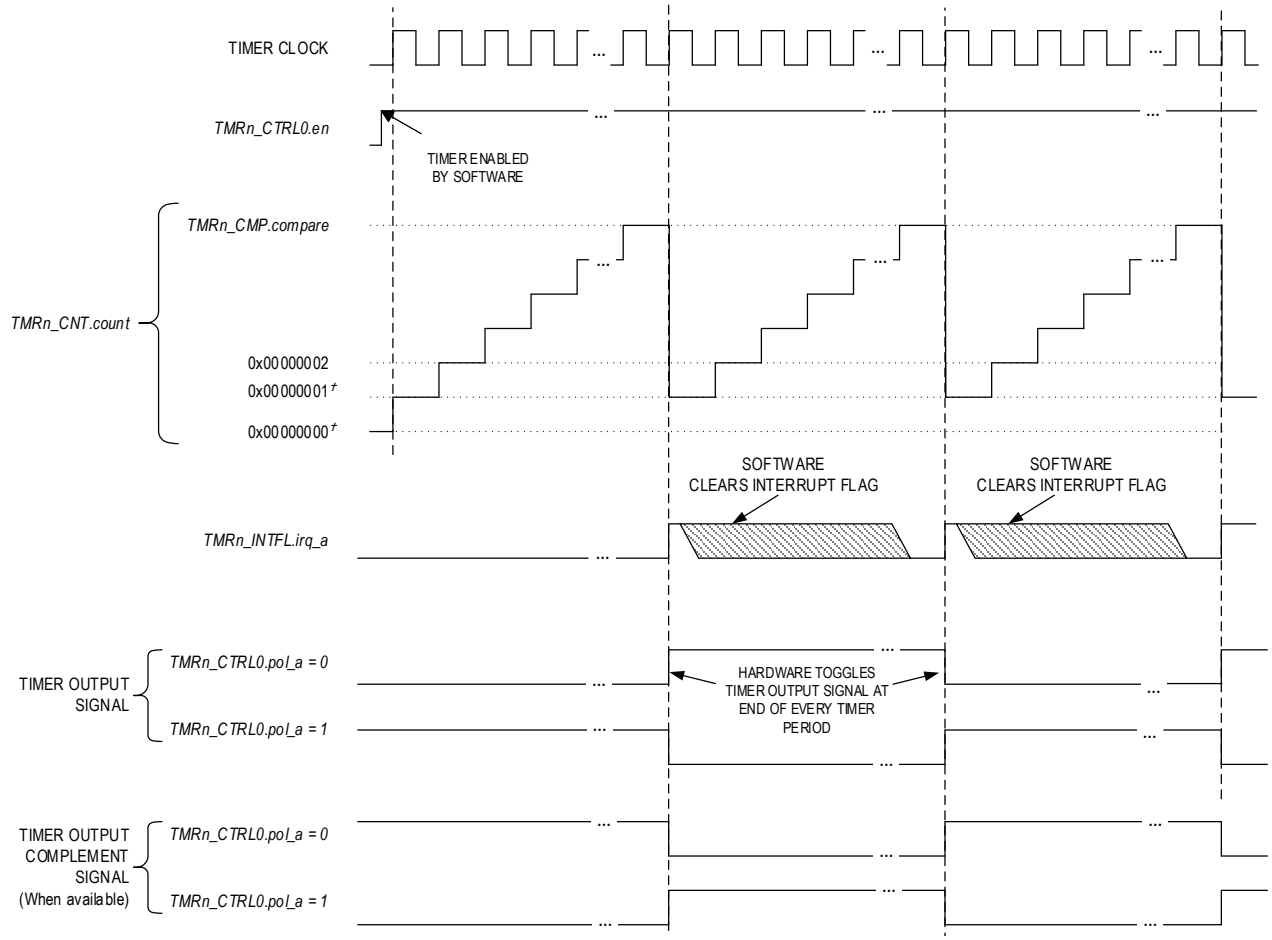
- The *TMRn_CNT*.*count* field is set to 0x0000 0001,
- if the timer output signal is toggled,
- the corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The continuous mode timer period is calculated using *Equation 17-3*.

Equation 17-3: Continuous Mode Timer Period

$$Continuous\ mode\ timer\ period\ (s) = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

Figure 17-5: Continuous Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 1 (Continuous)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for continuous mode by performing the following steps:

1. Disable the timer peripheral and set the timer clock as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0*.*mode* field to 1 to select continuous mode.
3. Set the *TMRn_CTRL0*.*pres* field to set the prescaler that determines the timer frequency.
4. If using the timer output function:
    a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
    b. Configure the GPIO electrical characteristics as desired.
    c. Select the correct alternate function mode for the timer output pin.
5. Of, if using the inverted timer output function:
    a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
    b. Configure the GPIO electrical characteristics as desired.
    c. Select the correct alternate function mode for the inverted timer output pin.

6. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP*.*compare* field.
8. If desired, write an initial value to the *TMRn_CNT*.count field.
   a. The initial value only effects the first period; subsequent timer periods always reset the *TMRn_CNT*.*count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

17.7.3 Counter Mode (2)

In counter mode, the timer peripheral increments the *TMRn_CNT*.*count* each time a transition occurs on the timer input signal. The transition must be greater than $4 \times PCLK$ for a count to occur. When the *TMRn_CNT*.*count* reaches the *TMRn_CMP*.*compare* field, the hardware automatically sets the interrupt bit to 1 (*TMRn_INTFL*.*irq)*, sets the *TMRn_CNT*.*count* field to 0x0000 0001, and continues incrementing. The timer can be configured to increment on either the rising edge or falling edge of the timer's input signal, but not both. Use the *TMRn_CTRL0*.*pol_* field to select which edge is used for the timer's input signal count.

The timer prescaler setting has no effect in this mode. The frequency of the timer's input signal (f_{CTR_CLK}) must not exceed 25 percent of the PCLK frequency as shown *Equation 17-4*.

Note: If the input signal's frequency is equal to f_{PCLK}, it is possible the transition can be missed by the timer hardware due to PCLK being an asynchronous internal clock. A minimum of 4 PCLK cycles is required for a count to occur. To guarantee a count occurs, the timer input signal should be greater than 4 PCKL cycles.

Equation 17-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

The timer period ends on the rising edge of PCLK following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral's hardware automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT*.*count* field is set to 0x0000 0001,
- the timer output signal is toggled if the timer output pin is enabled,
- the *TMRn_INTFL*.*irq* field to 1 indicating a timer interrupt event occurred,
- the timer remains enabled and continues incrementing.

Note: The software must clear the interrupt flag by writing 1 to the TMRn_INTFL.irq field. If the timer period ends and the interrupt flag is already set to 1, a second interrupt does not occur.

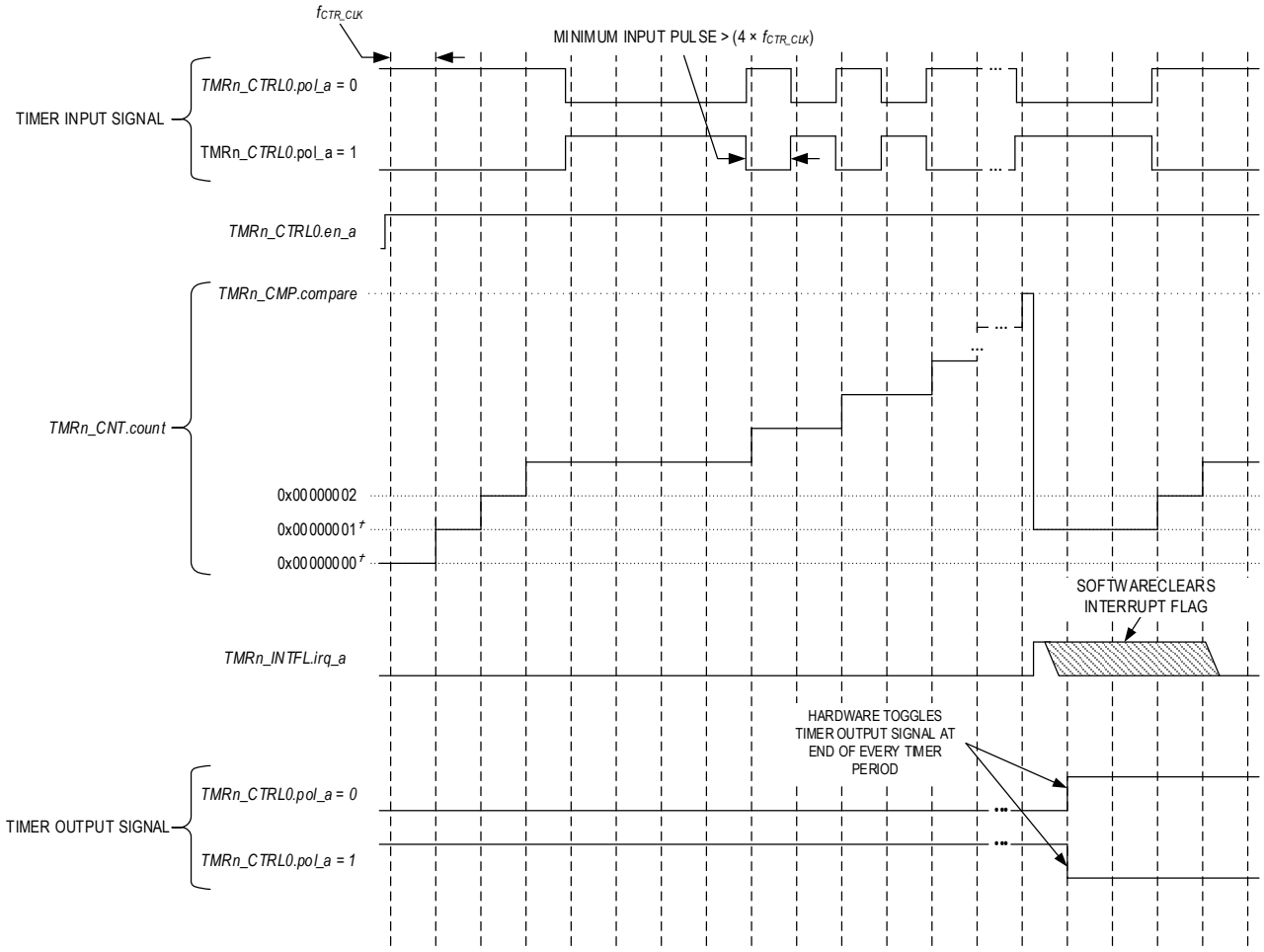
In counter mode, the number of timer input transitions that occurred during a period is equal to the *TMRn_CMP*.*compare* field's setting. Use *Equation 17-5* to determine the number of transitions that occurred prior to the end of the timer's period.

Note: Equation 17-5 *is only valid during an active timer count prior to the end of the timer's period.*

Equation 17-5: Counter Mode Timer Input Transitions

$$Counter\ mode\ timer\ input\ transitions = TMR\_CNT_{CURRENT\_VALUE}$$

Figure 17-6: Counter Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 2 (Counter)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for counter mode by performing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* 0x2 to select Counter mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Set *TMRn_CTRL1*.*outen_a* and *TMRn_CTRL1*.*outben* to the values shown in the *Operating Modes* section.
   d. Select the correct alternate function mode for the timer input pin.

5. Write the compare value to *TMRn_CMP*.*compare*.
6. If desired, write an initial value to *TMRn_CNT*.*count*. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

17.7.4 PWM Mode (3)

In PWM mode, the timer sends a PWM output using the timer's output signal. The timer first counts up to the match value stored in the *TMRn_PWM*.*pwm* register. At the end of the cycle where the *TMRn_CNT*.*count* value matches the *TMRn_PWM*.*pwm*, the timer output signal toggles state. The timer continues counting until it reaches the *TMRn_CMP*.*compare* value.

The timer period ends on the rising edge of f_{CNT_CLK} following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT*.*count* is reset to 0x0000 0001 and the timer resumes counting,
- the timer output signal is toggled,
- the corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

When *TMRn_CTRL0*.*pol* = 0, the timer output signal starts low and then transitions to high when the *TMRn_CNT*.*count* value matches the *TMRn_PWM* value. The timer output signal remains high until the *TMRn_CNT*.*count* value reaches the *TMRn_CMP*.*compare*, resulting in the timer output signal transitioning low, and the *TMRn_CNT*.*count* value resetting to 0x0000 0001.

When *TMRn_CTRL0*.*pol* = 1, the Timer output signal starts high and transitions low when the *TMRn_CNT*.*count* value matches the *TMRn_PWM* value. The timer output signal remains low until the *TMRn_CNT*.*count* value reaches *TMRn_CMP*.*compare*, resulting in the timer output signal transitioning high, and the *TMRn_CNT*.*count* value resetting to 0x0000 0001.

Complete the following steps to configure a timer for PWM mode and initiate the PWM operation:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set the *TMRn_CTRL0*.*mode field* to 3 to select PWM mode.
4. Set the *TMRn_CTRL0*.*pres* field to set the prescaler that determines the timer frequency.
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CTRL0*.*pol* to match the desired initial (inactive) state.
7. Set *TMRn_CTRL0*.*pol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
8. Set *TMRn_CNT*.*count* initial value if desired.
    a. The initial *TMRn_CNT*.*count* value only effects the initial period in PWM mode with subsequent periods always setting *TMRn_CNT*.*count* to 0x0000 0001.
9. Set the *TMRn_PWM* value to the transition period count.
10. Set the *TMRn_CMP*.*compare* value for the PWM second transition period. Note: *TMRn_CMP*.*compare* must be greater than the *TMRn_PWM* value.
11. If using the timer interrupt, set the interrupt priority and enable the interrupt.
12. Enable the timer peripheral as described in *Timer Clock Sources*.

Equation 17-6 shows the formula for calculating the timer PWM period.

Equation 17-6: Timer PWM Period

$$PWM\ period\ (s) = \frac{TMRn\_CNT}{f_{CNT\_CLK}\ (Hz)}$$

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT.count* register, use the one-shot mode equation, Equation 17-2, to determine the initial PWM period.

If *TMRn_CTRL0.pol* is 0, the ratio of the PWM output high time to the total period is calculated using Equation 17-7.

Equation 17-7: Timer PWM Output High Time Ratio with Polarity 0

$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

If *TMRn_CTRL0.pol* is set to 1, the ratio of the PWM output high time to the total period is calculated using Equation 17-8.

Equation 17-8: Timer PWM Output High Time Ratio with Polarity 1

$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{TMR\_PWM}{TMR\_CMP} \times 100$$

17.7.5 Capture Mode (4)

Capture mode is used to measure the time between software-determined events. The timer starts incrementing the timer's count field until a transition occurs on the timer's input pin or a rollover event occurs. A capture event is triggered by the hardware when the timer's input pin transitions state. Equation 17-9 shows the formula for calculating the capture event's elapsed time.

If a capture event does not occur prior to the timer's count value reaching the timer's compare value (*TMRn_CNT.count* = *TMRn_CMP.compare*), a rollover event occurs. Both the capture event and the rollover event set the timer's interrupt flag, *TMRn_INTFL.irq* to 1 and result in an interrupt if the timer's interrupt is enabled.

A capture event can occur before or after a rollover event. The software must track the number of rollover events that occur prior to a capture event to determine the elapsed time of the capture event. When a capture event occurs, the software should reset the count of rollover events.

Note: A capture event does not stop the timer's counter from incrementing and does not reset the timer's count value; a rollover event still occurs when the timer's count value reaches the timer's compare value.

17.7.5.1 Capture Event

When a capture event occurs, the timer hardware, on the next timer clock cycle, automatically performs the following actions:

- The *TMRn_CNT.count* value is copied to the *TMRn_PWM.pwm* field,
- the *TMRn_INTFL.irq* field is set to 1,
- the timer remains enabled and continues counting.

The software must check the value of the TMRn_PWM.pwm field to determine the trigger of the timer interrupt.

Equation 17-9: Capture Mode Elapsed Time Calculation in Seconds

$$Capture\ elapsed\ time\ (s) = \frac{\left(TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}\right) + \left((Number\ of\ rollover\ events) \times \left(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE}\right)\right)}{f_{CNT\_CLK}}$$

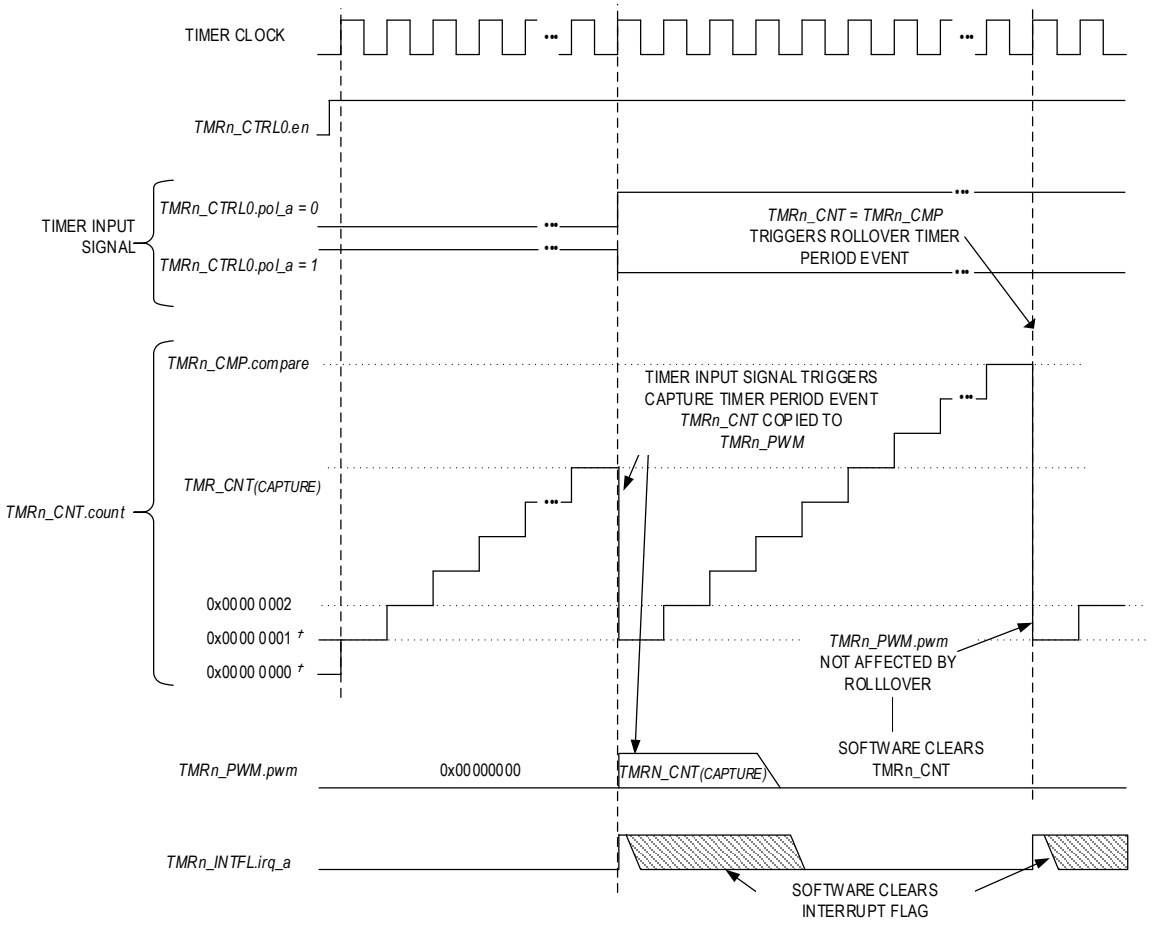
Note: The capture elapsed time calculation is only valid after the capture event occurs and the timer stores the captured count in the TMRn_PWM register.

17.7.5.2 Rollover Event

A rollover event occurs when the timer's count value reaches the timer's compare value (*TMRn_CNT.count* = *TMRn_CMP.compare*). A rollover event indicates that a capture event did not occur within the set timer period. When a rollover event occurs, the timer hardware automatically performs the following actions during the next timer clock period:

- The *TMRn_CNT.count* field is set to 0x0000 0001,
- the *TMRn_INTFL.irq* field is set to 1,
- the timer remains enabled and continues counting.

Figure 17-7: Capture Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 2 (COUNTER)

† *TMRn*_CNT.*count* DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT.count* RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 4 to select capture mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. Write the initial value to *TMRn_CNT*.*count*. if desired.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to the *TMRn_CMP*.*compare* field.
7. Select the capture event by setting *TMRn_CTRL1*.*capeventsel*.
8. Enable the timer peripheral as described in *Timer Clock Sources*.

The timer period is calculated using the following equation:

Equation 17-10: Capture Mode Elapsed Time Calculation in Seconds

$$Capture\ elapsed\ time\ in\ seconds = \frac{TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}}{f_{CNT\_CLK}}$$

Nsote: The capture elapsed time calculation is only valid after the capture event occurs, and the timer stores the captured count in the TMRn_PWM *register.*

17.7.6 Compare Mode (5)

In compare mode, the timer peripheral increments continually from 0x0000 0000 (after the first timer period) to the maximum value of the 32- or 16-bit mode, then rolls over to 0x0000 0000 and continues incrementing. The end of timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000 0000.

The timer period ends on the timer clock following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions when a timer period event:

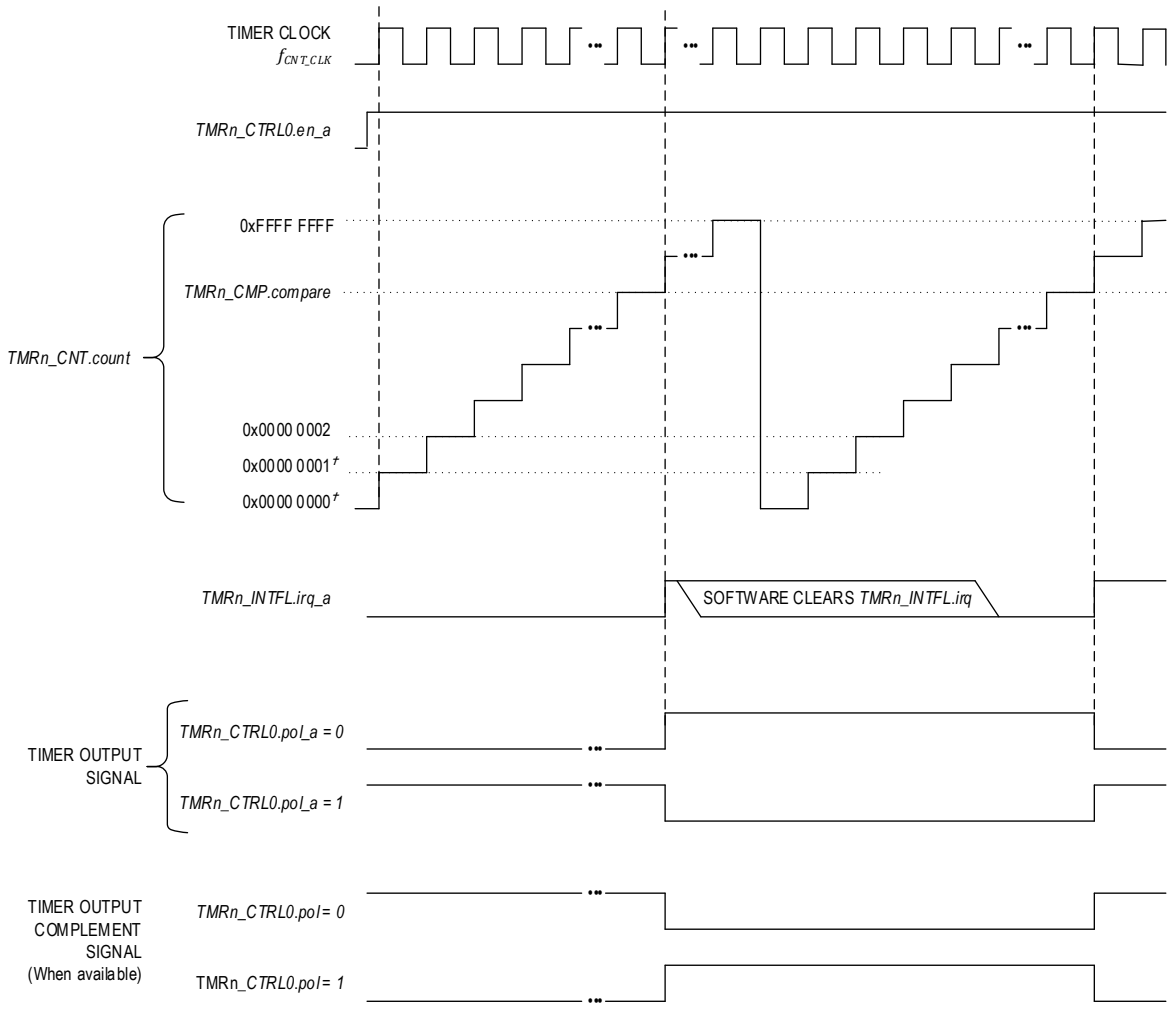
- Unlike other modes, *TMRn_CNT*.*count* is reset to 0x0000 00000 not 0x0000 0001 at the end of the timer period. The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.
- The hardware toggles the state of the timer output signal. The timer output pin changes state if the timer output is enabled.

The compare mode timer period is calculated using *Equation 17-12*.

Equation 17-11: Compare Mode Timer Period

$$Compare\ mode\ timer\ period\ in\ second = \frac{(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE} + 1)}{f_{CNT\_CLK}(Hz)}$$

Figure 17-8: Compare Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 5 (Compare)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 5 to select Compare mode.
4. Set *TMRn_CTRL0.pres* to set the prescaler that determines the timer frequency.
5. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.

6. If using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
7. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
8. Write the compare value to *TMRn_CMP*.*compare.*
9. If desired, write an initial value to *TMRn_CNT*.*count*.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
10. Enable the timer peripheral as described in *Timer Clock Sources*.

17.7.7 Gated Mode (6)

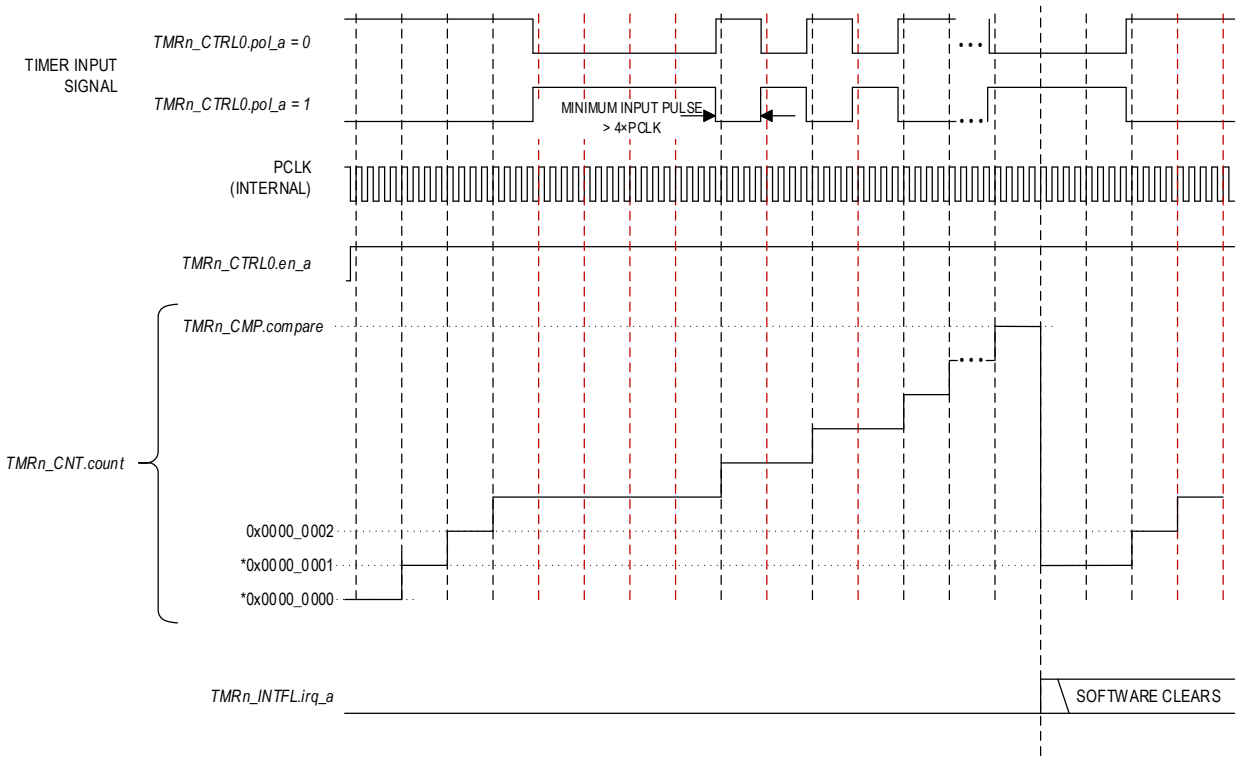
Gated mode is similar to continuous mode, except that *TMRn_CNT*.*count* only increments when the timer input signal is in its active state.

The timer period ends on the timer clock following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT*.*count* field is set to 0x0000 0001.
- the timer remains enabled and continues incrementing.
- If the timer output signal toggles state. The timer output pin changes state if the timer output is enabled.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

Figure 17-9: Gated Mode Diagram



This example uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 6 (Gated)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for gated mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 6 to select gated mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT*.*count* field.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP*.*compare*.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

17.7.8 Capture/Compare Mode (7)

In Capture/Compare mode, the timer starts counting after the first external timer input transition occurs. The transition, a rising edge or falling edge on the timer's input signal, is set using the *TMRn_CTRL0.pol* bit.

Each subsequent transition, after the first transition of the timer input signal, captures the *TMRn_CNT.count* value, writing it to the *TMRn_PWM.pwm* register (capture event). When a capture event occurs, a timer interrupt is generated, the *TMRn_CNT.count* value is reset to 0x0000_0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to *TMRn_CMP.compare*. At the end of the cycle where the *TMRn_CNT.count* equals the *TMRn_CMP.compare*, a timer interrupt is generated, the *TMRn_CNT.count* value is reset to 0x0000 0001, and the timer resumes counting.

The timer period ends when the selected transition occurs on the timer pin, or on the clock cycle following *TMRn_CNT.count* = *TMRn_CMP.compare*.

The actions performed at the end of the timer period are dependent on the event that ended the timer period:

If the end of the timer period was caused by a transition on the timer pin, the hardware automatically performs the following:

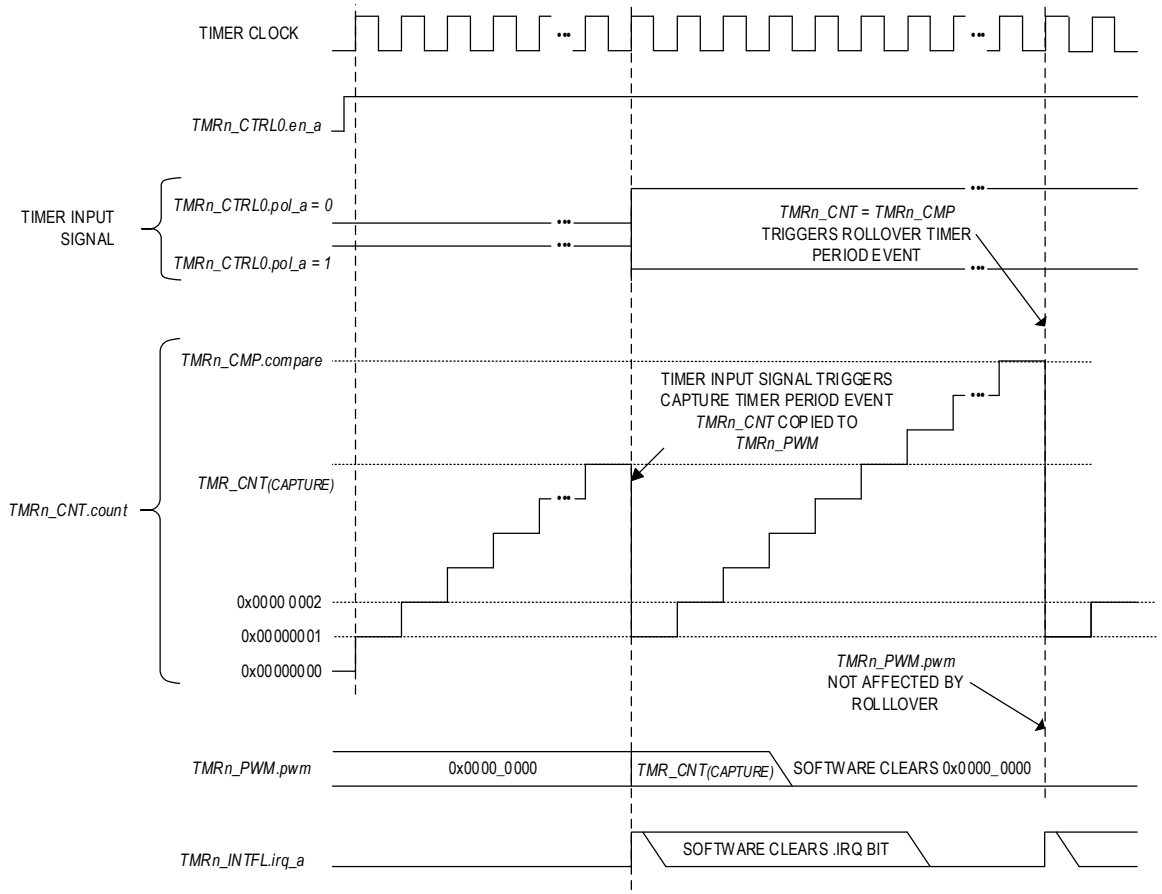
- The value in *TMRn_CNT.count* field is copied to the *TMRn_PWM.pwm* field,
- the *TMRn_CNT.count* field is set to 0x0000 0001,
- the timer remains enabled and continues incrementing,
- the corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 17-12*.

Equation 17-12: Capture Mode Elapsed Time

$$Capture\ elapsed\ time\ (seconds) = \frac{TMRn\_PWM - TMRn\_CNT_{INITIAL\_CNT\_VALUE}}{f_{CNT\_CLK}(Hz)}$$

Figure 17-10: Capture/Compare Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 7 (CAPTURE/COMPARE)

† *TMRn_CNT*.count DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT*.count RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture/Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 7 to select Capture/Compare mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to select the positive edge (*TMRn_CTRL0*.*pol* = 1) or negative edge (*TMRn_CTRL0*.*pol* = 0) transition to cause the capture event..
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT*.*count* field.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.

6 Write the compare value to *TMRn_CMP.compare*.

7. Enable the timer peripheral as described in *Timer Clock Sources*.

Note: No interrupt is generated by the first transition of the input signal.

17.7.9 Dual Edge Capture Mode (8)

Dual edge capture mode is similar to capture mode except the counter can capture on both edges of the timer input pin.

17.7.10 Inactive Gated Mode (14)

Inactive gated mode is similar to gated mode except that the interrupt is triggered when the timer input pin is in its inactive state.

17.8 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 17-8*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 17-8: Timer Register Summary

Offset	Register	Description
[0x0000]	*TMRn_CNT*	Timer Counter Register
[0x0004]	*TMRn_CMP*	Timer Compare Register
[0x0008]	*TMRn_PWM*	Timer PWM Register
[0x000C]	*TMRn_INTFL*	Timer Interrupt Register
[0x0010]	*TMRn_CTRL0*	Timer Control Register
[0x0014]	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register
[0x0018]	*TMRn_CTRL1*	Timer Configuration Register
[0x001C]	*TMRn_WKFL*	Timer Wakeup Status Register

17.8.1 Register Details

Table 17-9: Timer Count Register

Timer Count			TMRn_CNT	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count** This field increments at a rate dependent on the selected timer operating mode. The function of the bits in this field are dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value.

Table 17-10: Timer Compare Register

Timer Compare			TMRn_CMP	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The compare field meaning is determined by the specific mode of the timer. See the timer mode's detailed configuration section for compare usage and meaning.

Table 17-11: Timer PWM Register

Timer PWM			TMRn_PWM	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	pwm	R/W	0	**Timer PWM Match** In PWM mode, this field sets the count value for the first transition period of the PWM cycle. At the end of the cycle when *TMRn_CNT.count* = *TMRn_CMP.compare*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in *TMRn_CMP.compare*. *TMRn_PWM.pwm* must be less than *TMRn_CMP.compare* for PWM mode operation. **Timer Capture Value** In capture, compare, and capture/compare modes, this field is used to store the *TMRn_CNT.count* value when a Capture, Compare, or Capture/Compare event occurs.

Table 17-12: Timer Interrupt Register

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
24	wr_dis_b	R/W	0	**TimerB Write Protect in Dual Timer Mode** Set this field to 0 to write protect the TimerB fields in the *TMRn_CNT.count[31:16]* and *TMRn_PWM.pwm[31:16]*. When this field is set to 0, 32-bit writes to the *TMRn_CNT* and *TMRn_PWM* registers only modify the lower 16-bits associated with TimerA. 0: Enabled 1: Disabled *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
25	wrdone_b	R	0	**TimerB Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.count[31:16]* or *TMRn_PWM.pwm[31:16]* when in dual timer mode. Wait until the field is set to 1 before proceeding. 0: Operation in progress. 1: Operation complete.
23:17	-	RO	0	**Reserved**
16	irq_b	R/W1C	0	**TimerB Interrupt Event** This field is set when a TimerB interrupt event occurs. Write 1 to clear. 0: No event 1: Interrupt event occurred
15:10	-	RO	0	**Reserved**

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
9	wr_dis_a	R/W	0	**TimerB Dual Timer Mode Write Protect** This field disables write access to the *TMRn_CNT.count[31:16]* and *TMRn_PWM.pwm[31:16]* fields so that only the 16 bits associated with updating TimerA are modified during writes to the *TMRn_CNT* and *TMRn_PWM* registers. 0: Enabled 1: Disabled *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
8	wrdone_a	R	0	**TimerA Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.count[31:16]* or *TMRn_PWM.pwm[31:16]* when in dual 16-bit timer mode. Wait until the field reads 1 before proceeding. 0: Operation in progress 1: Operation complete
7:1	-	RO	0	**Reserved**
0	irq_a	W1C	0	**TimerA Interrupt Event** This field is set when a TimerA interrupt event occurs. Write 1 to clear. 0: No event 1: Interrupt event occurred

Table 17-13: Timer Control 0 Register

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31	en_b	R/W	0	**TimerB Enable** 0: Disabled 1: Enabled
30	clken_b	R/W	0	**TimerB Clock Enable** 0: Disabled 1: Enabled
29	rst_b	W1	0	**TimerB Reset** 0: No action 1: Reset TImerB
28:24	-	RO	0	**Reserved**

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
23:20	clkdiv_b	R/W	0	**TimerB Prescaler Select** The *clkdiv_b* field selects a prescaler that divides the timer's source clock to set the timer's count clock as follows: $f_{CNT\_CLK} = f_{CLK\_SOURCE} / prescaler$ See the *Operating Modes* section for details on which timer modes use the prescaler. 0: 1 1: 2 2: 4 3: 8 4: 16 5: 32 6: 64 7: 128 8: 256 9: 512 10: 1024 11: 2048 12: 4096 13-15: Reserved
19:16	mode_b	R/W	0	**TimerB Mode Select** Set this field to the desired mode for TimerB. 0: One-Shot 1: Continuous 2: Counter 3: PWM 4: Capture 5: Compare 6: Gated 7: Capture/Compare 8: Dual-Edge Capture 9-11: Reserved 12: Internally Gated 13-15: Reserved
15	en_a	R/W	0	**TimerA Enable** 0: Disabled 1: Enabled
14	clken_a	R/W	0	**TimerA Clock Enable** 0: Disabled 1: Enabled
13	rst_a	R/W1O	0	**TimerA Reset** 0: No action 1: Reset TimerA

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
12	pwmckbd_a	R/W	1	**TimerA PWM Output $\phi A'$ Disable** Set this field to 0 to enable the $\phi A'$ output signal. The $\phi A'$ output signal is disabled by default. 0: Enable the PWM $\phi A'$ output signal 1: Disable PWM $\phi A'$ output signal
11	nollpol_a	R/W	0	**TimerA PWM Output $\phi A'$ Polarity Bit** Set this field to 1 to invert the PWM $\phi A'$ signal. 0: Do not invert the PWM $\phi A'$ output signal 1: Invert the PWM $\phi A'$ output signal
10	nolhpol_a	R/W	0	**TimerA PWM Output ϕA Polarity Bit** Set this field to 1 to invert the PWM ϕA signal. 0: Do not invert the ϕA PWM output signal 1: Invert the ϕA output signal
9	pwmsync_a	R/W	0	**TimerA/TimerB PWM Synchronization Mode** 0: Disabled 1: Enabled
8	pol_a	R/W	0	**TimerA Polarity** Selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the timer's alternate function. This field's usage and settings are operating mode specific. See the *Operating Modes* section for details on the mode selected.
7:4	clkdiv_a	R/W	0	**TimerA Prescaler Select** The *clkdiv_a* field selects a prescaler that divides the timer's clock source to set the timer's count clock as follows: $f_{CNT\_CLK} = {f_{CLK\_SOURCE}}/{prescaler}$ See the *Operating Modes* section to determine which modes use the prescaler. 0: 1 1: 2 2: 4 3: 8 4: 16 5: 32 6: 64 7: 128 8: 256 9: 512 10: 1024 11: 2048 12: 4096 13-15: Reserved

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	mode_a	R/W	0	**TimerA Mode Select** Set this field to the desired operating mode for TimerA. 0: One-Shot 1: Continuous 2: Counter 3: PWM 4: Capture 5: Compare 6: Gated 7: Capture/Compare 8: Dual-Edge Capture 9-11: Reserved for Future Use 12: Internally Gated 13-15: Reserved for Future Use

Table 17-14: Timer Non-Overlapping Compare Register

Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	hi_b	R/W	0	**TimerA Non-Overlapping High Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ (phase A prime) and the next rising edge of the PWM output ϕA *(phase A)*.
23:16	lo_b	R/W	0	**TimerA Non-Overlapping Low Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.
15:8	hi_a	R/W	0	**TimerA Non-Overlapping High Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ and the next rising edge of the PWM output ϕA.
7:0	lo_a	R/W	0	**TimerA Non-Overlapping Low Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.

Table 17-15: Timer Control 1 Register

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cascade	R/W	0	**32-bit Cascade Timer Enable** This field is only supported by Timer instances with support for 32-bit cascade mode. 0: Dual 16-bit timers 1: 32-bit cascade timer
30	outben_b	R/W	0	**TimerB Output B Enable** Reserved for future use
29	outen_b	R/W	0	**TimerB Output Enable** Reserved for future use

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
28	we_b	R/W	0	**TimerB Wakeup Function** 0: Disabled 1: Enabled
27	sw_capevent_b	R/W	0	**TimerB Software Event Capture** Write this field to 1 to initiate a software event capture when operating the timer in capture mode to perform a software event capture. 0: No event 1: Reserved
26:25	capevent_sel_b	R/W	0	**TimerB Event Capture Selection** Set this field to the desired capture event source. See *Table 17-2* for available capture event 0 and capture event 1 options. 0-3: Reserved
24	ie_b	R/W	0	**TimerB Interrupt Enable** 0: Disabled 1: Enabled
23	negtrig_b	R/W	0	**TimerB Negative Edge Trigger for Event** 0: Rising-edge trigger 1: Falling-edge trigger
22:20	event_sel_b	R/W	0	**TimerB Event Selection** 0: Event disabled 1-7: Reserved
19	clkrdy_b	RO	0	**TimerB Clock Ready Status** This field indicates if the timer clock is ready. 0: Timer clock not ready or synchronization in progress 1: Timer clock is ready
18	clken_b	RO	0	**TimerB Clock Enable Status** This field indicates the status of the timer enable. 0: Timer not enabled or synchronization in progress 1: Timer is enabled
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 17-1* for the clock sources supported by each instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.
15	-	RO	0	**Reserved**
14	outben_a	R/W	0	**Output B Enable** Reserved for future use
13	outen_a	R/W	0	**Output Enable** Reserved for future use
12	we_a	R/W	0	**TimerA Wakeup Function** 0: Disabled 1: Enabled.
11	sw_capevent_a	R/W	0	**TimerA Software Event capture** 0: No software capture event triggered. 1: Trigger software capture event.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
10:9	capeventsel_a	R/W	0	**TimerA Event capture Selection** Set this field to the desired capture event source. See *Table 17-2* for available capture event 0 and capture event 1 options. 0: Capture event 0. 1: Capture event 1. 2: Capture event 2. 3: Capture event 3.
8	ie_a	R/W	0	**TimerA Interrupt Enable** 0: Disabled 1: Enabled
7	negtrig_a	R/W	0	**TimerA Edge Trigger Selection for Event** 0: Positive-edge triggered. 1: Negative-edge triggered.
6:4	event_sel_a	R/W	0	**TimerA Event Selection** 0: Event disabled 1-7: Reserved s
3	clkrdy_a	RO	0	**TimerA Clock Ready** This field is set to 1 after the software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_a* field. 0: Timer not enabled or synchronization in progress. 1: TimerA clock is ready.
2	clken_a	R/W	0	**TimerA Clock Enable** Write this field to 1 to enable the TimerA clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 17-1* for the available clock options for each timer instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.

Table 17-16: Timer Wakeup Status Register

Timer Wakeup Status			TMRn_WKFL	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	b	R/W1C	1	**TimerB Wakeup Event** This flag is set when a wakeup event occurs for TimerB. Write 1 to clear. 0: No event 1: Wakeup event occurred
15:1	-	RO	0	**Reserved**
0	a	R/W1C	1	**TimerA Wakeup Event** This flag is set when a wakeup event occurs for TimerA. Write 1 to clear. 0: No event 1: Wakeup event occurred

18. Wakeup Timer (WUT)

The Wakeup Timer (WUT) is a unique instance of a 32-bit timer.

- The wakeup timer uses the ERTCO for it's clock source.
- Programmable prescaler with values from 1 to 4096.
- Supports two timer modes:
  - One-Shot: The timer counts up to the terminal value then halts.
  - Continuous: The timer counts up to the terminal value then repeats.
- Independent interrupt handler (WUTn_IRQHandler).

18.1 Basic Operation

The timer modes operate by incrementing the *WUTn_CNT.count* register. The *WUTn_CNT.count* register is always readable, even while the timer is enabled and counting.

Each timer mode has a user-configurable timer period, which terminates on the timer clock cycle following the end of timer period condition. The end of a timer period always sets the corresponding interrupt flag and generates a wakeup timer interrupt (WUTn_IRQ), if enabled.

The timer peripheral automatically sets *WUTn_CNT.count* to 1 at the end of a timer period, but *WUTn_CNT.count* is set to 0 following a system reset. This means the first timer period following a system reset is one timer clock longer than subsequent timer periods if *WUTn_CNT.count* is not initialized to 1 during the timer configuration step.

The timer clock frequency, f_{CNT_CLK}. is a divided version of the 32.768kHz RTC clock as shown in *Equation 18-1*.

Equation 18-1: Wakeup Timer Clock Frequency

$$f_{CNT\_CLK} = \frac{f_{RTC\_CLK}}{prescaler}$$

The divisor (prescaler) can be set from 1 to 4096 using the concatenated fields *WUTn_CTRL.pres3*:*WUTn_CTRL.pres* as shown in *Table 18-1*.

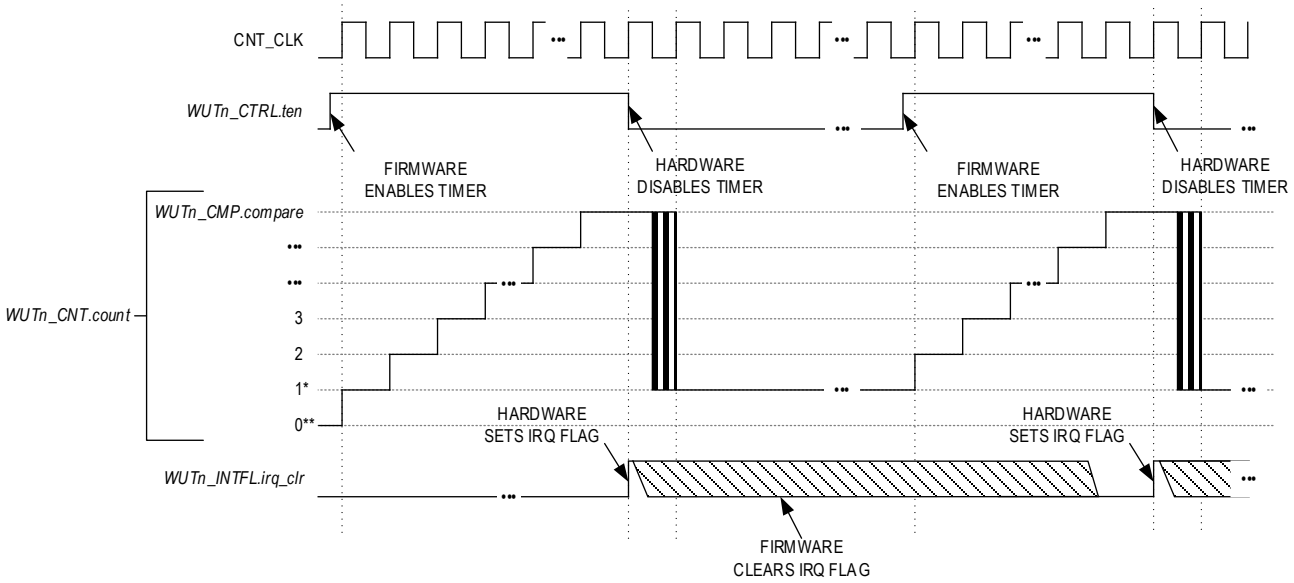
Table 18-1: MAX32655 WUT Clock Period

WUTn_CTRL.pres3	*WUTn_CTRL.pres*	Prescaler	f_{CNT_CLK} (Hz)
0	0b000	1	32,768
0	0b001	2	16,384
0	0b010	4	8,192
0	0b011	8	4,096
0	0b100	16	2,048
0	0b101	32	1,024
0	0b110	64	512
0	0b111	128	256
1	0b000	256	128
1	0b010	512	64
1	0b011	1024	32
1	0b100	2048	16
1	0b101	4096	8
1	0b110	Reserved	Reserved
1	0b111	Reserved	Reserved

18.2 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the *WUTn_CNT.count* field until it matches the *WUTn_CMP.compare* field and then stops incrementing and disables the timer. In this mode, the timer must be re-enabled to start another one-shot mode event.

Figure 18-1: One-Shot Mode Diagram



* *WUTn_CNT.count* automatically reloads with 1 at the end of the WUT PERIOD, but software can write any initial value to *WUTn_CNT.count* prior to enabling the timer.

** The default value of *WUTn_CNT.count* for the first period after a system reset is 0 unless changed by software.

18.2.1 One-Shot Mode Timer Period

The timer period ends on the timer clock when *WUTn_CNT.count* = *WUTn_CMP.compare*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *WUTn_CNT.count* is reset to 1.
2. The timer is disabled by setting *WUTn_CTRL.ten* = 0.
3. The timer interrupt bit *WUTn_INTFL.irq_clr* is set. An interrupt is generated if enabled.

18.2.2 One-Shot Mode Configuration

Configure the timer for one-shot mode by performing the following steps:

1. Set *WUTn_CTRL*.*ten* = 0 to disable the timer.
2. Set *WUTn_CTRL*.*tmode* to 0 to select one-shot mode.
3. Set *WUTn_CTRL*.*pres3*:*WUTn_CTRL*.*pres* to determine the timer period.
4. Enable the interrupt, if desired, and set the interrupt priority.
5. Write an initial value to *WUTn_CNT*.*count*, if desired. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.*count* to 1.
6. Write the compare value to *WUTn_CMP*.*compare*.
7. Set *WUTn_CTRL*.*ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

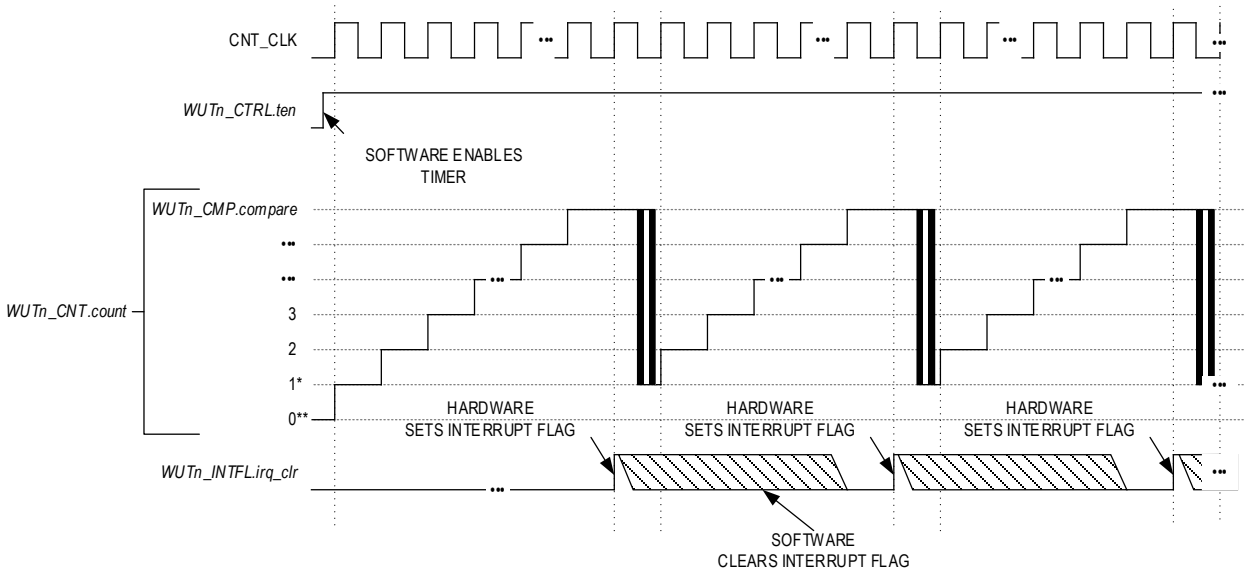
Equation 18-2: One-Shot Mode Timer Period

$$One\text{-}Shot\ mode\ timer\ period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

18.3 Continuous Mode (1)

In continuous mode, the timer peripheral increments *WUTn_CNT*.*count* until it matches *WUTn_CMP*.*compare* and the hardware resets *WUTn_CNT*.*count* to 1, and continues incrementing.

Figure 18-2: Continuous Mode Diagram



* *WUTn_CNT.count* automatically reloads with 1 at the en of the wakeup timer period, but softwrae can write any initial value to *WUTn_CNT.count* prior to enabling the wakeup timer.

** The value of *WUTn_CNT.count* for the first period after a system reset is 0 unless changed by software.

18.3.1 Continuous Mode Timer Period

The wakeup timer period ends on the timer clock following *WUTn_CNT*.*count* = *WUTn_CMP*.*compare*.

The wakeup timer peripheral automatically performs the following actions at the end of the timer period:

1. *WUTn_CNT*.*count* is reset to 1. The wakeup timer remains enabled and continues incrementing.
2. The timer interrupt bit *WUTn_INTFL*.*irq_clr* is set. An interrupt is generated if enabled.

18.3.2 Continuous Mode Configuration

Configure the timer for continuous mode by performing the steps following:

1. Set *WUTn_CTRL*.*ten* = 0 to disable the timer.
2. Set *WUTn_CTRL*.*tmode* to 1 to select continuous mode.
3. Set *WUTn_CTRL*.*pres3*:*WUTn_CTRL*.*pres* to determine the timer period.
4. Enable the interrupt, if desired, and set the interrupt priority.
5. Write an initial value to *WUTn_CNT*.*count*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *WUTn_CNT*.*count* register to 1.
6. Write the compare value to *WUTn_CMP*.*compare*.
7. Set *WUTn_CTRL*.*ten* to 1 to enable the timer.

The continuous mode timer period is calculated using *Equation 18-3*.

Equation 18-3: Continuous Mode Timer Period

$$Continuous\ Mode\ Timer\ Period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

18.4 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 18-2*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 18-2: Wakeup Timer Register Summary

Offset	Register Name	Description
[0x0000]	*WUTn_CNT*	Wakeup Timer Counter Register
[0x0004]	*WUTn_CMP*	Wakeup Timer Compare Register
[0x0008]	*WUTn_PWM*	Wakeup Timer PWM Register
[0x000C]	*WUTn_INTFL*	Wakeup Timer Interrupt Register
[0x0010]	*WUTn_CTRL*	Wakeup Timer Control Register
[0x0014]	*WUTn_NOLCMP*	Wakeup Timer Non-Overlapping Compare Register

18.4.1 Register Details

Table 18-3: Wakeup Timer Count Register

Wakeup Timer Count			WUTn_CNT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count Value** The current count value for the timer. This field increments as the timer counts. Reads of this register are always valid. Prior to writing this field, disable the timer by clearing the bit *WUTn_CTRL.ten*.

Table 18-4: Wakeup Timer Compare Register

Wakeup Timer Compare			WUTn_CMP	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The compare field meaning is determined by the specific mode of the timer. See the timer mode's detailed configuration section for compare usage and meaning.

Table 18-5: Wakeup Timer PWM Register

Wakeup Timer PWM			WUTn_PWM	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 18-6: Wakeup Timer Interrupt Register

Wakeup Timer Interrupt			WUTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	irq_clr	RW	0	**Timer Interrupt Flag** If set, this field indicates a wakeup timer interrupt condition occurred. Writing any value to this bit clears the wakeup timer's interrupt. 0: Wakeup Timer interrupt is not active. 1: Wakeup Timer interrupt occurred.

Table 18-7: Wakeup Timer Control Register

Wakeup Timer Control			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	DNM	0	**Reserved, Do Not Modify**
12	pwmckbd	DNM	0	**Reserved, Do Not Modify**
11	nollpol	DNM	0	**Reserved, Do Not Modify**
10	nolhpol	DNM	0	**Reserved, Do Not Modify**
9	pwmsync	DNM	0	**Reserved, Do Not Modify**
8	pres3	R/W	0	**Timer Prescaler Select MSB** See *WUTn_CTRL.pres* for details on this field's usage.
7	ten	R/W	0	**Timer Enable** 0: Timer disable 1: Timer enabled
6	tpol	DNM	0	**Reserved, Do Not Modify**

Wakeup Timer Control			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
5:3	pres	R/W	0	**Timer Prescaler Select** Sets the timer's prescaler value. The prescaler divides the RTC 's 32.768KHz input clock sets the timer's count clock as shown in *Equation 18-1*. The wakeup timer's prescaler setting is a 4-bit value with *pres3* as the most significant bit and *pres* as the three least significant bits. See *Table 18-1* for details.
2:0	tmode	R/W	0	**Timer Mode Select** Sets the timer's operating mode. 0: One-shot 1: Continuous 2 – 7: Reserved

Table 18-8: Wakeup Timer Non-Overlapping Compare Register

Wakeup Timer Non-Overlapping Compare			WUTn_NOLCMP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	DNM	0	**Reserved, Do Not Modify**

19. Watchdog Timer (WDT)

The watchdog timer (WDT) protects against corrupt or unreliable software, power faults, and other system-level problems which may place the IC into an improper operating state. The software must periodically write a special sequence to a dedicated register to confirm the application is operating correctly. Failure to reset the watchdog timer within a user-specified time frame can first generate an interrupt allowing the application the opportunity to identify and correct the problem. In the event the application cannot regain normal operation, as a last resort the watchdog timer can generate a system reset.

Some instances provide a windowed timer function. These instances support an additional feature that can detect watchdog timer resets that occur too early as well as too late (or never). This could happen if program execution is corrupted and is accidentally forced into a tight loop of code that contains a watchdog sequence. This would not be detected with a traditional WDT, because the end of the timeout periods would never be reached. A new set of "watchdog timer early" fields are available to support the lower limits required for windowing. Traditional watchdog timers can only detect a loss of program control that fails to reset the watchdog timer.

Each time the application performs a reset, as early as possible in the application software, the peripheral control register should be examined to determine if the reset was cause by a WDT late reset event or a WDT early reset event if the window function is supported. If so, the software should take the desired action as part of its restart sequence.

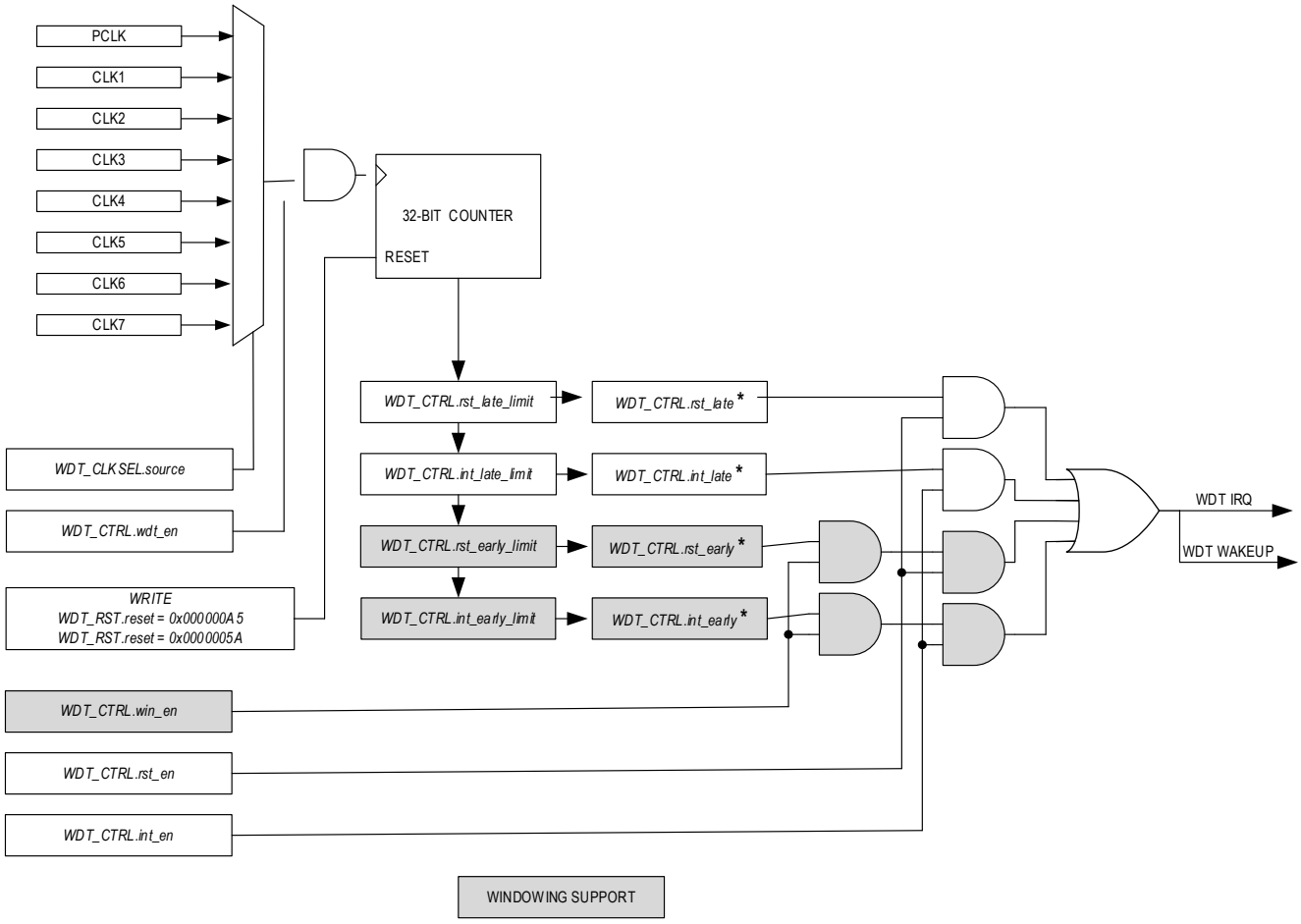
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

Features:

- Single ended (legacy) watchdog timeout
- Windowed mode adds lower-limit timeout settings to detect loss of control in tight code loops.
- Configurable clock source
- Configurable time-base
- Programmable upper and lower limits for reset and interrupts from 2^{16} to 2^{1327} time-base ticks
- New register to read the WDT counter register, simplifying code development

Figure 19-1 shows a high level block diagram of the WDT.

Figure 19-1: Windowed Watchdog Timer Block Diagram



* *INTERRUPT FLAGS ARE SET REGARDLESS OF THE ENABLED STATE OF WDTn_CTRL.win_en, WDTn_CTRL.wdt_int_en and WDTn_CTRL.wdt_rst_en.*

19.1 Instances

Table 19-1 shows the peripheral instances, available clock sources, and windowed watchdog support.

Table 19-1: MAX32655 WDT Instances Summary

Instance	Register Access Name	Window Support	External Clock	CLK0	CLK1	CLK2
WDT0	WDT0	Yes	N/A	PCLK	IBRO	-
LPWDT0	WDT1	Yes	N/A	IBRO	ERTC0	INRO

19.2 Usage

When enabled, *WDTn_CNT.count* increments once every t_{WDTCLK} period. During correct operation, the WDT periodically executes the feed sequence and resets the field *WDTn_CNT.count* to 0x0000 0000 within the target window.

The upper and lower limits of the target window are user-configurable to accommodate different applications and non-deterministic execution times within an application.

The WDT can generate interrupts and/or reset events in response to the WDT activity. Interrupts are typically configured to respond first to an event outside the target window. The approach is that a minor system event may have temporarily

delayed the execution of the feed sequence, so the event can be diagnosed in an interrupt routine and control returned to the system. When the WDT feed sequence occurs much earlier than expected or not at all, a reset event can be generated that forces the system to a known good state before continuing.

Traditional WDTs only detect execution errors that fail to perform the WDT feed sequence. If the counter reaches the WDT late interrupt threshold, the device attempts to regain program control by vectoring to the dedicated WDT ISR. The ISR should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

If the execution error prevents the successful execution of the ISR, the WDT continues to increment until the count reaches the WDT late reset threshold. The WDT generates a late reset event which sets the WDT late reset flag and generates a system interrupt.

Instances which support the window feature (*WDTn_CTRL*.*win_en* = 1) can generate a WDT early interrupt event if the WDT feed sequence occurs earlier than expected. In an analogous manner, the device attempts to regain program control by vectoring to the dedicated WDT ISR. The WDT ISR should reset the WDT counter, perform the desired recovery activity ,and then return execution to a known good address.

A WDT feed sequence that occurs earlier than the WDT early reset threshold indicates the execution error is significant enough to initiate a reset the device to correct the problem. The WDT generates an early reset event that sets the WDT late reset flag and generates a system interrupt.

The event flags are set regardless of the corresponding interrupt or reset enable. This includes the early interrupt and early event flags, even if the WDT is disabled (*WDTn_CTRL*.*win_en* = 0).

19.2.1 Using the WDT as a Long-Interval Timer

One application of the WDT is as a very long interval timer in ACTIVE mode. The timer can be configured to generate a WDT late interrupt event for as long as 2^{32} periods of the selected watchdog clock source. The WDT should not be enabled to generate WDT reset events in this application.

19.2.2 Using the WDT as a Long-Interval Wakeup Timer

The WDT can be used as a very long internal wakeup source. Another application of the WDT is as a very long interval wakeup source from *SLEEP*. Additionally, the LPWDT can be used as a wakeup source from *SLEEP*, *LPM*, and *UPM*. The timer can be configured to generate a WDT wakeup event for as long as 2^{32} periods of the selected watchdog clock source.

19.3 WDT Feed Sequence

The WDT feed sequence protects the system against unintentional altering of the WDT count and unintentional enabling or disabling of the timer itself.

Two consecutive write instructions to the *WDTn_RST*.*reset* field are required to reset the *WDTn_CNT*.*count* = 0. Global interrupts should be disabled immediately before, and reenabled after the writes, to ensure both writes to the *WDTn_RST*.*reset* field complete without interruption.

The feed sequence must also be performed immediately before enabling the WDT, to prevent an accidental triggering of the reset or interrupt as soon as the timer is enabled. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence

1. Disable interrupts.
2. In consecutive write operations:
   a. Write *WDTn_RST*.*reset*: 0x0000 00A5.
   b. Write *WDTn_RST*.*reset*: 0x0000 005A.
3. If desired, enable or disable the timer.
4. Re-enable interrupts.

19.4 WDT Events

Multiple events are supported as shown in *Table 19-2*. The corresponding event flag is set when the event occurs.

Typically the system is configured such that the late interrupt events occur prior to the late reset events, and early interrupts occur when the feed sequence has the least error from the target time prior to the early reset events.

The event flags are set even if the corresponding interrupt enable or reset enable are not enabled. This includes the early interrupt flag and early event flag even if the window feature is diabled (*WDTn_CTRL.win_en* = 0).

The software must clear the event flags before enabling the WDT.

Table 19-2: WDT Event Summary

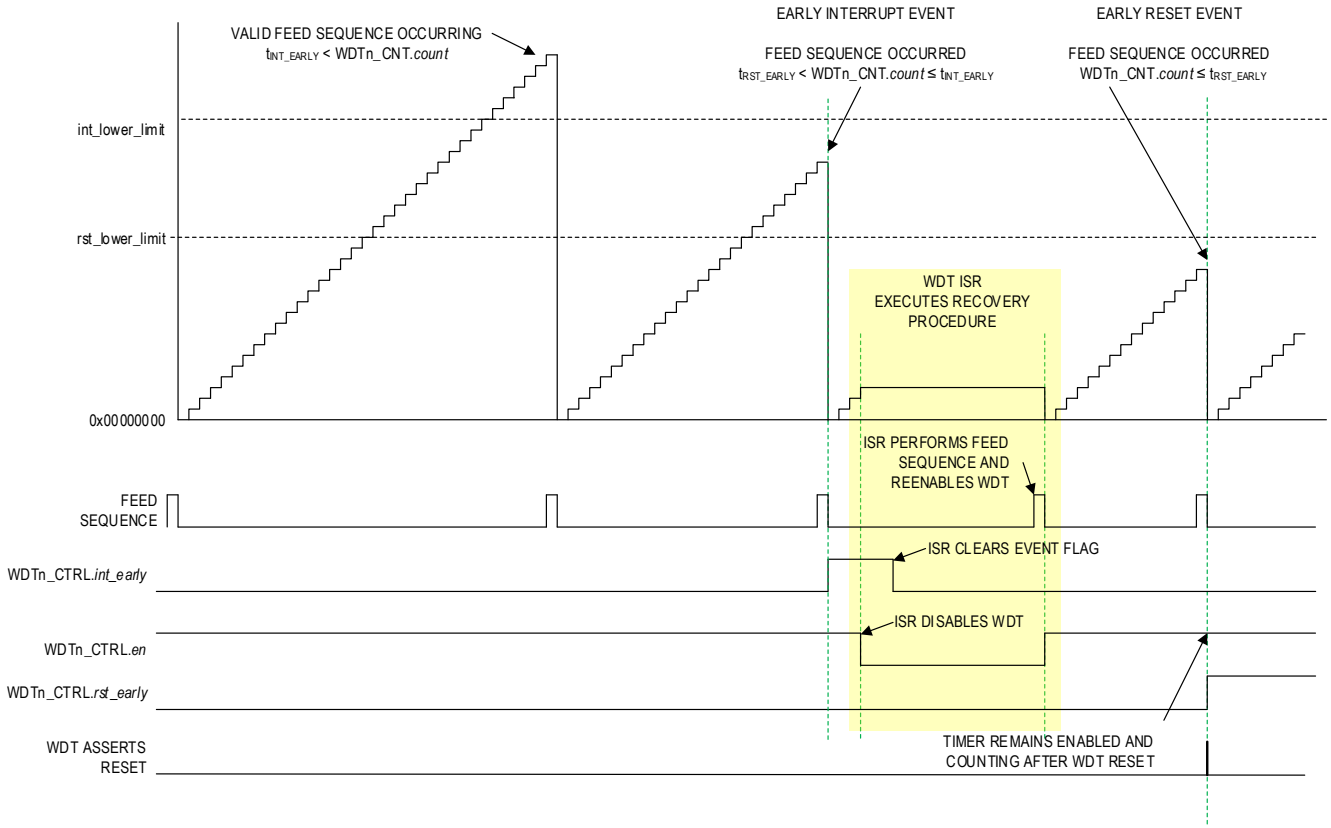
Event	Condition	Peripheral Interrupt Event Flag	Peripheral Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while *WDTn_CNT.count* < *WDTn_CTRL.rst_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.rst_early*	*WDTn_CTRL.wdt_rst_en*
Interrupt Late	*WDTn_CNT.count* = *WDTn_CTRL.int_late_val*	*WDTn_CTRL.int_late*	*WDTn_CTRL.wdt_int_en*
Reset Late	*WDTn_CNT.count* = *WDTn_CTRL.rst_late_val*	*WDTn_CTRL.rst_late*	*WDTn_CTRL.wdt_rst_en*
Timer Enabled	*WDTn_CTRL.clkrdy* 0 → 1	No event flags are set by a timer enabled event	

19.4.1 WDT Early Reset

The early reset event occurs if the software performs the WDT feed sequence while the WDT count is less than the reset late value (*WDTn_CNT.count* < *WDTn_CTRL.rst_late_val*).

Figure 19-2 shows the sequencing details associated with an early reset event.

Figure 19-2: WDT Early Interrupt and Reset Event Sequencing Details



The following occurs when a WDT early reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_early* to 1.
2. The hardware initiates a system reset.
   a. The hardware resets *WDTn_CNT.count* to 0x0000_0000 during the system reset event.
   b. The *WDTn_CTRL.en* and the *WDTn_CTRL.rst_early* fields are unaffected by a system reset.
3. After the system reset is complete, the WDT continues incrementing.

19.4.2 WDT Early Interrupt

The early interrupt event occurs if the software performs the WDT feed sequence while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* as shown in *Table 19-2*. *Figure 19-2* shows the sequencing details associated with an early reset event including:

- The sequencing details associated with an early interrupt event.
- The required functions performed by the WDT interrupt handler.

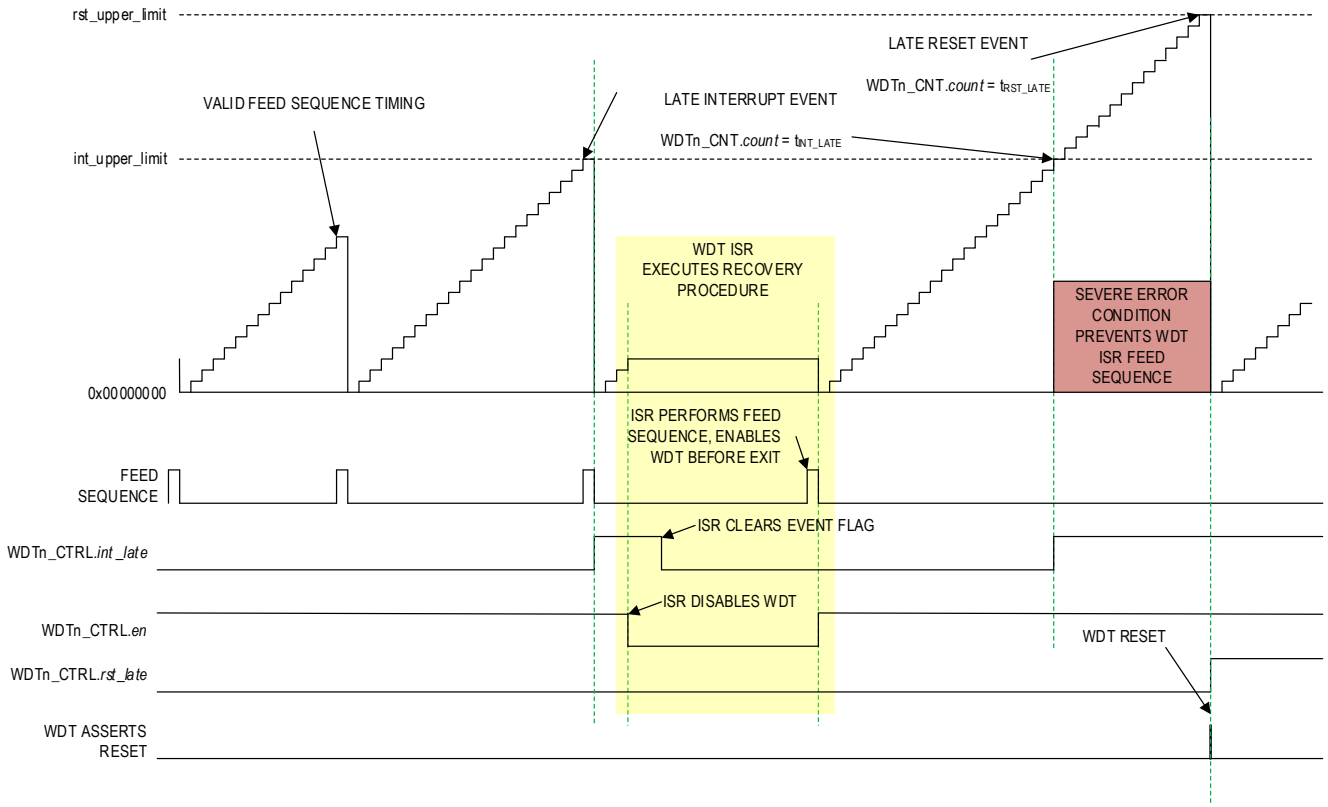
The following occurs when a WDT late interrupt event occurs:

1. The hardware sets *WDTn_CTRL.int_late* to 1.
2. The hardware initiates the WDT interrupt if enabled.

19.4.3 WDT Late Reset

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold as shown in *Table 19-2*. *Figure 19-3* shows the sequencing details associated with a late reset event.

Figure 19-3: WDT Late Interrupt and Reset Event Sequencing Details



The following occurs when a WDT late reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_late* to 1.
2. The hardware initiates a system reset:
   a. The hardware resets *WDTn_CNT.count* to 0x0000 0000 during the reset event.
   b. The *WDTn_CTRL.en* and *WDTn_CTRL.rst_late* fields are unaffected by a system reset.
3. After the hardware exits the system reset, the WDT continues incrementing after the system reset completes.

19.4.4 WDT Late Interrupt

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold as shown in *Table 19-2*. *Figure 19-3* shows the sequencing details associated with a late interrupt event including the required functions performed by the WDT interrupt handler.

The following occurs when WDT late interrupt event occurs:

1. The hardware sets *WDTn_CTRL.int_late* to 1.
2. The hardware initiates the WDT interrupt if enabled.

19.5 Initializing the WDT

The full procedure for configuring the WDT is as follows:

1. Execute the WDT feed sequence and disable the WDT:
   a. Disable global interrupts.

b. Write *WDTn_RST.reset* to 0x0000 00A5.
c. Write *WDTn_RST.reset* to 0x0000 005A.
d. The hardware resets the WDT count (*WDTn_CNT.count* = 0x0000 0000).
e. Set *WDTn_CTRL.en* to 0 to disable the WDT.

2. Verify the peripheral is disabled before proceeding:
   a. Poll *WDTn_CTRL.clkrdy* until it reads 1.
3. Set *WDTn_CTRL.clk_rdy_ie* = 1 to generate a WDT enabled interrupt event.
4. Re-enable global interrupts.
5. Configure *WDTn_CLKSEL.source* to select the clock source.
6. Configure the standard thresholds:
   a. Configure *WDTn_CTRL.int_late_thd* to the desired threshold for the WDT late interrupt event.
   b. Configure *WDTn_CTRL.rst_late_val* to the desired threshold for the WDT late reset event.
7. If using the optional windowed WDT feature:
   a. Set *WDTn_CTRL.win_en* = 1 to enable the windowed WDT feature.
   b. Configure *WDTn_CTRL.int_early_val* to the desired threshold for the WDT early interrupt event.
   c. Configure *WDTn_CTRL.rst_early_val* to the desired threshold for the WDT early reset event.
8. Set *WDTn_CTRL.wdt_int_en* to generate an interrupt when a WDT late interrupt event occurs. If *WDTn_CTRL.win_en* = 1 an interrupt is generated by both a WDT late interrupt event and also a WDT early interrupt event.
9. Set *WDTn_CTRL.wdt_rst_en* to generate an interrupt when a WDT late reset event occurs. If *WDTn_CTRL.win_en* = 1 an interrupt is generated by a WDT late reset event and also a WDT early reset event.
10. Execute the WDT feed sequence and enable the WDT:
    a. Disable global interrupts.
    b. Write *WDTn_RST.reset* to 0x0000 00A5.
    c. Write *WDTn_RST.reset* to 0x0000 005A. The hardware resets *WDTn_CNT.count* = 0x0000 0000.
    d. Set *WDTn_CTRL.en* to 1 to enable the WDT.
11. Verify the peripheral is enabled before proceeding:
    a. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
12. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled event interrupt.
13. Re-enable global interrupts.

19.6 Resets

The WDT is a critical safety feature and most of the fields are reset by a POR or system reset events only; however, the enable field (*WDTn_CTRL.en*) and the interrupt flag fields are not reset by a system reset event.

19.7 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 19-3*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 19-3: WDT Register Summary

Offset	Register	Name
[0x0000]	*WDTn_CTRL*	WDT Control Register
[0x0004]	*WDTn_RST*	WDT Reset Register
[0x0008]	*WDTn_CLKSEL*	WDT Clock Select Register
[0x000C]	*WDTn_CNT*	WDT Count Register

19.7.1 Register Details

Table 19-4: WDT Control Register

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rst_late	R/W	0	**Reset Late Event** A watchdog reset event occurred after the time specified in *WDTn_CTRL.rst_late_val*. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred after *WDTn_CTRL.rst_early_val*.
30	rst_early	R/W	0	**Reset Early Event** A watchdog reset event occurred before the time specified in the *WDTn_CTRL.rst_early_val* field. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred prior to the time specified in the *WDTn_CTRL.rst_early_val* field.
29	win_en	R/W	0	**Window Function Enable** 0: Disabled. WDT recognizes interrupt late and reset late events, supporting legacy implementations. 1: Enabled
28	clkrdy	R	0	**Clock Status** This field is cleared to 0 by the hardware when the software changes the state of the *WDTn_CTRL.en* field. The hardware sets this field to 1 when the change to the requested enable or disable is complete. 0: WDT clock is off 1: WDT clock is on
27	clkrdy_ie	R/W	0	**Clock Switch Ready Interrupt Enable** This interrupt prevents the application firmware from needing to poll the *WDTn_CTRL.clkrdy* field to determine when the WDT clock is ready. When the *WDTn_CTRLclkrdy* field transitions from 1 to 0 this interrupt signals the transition is complete. 0: Disabled 1: Enabled
26:24	-	RO	0	**Reserved**

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
23:20	rst_early_val	R/W	0	**Reset Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$
19:16	int_early_val	R/W	0	**Interrupt Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$
15:13	-	RO	0	**Reserved**
12	int_early	R/W	0	**Interrupt Early Flag** A feed sequence was performed earlier than the time determined by the *WDTn_CTRL.int_early_thd* field. This flag is set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
11	wdt_rst_en	R/W	0	**WDT Reset Enable** 0: Disabled 1: Enabled
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled 1: Enabled

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
9	int_late	R/W	0	**Interrupt Late Flag** A watchdog feed sequence did not occur before the time determined by the *WDTn_CTRL.int_late_val* field. 0: No interrupt event 1: Interrupt event occurred *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
8	en	R/W	0	**WDT Enable** This field enables/disables the WDTCLK into the peripheral. *WDTn_CTRL.count* holds its value while the WDT is disabled. The WDT feed sequence must be performed immediately before any change to this field. 0: Disabled 1: Enabled
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$
3:0	int_late_val	R/W	0	**Interrupt Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$

Table 19-5: WDT Reset Register

WDT Reset			WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved** Do not modify this field.
7:0	reset	R/W	0†	**Reset Watchdog Timer Count** Writing the WDT feed sequence in two consecutive write instructions to this register resets the internal counter to 0x0000 0000. 1. Write *WDTn_RST.reset*: 0x0000 00A5 2. Write *WDTn_RST.reset*: 0x0000 005A Writes to the *WDTn_CTRL.en* field, which enable or disable the WDT, must be the next instruction following the WDT feed sequence. †*Note: This field is set to 0 on a POR and is not affected by other resets.*

Table 19-6: WDT Clock Source Select Register

WDT Clock Source Select			WDTn_CLKSEL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	source	R/W	0†	**Clock Source Select** See *Table 19-1* for the available clock options. 0: CLK0 1: CLK1 2: CLK2 3: CLK3 4: CLK4 5: CLK5 6: CLK6 7: CLK7 †*Note: This field is only reset on a POR and unaffected by other resets.*

Table 19-7: WDT Count Register

WDT Count			WDTn_CNT	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R	0	**WDT Counter** The counter value for debug. This register is reset by system reset, as well as the watchdog feeding sequence. When the WDT clock is off, the feeding sequence generates an asynchronous reset of 1 PCLK width. When the WDT clock is on, the feeding sequence generates a synchronous reset that is handshake to the WDT clock domain.

20. Pulse Train Engine (PT)

Each independent pulse train engine operates either in square wave mode, which generates a continuous 50% duty-cycle square wave, or pulse train mode, which generates a continuous programmed bit pattern from 2- to 32-bits in length. Pulse train engines are used independently or may be synchronized together to generate signals in unison. The frequency of each generated output can be set separately based on a divisor of the PCLK.

20.1 Instances

The device provides four instances of the pulse train engine peripheral.

- PT0 to PT3

All peripheral registers share a common register set.

20.2 Features

The pulse train outputs with individually programmable modes, patterns, and output enables. The pulse train engine uses the Peripheral Clock (PCLK), $f_{PTE\_CLK} = f_{PCLK}$, ensuring all pulse train outputs use the same clock source.

- Independent or synchronous pulse train output operation
- Atomic Enable and Atomic Disable
- Synchronous enable or disable of pulse train output(s) without modification to non-intended pulse train outputs
- Multiple Output Modes:
  - Square Wave Output mode generates a repeating square wave (50% duty cycle)
  - Pattern Output mode for generating a customizable output wave based on a programmable bit pattern from 2 to 32 output cycles
- Global clock for all generated outputs
- Individual rate configuration for each pulse train output
- Configuration registers are modifiable while the pulse train engine is running
- Pulse train outputs can be halted and resumed at the same point

20.3 Engine

The pulse train engine uses the Peripheral Clock as the peripheral input clock. Each pulse train output is individually configurable and independently controlled.

The following sections describe the available configuration options for each individual pulse train output.

20.3.1 Pulse Train Output Modes

Each pulse train output supports the following modes:

- Pulse train
- Bit pattern length
- Square wave

20.3.1.1 Pulse Train Mode

When Pulse Train *n* (*PTn*) is configured in pulse train mode, the configuration also includes the bit length (up to 32-bits) of the custom pulse train. This is configured using the 5-bit field *PTn_RATE_LENGTH.mode* as follows:

PTn_RATE_LENGTH.mode = 1 (*PTn* configured in Square Wave mode)

PTn_RATE_LENGTH.mode > 1 (*PTn* configured in pulse train mode. The value of mode is the pattern bit length.)

PTn_RATE_LENGTH.mode = 0 (*PTn* configured for pulse train mode, 32-bit pattern)

20.3.1.2 In Pulse Train Mode, Set the Bit Pattern

If an output is set to pulse train mode, then configure a custom bit pattern from 2-bits to 32-bits in length in the 32-bit register *PTn_TRAIN*. The pattern is shifted out least significant bit (LSB) first. If the output is configured in Square Wave mode, then the *PTn_TRAIN* register is ignored.

Equation 20-1: Pulse Train Mode Output Function

$$PTn\_TRAIN = [Bit\ pattern\ for\ PTn]$$

20.3.1.3 Synchronize Two or More Outputs, if Needed

The write-only register *PTG_RESYNC* "PT Global Resync" allows two or more outputs to be reset and synchronized. Write to any bit in *PTG_RESYNC* to simultaneously reset any outputs in pulse train mode to the beginning of the pattern (the LSB) set in the *PTn_TRAIN* bit-pattern register, and reset the output to 0 for outputs in Square Wave mode.

20.3.1.4 Pulse Train Loop Mode

By default, a pulse train engine runs indefinitely until it is disabled by the software.

A pulse train engine can be configured to repeat its pattern a specified number of times, called Loop mode. To select Loop mode, write a non-zero value to the 16-bit field *PTn_LOOP.count*. When the pulse train engine is enabled, this field decrements by 1 each time a complete pattern is shifted through the output pin. When the count reaches 0, the output is halted, and the corresponding flag in the *PTG_INTFL* register is set.

20.3.1.5 Pulse Train Loop Delay

If the pulse train is configured in Loop mode, a delay can be inserted after each repeated output pattern. To enable a delay, write the 12-bit field *PTn_LOOP.delay* with the number of Peripheral Clock cycles to delay between the most significant bit (MSB) of the last pattern to the least-significant bit (LSB) of the next pattern. During this delay, the output is held at the MSB of the last pattern. If the loop counter has not reached 0, then it is decremented when the next pattern starts.

20.3.1.6 Pulse Train Automatic Restart Mode

When an engine in pulse train mode is in Loop mode and stops when the loop count reaches 0, this is called a Stop Event. A Stop Event can optionally trigger one or more pulse trains to restart from the beginning. This is called Automatic Restart mode. While only pulse train engines operating in pulse train mode can operate in Loop mode and can optionally restart a pulse train engine, Automatic Restart mode can trigger pulse train engines operating in pulse train mode or in Square Wave mode.

If a running pulse train engine is triggered by another pulse train's Stop Event, Automatic Restart restarts the running pulse train engine from the beginning of its pattern. If a pulse train engine is triggered by another pulse train's Stop Event, and it is not running, Automatic Restart sets the enable bit to 1, and starts the pulse train engine.

The settings for this mode are contained in the *PTn_RESTART* register for each pulse train engine. Note that the configuration for automatic restart is set using the pulse engine(s) triggered by the automatic restart, not the pulse train engine(s) that trigger the automatic restart. For example, the PT8_RESTART register configures which pulse train engine triggers PT8 to restart.

Each pulse train engine can be configured to perform an Automatic Restart when it detects a Stop Event from one or two pulse trains.

If *PTn_RESTART.on_pt_x_loop_exit* = 1, then pulse train engine *n* automatically restarts when it detects a Stop Event from pulse train *x*, where *x* is the value in the 5-bit field *PTn_RESTART.pt_x_select*.

If *PTn_RESTART.on_pt_y_loop_exit* = 1, then pulse train engine *n* automatically restarts when it detects a Stop Event from pulse train *y*, where *y* is the value in 5-bit field *PTn_RESTART.pt_y_select*.

A pulse train engine can be configured to restart on its own Stop Event, allowing the pulse train to run indefinitely.

Each individual pulse train can be configured for:

- No automatic restart.
- Automatic restart triggered by a stop event from Pulse Train *x* only.
- Automatic restart triggered by a stop event from Pulse Train *y* only.
- Automatic restart triggered by a stop event from both Pulse Train *x* and Pulse Train *y.*

20.4 Enabling and Disabling a Pulse Train Output

The *PTG_ENABLE* register is used to enable and disable each of the individual pulse train outputs. Enable a given pulse train output by setting the respective bit in the *PTG_ENABLE* register. Halt a pulse train output by clearing the respective bit in the *PTG_ENABLE* register.

Note: Prior to changing a pulse train output's configuration, the corresponding pulse train output should be halted to prevent unexpected behavior.

20.5 Atomic Pulse Train Output Enable and Disable

Deterministic enable and disable operations are critical for pulse train outputs that must be synchronized in an application. The *PTG_ENABLE* register does not perform atomic access directly. Atomic operations are supported using the registers *PTG_SAFE_EN*, *PTG_SAFE_DIS*.

For most pulse train peripherals, enabling and disabling individual pulse trains is performed by setting and clearing bits in the global enable/disable register, which for this peripheral is *PTG_ENABLE*. For most Arm Cortex-M microcontrollers, this is usually done by bit banding. Because bit banding performs a read, modify, write (RMW), some pulse trains could start and end during the RMW operation, often with unpredictable results.

To ensure safe and predictable operation, two additional registers are used to enable and disable the outputs.

20.5.1 Pulse Train Atomic Enable

The global safe enable register, *PTG_SAFE_EN*, is a write-only register. To safely enable outputs without a RMW, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be enabled. This immediately sets to 1 the corresponding bits in the *PTG_ENABLE* register to 1, which enables the corresponding pulse train engine. Writing a 0 to any bit position in the *PTG_SAFE_EN* register has no effect on the state of the corresponding pulse train enable bit. If the corresponding pulse train engine is already enabled and running, writing a 1 to that bit position in the *PTG_SAFE_EN* register has no effect.

20.5.2 Pulse Train Atomic Disable

The global safe disable register, *PTG_SAFE_DIS*, is a write-only register for disabling a pulse train engine without performing a RMW. To safely disable pulse train engines, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be disabled. This immediately clears to 0 the corresponding bits in *PTG_ENABLE*, which disables the corresponding pulse train engines. Writing a 0 to any bit position in the *PTG_SAFE_DIS* register has no effect on the state of the corresponding pulse train enable bit.

Bit banding is not supported for the *PTG_ENABLE*, *PTG_SAFE_EN*, and *PTG_SAFE_DIS* registers and can have unpredictable results.

20.6 Halt and Disable

Once a pulse train engine is enabled and running, it continues to run until one of the following events stops the output:

- The corresponding enable bit in the *PTG_ENABLE* register is cleared to 0 to halt the output.
- A 1 is written to the corresponding disable bit in the *PTG_SAFE_DIS* register to halt the output.
- The corresponding resync bit in the *PTG_RESYNC* register is cleared to 0 to halt and reset the output.
- *PTn_LOOP* was initialized to a non-zero value, and the loop count has reached 0 (this has no effect in Square Wave mode; it only applies to pulse train mode).

When a pulse train is halted, the corresponding enable bit in *PTG_ENABLE* is automatically cleared to 0.

20.7 Interrupts

Each pulse train can generate an interrupt only if it is configured in pulse train mode, and the loop counter, *PTn_LOOP*, was initialized to a non-zero number. When *PTn_LOOP* counts down to 0, the corresponding status flag in the *PTG_INTFL* register is set. If the corresponding interrupt enable bit in the *PTG_INTEN* register is set, the event also generates an interrupt.

20.8 Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 20-1: Pulse Train Engine Register Summary

Offset	Register	Description
[0x0000]	*PTG_ENABLE*	PT Global Enable/Disable Control Register
[0x0004]	*PTG_RESYNC*	PT Global Resync Register
[0x0008]	*PTG_INTFL*	PT Stopped Global Status Flags Register
[0x000C]	*PTG_INTEN*	PT Global Interrupt Enable Register
[0x0010]	*PTG_SAFE_EN*	PT Global Safe Enable Register
[0x0014]	*PTG_SAFE_DIS*	PT Global Safe Disable Register
[0x0020]	*PT0_RATE_LENGTH*	PT0 Configuration Register
[0x0024]	*PT0_TRAIN*	PT0 Pulse Train Mode Bit Pattern Register
[0x0028]	*PT0_LOOP*	PT0 Loop Control Register
[0x002C]	*PT0_RESTART*	PT0 Automatic Restart Register
[0x0030]	*PT1_RATE_LENGTH*	PT1 Configuration Register
[0x0034]	*PT1_TRAIN*	PT1 Pulse Train Mode Bit Pattern Register
[0x0038]	*PT1_LOOP*	PT1 Loop Control Register
[0x003C]	*PT1_RESTART*	PT1 Automatic Restart Register
[0x0040]	*PT2_RATE_LENGTH*	PT2 Configuration Register
[0x0044]	*PT2_TRAIN*	PT2 Pulse Train Mode Bit Pattern Register
[0x0048]	*PT2_LOOP*	PT2 Loop Control Register
[0x004C]	*PT2_RESTART*	PT2 Automatic Restart Register
[0x0050]	*PT3_RATE_LENGTH*	PT3 Configuration Register
[0x0054]	*PT3_TRAIN*	PT3 Pulse Train Mode Bit Pattern Register
[0x0058]	*PT3_LOOP*	PT3 Loop Control Register
[0x005C]	*PT3_RESTART*	PT3 Automatic Restart Register

20.8.1 Register Details

20.8.1.1 Pulse Train Engine Global Enable/Disable Register

This register enables each of the individual Pulse Trains. Write a 1 to the induvial Pulse Train enable bits to enable the corresponding Pulse Train. When, for a given Pulse Train, the *PTn_LOOP.count* loop counter is set to a non-zero number, when the loop counter reaches zero then the given Pulse Train engine stops, and the corresponding enable bit in this register is cleared by the hardware.

Table 20-2: Pulse Train Engine Global Enable/Disable Register

PT Global Enable/Disable Control			PTG_ENABLE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**Enable PT3** 0: Disable 1: Enable *Note: Disabling an active pulse train halts the output and does not generate a Stop Event.*
2	pt2	R/W	0	**Enable PT2** 0: Disable 1: Enable *Note: Disabling an active pulse train halts the output and does not generate a Stop Event.*
1	pt1	R/W	0	**Enable PT1** 0: Disable 1: Enable *Note: Disabling an active pulse train halts the output and does not generate a Stop Event.*
0	pt0	R/W	0	**Enable PT0** 0: Disable 1: Enable *Note: Disabling an active pulse train halts the output and does not generate a Stop Event.*

Table 20-3: Pulse Train Engine Resync Register

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Resync Control for PT3** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave mode, the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
2	pt2	WO	-	**Resync Control for PT2** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave, mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
1	pt1	WO	-	**Resync Control for PT1** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave, mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
0	pt0	WO	-	**Resync Control for PT0** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave, mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

Table 20-4:Pulse Train Engine Stopped Interrupt Flag Register

PT Stopped Interrupt Flag Register			PTG_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W1C	0	**PT3 Stopped Status Flag** This bit is set to 1 by the hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
2	pt2	R/W1C	0	**PT2 Stopped Status Flag** This bit is set to 1 by the hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
1	pt1	R/W1C	0	**PT1 Stopped Status Flag** This bit is set to 1 by the hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

PT Stopped Interrupt Flag Register			PTG_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
0	pt0	R/W1C	0	**PT0 Stopped Status Flag** This bit is set to 1 by the hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

Table 20-5: Pulse Train Engine Interrupt Enable Register

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**PT3 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled 1: Enabled
2	pt2	R/W	0	**PT2 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled 1: Enabled
1	pt1	R/W	0	**PT1 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled 1: Enabled
0	pt0	R/W	0	**PT0 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled 1: Enabled

20.8.1.2 Pulse Train Engine Safe Enable Register

A 32-bit value written to this register performs an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

Table 20-6: Pulse Train Engine Safe Enable Register

Pulse Train Engine Safe Enable Register			PTG_SAFE_EN	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Safe Enable Control for PT3** Writing a 1 to this field sets the *PTG_ENABLE.pt3* field to 1. 0: No effect 1: Enable corresponding pulse train.
2	pt2	WO	-	**Safe Enable Control for PT2** Writing a 1 to this field sets the *PTG_ENABLE.pt2* field to 1. 0: No effect 1: Enable corresponding pulse train.

Pulse Train Engine Safe Enable Register		PTG_SAFE_EN		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
1	pt1	WO	-	**Safe Enable Control for PT1** Writing a 1 to this field sets the *PTG_ENABLE.pt1* field to 1. 0: No effect 1: Enable corresponding pulse train.
0	pt0	WO	-	**Safe Enable Control for PT0** Writing a 1 to this field sets the *PTG_ENABLE.pt0* field to 1. 0: No effect 1: Enable corresponding pulse train.

20.8.1.3 Pulse Train Engine Safe Disable Register

A 32-bit value written to this register performs an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

Table 20-7: Pulse Train Engine Safe Disable Register

Pulse Train Engine Safe Disable Register		PTG_SAFE_DIS		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Safe Disable Control for PT3** Writing a 1 to this field clears the *PTG_ENABLE.pt3* field. 0: No effect 1: Disable corresponding pulse train.
2	pt2	WO	-	**Safe Disable Control for PT2** Writing a 1 to this field clears the *PTG_ENABLE.pt2* field. 0: No effect 1: Disable corresponding pulse train.
1	pt1	WO	-	**Safe Disable Control for PT1** Writing a 1 to this field clears the *PTG_ENABLE.pt1* field. 0: No effect 1: Disable corresponding pulse train.
0	pt0	WO	-	**Safe Disable Control for PT0** Writing a 1 to this field clears the *PTG_ENABLE.pt0* field. 0: No effect 1: Disable corresponding pulse train.

20.8.1.4 Pulse Train Registers

Table 20-8: Pulse Train Engine Configuration Register

Pulse Train *n* Configuration Register		PTn_RATE_LENGTH		[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27	mode	R/W	1	**Square Wave or Pulse Train Output Mode** Sets either pulse train mode with length, or Square Wave mode. 0: Pulse train mode, 32-bits long. 1: Square Wave mode. 2: Pulse train mode, 2-bits long. 3: Pulse train mode, 3-bits long. …: … 31: Pulse train mode, 31-bits long. *Note: If this field is set to 1, square wave mode, the PTn_TRAIN register is not used.*

Pulse Train *n* Configuration Register			PTn_RATE_LENGTH	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
26:0	rate_control	R/W	0	**Pulse Train Enable and Rate Control** Defines the rate at which the output for PT*n* changes state by setting the divisor of the PT clock. Setting this field to 0 disables the PT*n*. For all other values, the following equation is used to calculate the rate: $f_{PTn} = \frac{f_{PTE\_CLK}}{2^{rate\_control} - 1}$ 0: Output halted. 1: $f_{PTn} = f_{PTE\_CLK}$ 2: $f_{PTn} = f_{PTE\_CLK}/_2$ 3: $f_{PTn} = f_{PTE\_CLK}/_4$ …

Table 20-9: Pulse Train Mode Bit Pattern Register

Pulse Train Mode Bit Pattern			PTn_TRAIN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ptn_train	R/W	0	**Pulse Train Mode Bit Pattern** Write the repeating bit pattern that is shifted out, LSB first, when configured in pulse train mode. Set the bit pattern length with the *PTn_RATE_LENGTH.mode* field. *Note: This register is ignored in Square Wave mode.* *Note: 0x0000 0000 and 0x0001 0000 are invalid values for this register.*

Table 20-10: Pulse Train n Loop Configuration Register

Pulse Train Loop Configuration			PTn_LOOP	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:28	-	RO	0	**Reserved**
27:16	delay	R/W	0	**Pulse Train Delay Between Loops** Sets the delay, in number of peripheral clock cycles, that the output pauses between loops. The *PTn_LOOP.count* is decremented after the delay. *Note: This field is ignored if firmware writes 0 to PTn_LOOP.count field.*
15:0	count	R/W	0	**Pulse Train Loop Countdown** Sets the number of times a pulse train pattern is repeated until it automatically stops. Reading this field returns the number of loops remaining. When this field counts down to zero, the corresponding pulse train's interrupt flag is set in the *PTG_INTFL* register. Writing this field to 0 to repeat the pulse train pattern indefinitely. *Note: This field is ignored in square wave mode.*

Table 20-11: Pulse Train n Automatic Restart Configuration Register

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
15	on_pt_y_loop_exit	R/W	0	**Enable Automatic Restart for This Pulse Train on PTy Stop Event** 0: Disable automatic restart. 1: When PTy has a stop event, automatically restart this pulse train from the beginning of its pattern.
14:11	-	RO	0	**Reserved**
12:8	pt_y_select	R/W	0	**Select PTy** Select the pulse train number to be associated with PTy. This engine must be in pulse train mode. 0: PT0 1: PT1 2: PT2 3: PT3 4 - 31: Reserved
7	on_pt_x_loop_exit	R/W	0	**Enable Automatic Restart for this Pulse Train on a *PTn* Stop Event** 0: Disable automatic restart 1: When *PTn* has a stop event, automatically restart the pulse train from the beginning of its pattern.
6:5	-	RO	0	**Reserved**
4:0	pt_x_select	R/W	0	**Select *PTn*** Select the pulse train number to be associated with *PTn*. This engine must be in pulse train mode. 0: PT0 1: PT1 2: PT2 3: PT3 4 - 31: Reserved

21. CRC

The cyclic redundancy check (CRC) engine performs CRC calculations on data stored in SRAM. The CRC engine cannot directly perform a CRC of data stored in flash memory.

The features include:

- User-definable polynomials up to x^{32} (33 terms).
- DMA support.
- Optional automatic byte swap of data input and calculated output.
- Most-significant bit or least-significant bit data input and calculated output.

An *n*-bit CRC can detect the following types of errors:

- Single-bit errors.
- Two-bit errors for block lengths less than 2^k where k is the order of the longest irreducible factor of the polynomial.
- Odd numbers of errors for polynomials with the parity polynomial (x+1) as one of its factors (polynomials with an even number of terms).
- Burst errors less than *n*-bits.

In general, all but 1 out of 2^n errors are detected:

- 99.998% for a 16-bit CRC.
- 99.99999998% for a 32-bit CRC.

21.1 Instances

Instances of the peripheral are listed in *Table 21-1*.

Table 21-1: MAX32655 CRC Instances

Instance	Maximum Number of Terms	DMA Support	Big- and Little-Endian
CRC	33 (2^{32})	Yes	Yes

21.2 Usage

A CRC value is often appended to the end of a data exchange between a transmitter and receiver. The transmitter appends the calculated CRC to the end of the transmission. The receiver independently calculates a CRC on the data it received, the result should be a known constant if the data and CRC were received error-free.

The software must configure the CRC polynomial, the starting CRC value, and endianness of the data. Once configured, the firmware or the standard DMA engine transfers the data in either 8-bit, 16-bit, or 32-bit words to the CRC engine by writing to the *CRC_DATAIN32.data* field. The hardware automatically sets the *CRC_CTRL.busy* field to 1 while the CRC engine is calculating a CRC over the input data; the software must not access any of the CRC registers until the *CRC_CTRL.busy* field reads 0. The CRC value, *CRC_VAL.value*, is updated by the hardware after each write of data to the *CRC_DATAIN32.data* field. The software or the standard DMA engine must track the data transferred to the CRC engine to determine when the CRC is finalized.

Because the receiving end calculates a new CRC on both the data and received CRC, send the received CRC in the correct order, so the highest-order term of the CRC is shifted through the generator first. Because data is typically shifted through the generator LSB first, this means the CRC is reversed bitwise, with the highest-order term of the remainder in the LSB position. Software CRC algorithms typically manage this by calculating everything backwards. They reverse the polynomial and do right shifts on the data. The resulting CRC ends up being bit swapped and in the correct format.

By default, the CRC is calculated using the LSB first (*CRC_CTRL.msb* = 0.) When calculating the CRC using MSB first data, the software must set *CRC_CTRL.msb* to 1.

When calculating the CRC on data LSB first, the polynomial should be reversed so that the coefficient of the highest power term is in the LSB position. The largest term, x^n, is implied (always one) and should be omitted when writing to the *CRC_POLY* register. This is necessary because the polynomial is always one bit larger than the resulting CRC, so a 32-bit CRC has a polynomial with 33 terms (x^0 … x^{32}).

Table 21-2: Organization of Calculated Result in CRC_VAL.value

CRC_CTRL.msb	CRC_CTRL.byte_swap_out	Order
0	0	The CRC value returned is the raw value
1	0	The CRC value returned is mirrored, but not byte swapped
0	1	The CRC value returned is byte swapped, but not mirrored
1	1	The CRC value returned is mirrored and then byte swapped

By default, the CRC engine calculates the CRC on the LSB of the data first.

The CRC can be calculated on the MSB of the data first by setting the *CRC_CTRL.msb* field to 1. The CRC polynomial register, *CRC_POLY*, must be left justified. The hardware implies the MSB of the polynomial just as it does when calculating the CRC LSB first. The LSB position of the polynomial must be set; this defines the length of the CRC. The initial value of the CRC, *CRC_VAL.value*, must also be left justified. When the CRC calculation is complete using MSB first, the software must right shift the calculated CRC value, *CRC_VAL.value*, by right shifting the output value if the CRC polynomial is less than 32-bits.

21.3 Polynomial Generation

The CRC can be configured for any polynomial up to x^{32} (33 terms) by writing to the *CRC_POLY.poly* field. *Table 21-3* shows common CRC polynomials.

The reset value of the *CRC_POLY.poly* field is the *CRC-32 Ethernet* polynomial. This polynomial is used by Ethernet and file compression utilities such as zip or gzip.

Note: Only write to the CRC polynomial register, CRC_POLY.poly, when the CRC_CTRL.busy field is 0.

Table 21-3: Common CRC Polynomials

Algorithm	Polynomial Expression	Order	Polynomial
CRC-32 Ethernet	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$	LSB	0xEDB8 8320
CRC-CCITT	$x^{16}+x^{12}+x^5+x^0$	LSB	0x0000 8408
CRC-16	$x^{16}+x^{15}+x^2+x^0$	LSB	0x0000 A001
USB Data	$x^{16}+x^{15}+x^2+x^0$	MSB	0x8005 0000
Parity	x^1+x^0	LSB	0x0000 0001

21.4 Calculations Using Firmware Writes to FIFO

The software can perform CRC calculations by writing directly to the *CRC_DATAIN32.data* field. Each write to the *CRC_DATAIN32.data* field triggers the hardware to compute the CRC value. The software is responsible for loading all data for the CRC into the *CRC_DATAIN32.data* field. When complete, the result is read from the *CRC_VAL.value* field.

Use the following procedure to calculate a CRC value by writing to the FIFO:

1. Disable the CRC peripheral by setting the field *CRC_CTRL.en* to 0.
2. Configure the following fields for the desired input and output data formats:
   a. *CRC_CTRL.msb*
   b. *CRC_CTRL.byte_swap_in*
   c. *CRC_CTRL.byte_swap_out*

3. Configure the CRC polynomial by writing to the *CRC_POLY.poly* field.
4. Set the CRC initial value by writing to the *CRC_VAL.value* field.
5. Set the *CRC_CTRL.en* field to 1 to enable the peripheral.
6. Write a value to be processed to the *CRC_DATAIN32.data* field.
    a. The hardware automatically sets the *CRC_CTRL.busy* field to 1 while performing the CRC.
7. At the end of the calculation, the hardware automatically:
    a. Clears the *CRC_CTRL.busy* field to 0.
    b. Loads the computed CRC value into the *CRC_VAL.value* field.
8. Repeat steps 6 and 7 until all input data is complete.
9. Disable the CRC peripheral by clearing the *CRC_CTRL.en* field to 0.
10. Read the CRC value from the *CRC_VAL.value* field.

21.5 Calculations Using DMA

The CRC engine requests new data from the DMA controller while the fields *CRC_CTRL.en* and *CRC_CTRL.dma_en* are both set to 1. Enable the corresponding DMA interrupt to receive an interrupt event when the CRC engine completes the CRC of the input data.

Use the following procedure to calculate a CRC value using the DMA:

1. Set *CRC_CTRL.en* = 0 to disable the peripheral.
2. Configure the DMA
    a. Set *CRC_CTRL.dma_en* = 1 to enable DMA mode.
3. Configure the peripheral for the required input and output data formats using the following fields:
    a. *CRC_CTRL.msb*
    b. *CRC_CTRL.byte_swap_in*
    c. *CRC_CTRL.byte_swap_out*
4. Configure the CRC polynomial by writing to the *CRC_POLY.poly* field.
5. Set the CRC initial value by writing to the *CRC_VAL.value* field.
6. Set the *CRC_CTRL.en* field to 1 to enable the peripheral.
    a. The hardware automatically clears the *CRC_CTRL.busy* field to 0.
    b. The DMA loads data into the *CRC_DATAIN32.data* field.
7. When the DMA operation completes, the hardware automatically:
    a. Clears the *CRC_CTRL.busy* field to 0.
    b. Loads the new CRC value into the *CRC_VAL.value* field.
    c. Generates a DMA interrupt request.
8. Disable the CRC peripheral by clearing the *CRC_CTRL.en* field to 0.
9. Read the CRC value from the *CRC_VAL.value* field.

21.6 Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 21-4: CRC Register Summary

Offset	Name	Description
[0x0000]	*CRC_CTRL*	CRC Control Register
[0x0004]	*CRC_DATAIN32*	CRC Data Input Register
[0x0008]	*CRC_POLY*	CRC Polynomial Register
[0x000C]	*CRC_VAL*	CRC Value Register

21.6.1 Register Details

Table 21-5: CRC Control Register

CRC Control			**CRC_CTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	busy	R	0	**CRC Busy** 0: Not busy 1: Busy
15:5	-	RO	0	**Reserved**
4	byte_swap_out	R/W	0	**Byte Swap CRC Value Output** 0: *CRC_VAL.value* is not byte swapped 1: *CRC_VAL.value* is byte swapped
3	byte_swap_in	R/W	0	**Byte Swap CRC Data Input** 0: *CRC_DATAIN32.data* is processed least significant byte first 1: *CRC_DATAIN32.data* is processed most significant byte first
2	msb	R/W	0	**Most Significant Bit Order** 0: LSBit data first 1: MSBit data first (mirrored)
1	dma_en	R/W	0	**DMA Enable** Set this field to 1 to enable a DMA request when the output FIFO is full. 0: Disabled 1: Enabled
0	en	R/W	0	**CRC Enable** 0: Disabled 1: Enabled

Table 21-6: CRC Data Input 32 Register

CRC Data In 32-bit			**CRC_DATAIN32**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	W	0	**CRC Data Input** This register can be written as a word, half-word, or byte. See *Table 21-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if* CRC_CTRL*.busy = 1 or* CRC_CTRL*.en = 0.*

Table 21-7: CRC Polynomial Register

CRC Polynomial			CRC_POLY	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	poly	R/W	0xEDB8 8320	**CRC Polynomial** See *Table 21-2* for details on the byte and bit ordering of the data in this register.

Table 21-8: CRC Value Register

CRC Value			CRC_VAL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	value	R/W	0	**Current CRC Value** The software can write to this register to set the initial state of the accelerator. This register should only be read or written when *CRC_CTRL.busy* = 0. See *Table 21-2* for details on the byte and bit ordering of the data in this register.

22. AES

The provided hardware AES accelerator performs calculations on blocks up to 128 bits.

The features include:

- Supports multiple key sizes:
  - 128 Bits
  - 192 Bits
  - 256 Bits
- DMA support for both receive and transmit channels
- Supports multiple key sources:
  - Encryption using the external AES key
  - Decryption using the external AES key
  - Decryption using the locally generated decryption key

22.1 Instances

Instances of the peripheral are listed in *Table 22-1*. Disable the peripheral by clearing *AESn_CTRL*.*en* = 0 before writing to any register field.

Table 22-1: MAX32655 AES Instances

Instance	128-Bit Key	192-Bit Key	256-Bit Key	DMA Support
AES0	Yes	Yes	Yes	Yes

22.2 Encryption of 128-Bit Blocks of Data Using FIFO

AES operations are typically performed on 128-bit of data at a time. The simplest use case is to have software encrypt 128-bit blocks of data. The *AESn_CTRL*.*start* field is unused in this case.

1. Generate key.
2. Wait for hardware to clear *AESn_STATUS*.*busy* = 0.
3. Clear *AESn_CTRL*.*en* = 0 to disable the peripheral.
4. If *AESn_STATUS*.*input_em* = 0, set *AESn_CTRL*.*input_flush* = 1 to flush the input FIFO.
5. If *AESn_STATUS*.*output_em* = 0, set *AESn_CTRL*.*output_flush* = 1 to flush the output FIFO.
6. Set *AESn_CTRL*.*key_size* to desired setting
7. Configure *AESn_CTRL*.*type* =00 (encryption with external key)
8. If interrupts are desired, set *AESn_INTEN*.*done* = 1 so that an interrupt will trigger at the end of the AES calculation.
9. Set *AESn_CTRL*.*en* = 1 to enable the peripheral.
10. Write 4×32-bit words of data to *AESn_FIFO*.*data*. The hardware starts the AES calculation.
11. If *AESn_INTEN*. *done* = 1, an interrupt triggers after the AES calculation is complete.
12. If *AESn_INTEN*.*done*= 0, the software should poll until *AESn_STATUS*.*busy* = 0.
13. Read 4 32-bit words from *AESn_FIFO*.*data* (least significant word first).
14. Repeat steps 9-13 until all 128-bit blocks have been processed.

22.3 Encryption of 128-Bit Blocks Using DMA

For this example, it is assumed that the DMA both reads and writes data to/from the AES. This is not a requirement. The AES could use DMA on one side and software on the other. It is required that for each DMA transmit request the DMA

writes 4×32-bit words of data into the AES. It is required that for each DMA receive request the DMA reads 4×32-bit words of data out of the AES.

The *AESn_CTRL*.*start* field is unused in this case. The state of *AESn_STATUS*.*busy* and *AESn_INTFL*.*done* are indeterminate during DMA operations. The software must clear *AESn_INTEN*.*done* = 0 when using the DMA mode. Use the appropriate DMA interrupt instead to determine when the DMA operation is complete and the results can be read from *AESn_FIFO*.data.

Assuming the DMA is continuously filling the data input FIFO, the calculations complete in the following number of SYS_CLK cycles:

- 128-Bit key: 181
- 192-Bit key: 213
- 256-Bit key: 245

The procedure to use DMA encryption/decryption is:

1. Generate key.
2. Initialize the AES receive and transmit channels for the DMA controller.
3. Wait for the hardware to clear *AESn_STATUS*.*busy* = 0.
4. Clear *AESn_CTRL*.*en* = 0 to disable the peripheral.
5. If *AESn_STATUS*.*input_em* = 0, set *AESn_CTRL*.*input_flush* = 1to flush the input FIFO.
6. If *AESn_STATUS*.*output_em* = 0, set *AESn_CTRL*.*output_flush* = 1 to flush the output FIFO.
7. Set *AESn_CTRL*.*key_size* to the desired setting.
8. Configure the AES encryption type to use an external key by setting the field *AESn_CTRL*.*type* to 0.
9. Ensure *AESn_INTEN*.*done* = 0 during DMA operations.
10. Set *AESn_CTRL*.*en* = 1 to enable the peripheral.
11. The DMA fills the FIFO and hardware begins the AES calculation.
12. When an AES calculation is completed, the AES hardware signals to the DMA that the data output FIFO is full and that it should be emptied. If the DMA does not empty the FIFO prior to the next calculation being complete, the hardware sets *AESn_STATUS*.*output_full* = 1.

Step 11 and step 12 are repeated if the DMA has new data to write to the data input FIFO.

Note that the interface from the DMA to the AES only works when the amount of data is a multiple of 128-bits.
For non-multiples of 128-bits, the remainder after calculating all of the 128-bit blocks needs to be encrypted manually.

22.4 Encryption of Blocks Less Than 128-Bits

The AES engine automatically starts a calculation when 128 bits (four writes of 32 bits) occurs. Operations of less than 128-bits use the start field to initiate the AES calculation.

1. Generate key.
2. Wait for the hardware to clear *AESn_STATUS.busy* = 0.
3. Clear *AESn_CTRL.en* = 0 to disable the peripheral.
4. If *AESn_STATUS.input_em* =0, set *AESn_CTRL.input_flush* = 1 to flush the input FIFO.
5. If *AESn_STATUS.output_em* =0, set *AESn_CTRL.output_flush* = 1 to flush the output FIFO.
6. Set *AESn_CTRL.key_size* to desired setting.
7. Configure *AESn_CTRL.type* =00 (encryption with external key).
8. If interrupts are desired, set *AESn_INTEN.done* = 1 so that an interrupt triggers at the end of the AES calculation.
9. Set *AESn_CTRL.en* = 1 to enable the peripheral.
10. Write from one to three 128 bits of data to *AESn_FIFO.data* (least significant word first).
11. Start the calculation manually by setting *AESn_CTRL.start* = 1.
12. If *AESn_INTEN.done* = 1, an interrupt triggers after the AES calculation is complete.
13. If *AESn_INTEN.done* = 0, the software should poll until *AESn_STATUS.busy* = 0.
14. Read 4 32-bit words from *AESn_FIFO.data* (least significant word first).

22.5 Decryption

Decryption of data is very similar to encryption. The only difference is in the setting of *AESn_CTRL.type*. There are two settings of this field for decryption:

- Decrypt with external key.
- Decrypt with internal decryption key.

The internal decryption key is generated during an encryption operation. It may be necessary to complete a dummy encryption prior to doing the first decryption to ensure that it is generated.

22.6 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 22-2*. Unless noted otherwise, each instance has its own independent set of interrupts, and higher-level flag and enable fields.

Multiple events may set an interrupt flag and generate an interrupt if the corresponding interrupt enable is set. The flags must be cleared by the dsoftware in the ISR.

Table 22-2: Interrupt Events

Event	Local Interrupt Flag	Local Interrupt Enable
Data Output FIFO Overrun	*AESn_INTFL*.ov	*AESn_INTEN.ov*
Key Zero	*AESn_INTFL.key_zero*	*AESn_INTEN.key_zero*
Key Change	*AESn_INTFL.key_change*	*AESn_INTEN.key_change*
Calculation Done	*AESn_INTFL.done*	*AESn_INTEN.done*

22.6.1 Data Output FIFO Overrun

When an AES calculation is completed, the AES will signal to the DMA that the Data Output FIFO is full and that it should be emptied. If the DMA does not empty the FIFO prior to the next calculation being complete, a data output FIFO overrun event occurs and the corresponding local interrupt flag is set.

22.6.2 Key Zero

Attempting a calculation with a key of all zeros generates a key zero event.

22.6.3 Key Change

Writing to any key register while *AESn_STATUS*.*busy* = 1 generates a key change event.

22.6.4 Calculation Done

The transition of *AESn_STATUS*.*busy* = 1 to *AESn_STATUS*.*busy* = 0 generates a calculation done event. The calculation done event interrupt must be disabled when using the DMA.

22.7 Registers

See *Table 2-4* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 22-3*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 22-3: AES Register Summary

Offset	Name	Description
[0x0000]	*AESn_CTRL*	AES Control Register
[0x0004]	*AESn_STATUS*	AES Status Register
[0x0008]	*AESn_INTFL*	AES Interrupt Flag Register
[0x000C]	*AESn_INTEN*	AES Interrupt Enable Register
[0x0010]	*AESn_FIFO*	AES Data FIFO

22.7.1 Register Details

Table 22-4: AES Control Register

AES Control			AESn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9:8	type	R/W	0	**Encryption Type** 00: Encryption using the external AES key 01: Decryption using the external AES key 10: Decryption using the locally generated decryption key 11: Reserved
7:6	key_size	R/W	0	**Encryption Key Size** 00: 128 Bits 01: 192 Bits 10: 256 Bits 11: Reserved
5	output_flush	W1	0	**Flush Data Output FIFO** This field will always read 0. 0: No Action 1: Flush

AES Control			AESn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
4	input_flush	W1	0	**Flush Data Input FIFO** This field always reads 0. 0: No Action 1: Flush
3	start	W1	0	**Start AES Calculation** This field forces the start of an AES calculation regardless of the state of the data input FIFO. This allows doing an AES calculation on less than 128-bits of data since normally an AES calculation starts when the data input FIFO is full. This field always reads 0. 0: No Action 1: Start calculation
2	dma_tx_en	R/W	0	**DMA Request To Write Data Input FIFO** 0: Disabled 1: Enabled. DMA request is generated if the data input FIFO is empty.
1	dma_rx_en	R/W	0	**DMA Request To Read Data Output FIFO** 0: Disabled 1: Enabled. DMA request is generated if the data output FIFO is full.
0	en	R/W	0	**AES Enable** 0: Disabled 1: Enabled.

Table 22-5: AES Status Register

AES Status			AESn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	output_full	R	0	**Output FIFO Full** 0: Not full 1: Full
3	output_em	R	0	**Output FIFO Empty** 0: Not empty 1: Empty
2	input_full	R	0	**Input FIFO Full** 0: Not full 1: Full
1	input_em	R	0	**Input FIFO Empty** 0: Not empty 1: Empty
0	busy	R	0	**AES Busy** 0: Not busy 1: Busy

Table 22-6: AES Interrupt Flag Register

AES Interrupt Flag			AESn_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ov	W1C	0	**Data Output FIFO Overrun Event Interrupt** 0: No event 1: Event occurred
2	key_zero	W1C	0	**Key Zero Event Interrupt** 0: No event 1: Event occurred

AES Interrupt Flag			AESn_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
1	key_change	W1C	0	**Key Change Event Interrupt** 0: No event 1: Event occurred
0	done	W1C	0	**Calculation Done Event Interrupt** 0: No event 1: Event occurred

Table 22-7: AES Interrupt Enable Register

AES Interrupt Enable			AESn_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ov	W1C	0	**Data Output FIFO Overrun Event Interrupt Enable** 0: Enabled 1: Disabled
2	key_zero	W1C	0	**Key Zero Event Interrupt Enable** 0: Enabled 1: Disabled
1	key_change	W1C	0	**Key Change Event Interrupt Enable** 0: Enabled 1: Disabled
0	done	W1C	0	**Calculation Done Event Interrupt Enable** This event interrupt must be disabled when using the DMA. 0: Enabled 1: Disabled

Table 22-8: AES FIFO Register

AES Data			AESn_FIFO	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**AES FIFO** Writing this register puts data to the data input FIFO. The hardware automatically starts a calculation after four words are written to this FIFO. The data should be written with the least significant word first. Reading this register pulls data from the data output FIFO. The least significant word is read first.

23. TRNG Engine

The Maxim-supplied Universal Cryptographic Library (UCL) provides a function to generate random numbers intended to meet the requirements of commonly security validations. The entropy from one or more internal noise sources continually feeds a TRNG, the output of which is then processed by software and hardware to generate the number returned by the UCL function. Maxim works directly with the customer's accredited testing laboratory to provide any information regarding the TRNG needed to support the customer's validation requirements.

The general information in this section is provided only for completeness; customers are expected to use the Maxim UCL for the generation of random number.

The TRNG engine can also be used to generate AES keys by software using a Hardware Key Derivation Function (HKDF) and using the TRNG as input to the HKDF.

23.1 TRNG Registers

See *Table 2-4* for the base address of this peripheral/module. See *Table 2-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 23-1: Register Summary

Offset	Register	Name
[0x0000]	*TRNG_CTRL*	TRNG Control Register
[0x0004]	*TRNG_STATUS*	TRNG Status Register
[0x0008]	*TRNG_DATA*	TRNG Data Register

23.1.1 Register Details

Table 23-2: TRNG Control Register

Cryptographic Control			TRNG_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	keywipe	R/W	0	**Wipe Key** Write this field to 1 to wipe the TRNG key.
14:4	-	RO	0	**Reserved**
3	keygen	R/W	0	**Generate Key** Write this field to 1 to generate a key using the TRNG.
2	-	RO	0	**Reserved**
1	rnd_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_STATUS.rdy* = 1. 0: Disabled 1: Enabled
0	-	RO	0	**Reserved**

Table 23-3: TRNG Status Register

TRNG Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

TRNG Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
0	rdy	R	0	**Random Number Ready** This bit is automatically cleared to 0 and a new random number is generated when *TRNG_DATA*.*data* is read. 0: Random number generation in progress. The content of *TRNG_DATA*.*data* is invalid. 1: *TRNG_DATA*.*data* contains new 32-bit random data. An interrupt is generated if *TRNG_CTRL*.*rnd_ie* = 1

Table 23-4: TRNG Data Register

TRNG Data			TRNG_DATA	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	RO	0	**TRNG Data** The 32-bit random number generated is available here when *TRNG_STATUS*.*rdy* = 1.

24. Bluetooth 5 Low Energy (LE) Radio

Bluetooth 5 Low Energy (LE) radio is the latest version of the Bluetooth wireless communication standard. It is used for wireless headphones and other audio hardware, as well as for communication between various smart home and Internet of Things (IoT) devices.

Devices operate in the unlicensed 2.4GHz ISM (Industrial, Scientific, Medical) band. A frequency-hopping transceiver is used to combat interference and fading. The system operates in the 2.4GHz ISM band at 2400MHz to 2483.5MHz. It uses 40 RF channels. These RF channels have center frequencies of 2402 + k × 2MHz, where k = 0, …, 39.

Bluetooth 5 technology provides:

- 1Mbps, 2Mbps, and Long Range coded (125kbps and 500kbps) data rates
- Increased broadcast capability
- Advertising packet up to 255 bytes
- On-chip matching network to the antenna
- Hardware on-the-fly encryption and decryption for lower power consumption
- Supports mesh networking
- Supports high-quality audio streaming (isochronous)
- Low-power proprietary mode that supports 20kbps, 40kbps, 500kbps-MSK/GFSK, 1Mpbs-GFSK

24.1 Power-Efficient Design

The provided Bluetooth Low Energy radio is optimized for low-power operation.

- Higher transmit power up to +9.5dbm
- Low transmit current of 2.5mA at 0dbm at 3.3V
- Low receive current of 1.5mA at 3.3V

24.2 Bluetooth Hardware Accelerator

The dedicated Bluetooth hardware accelerator eliminates the need for application software to accommodate the strict timing requirements and restrictions. It transparently increases system performance while reducing overall power consumption of the Bluetooth system.

24.3 Arm Cordio®-B50 Software Stack

Maxim provides the Arm Cordio®-B50 stack in library form. This provides application developers access to Bluetooth without validation and development of a software stack.

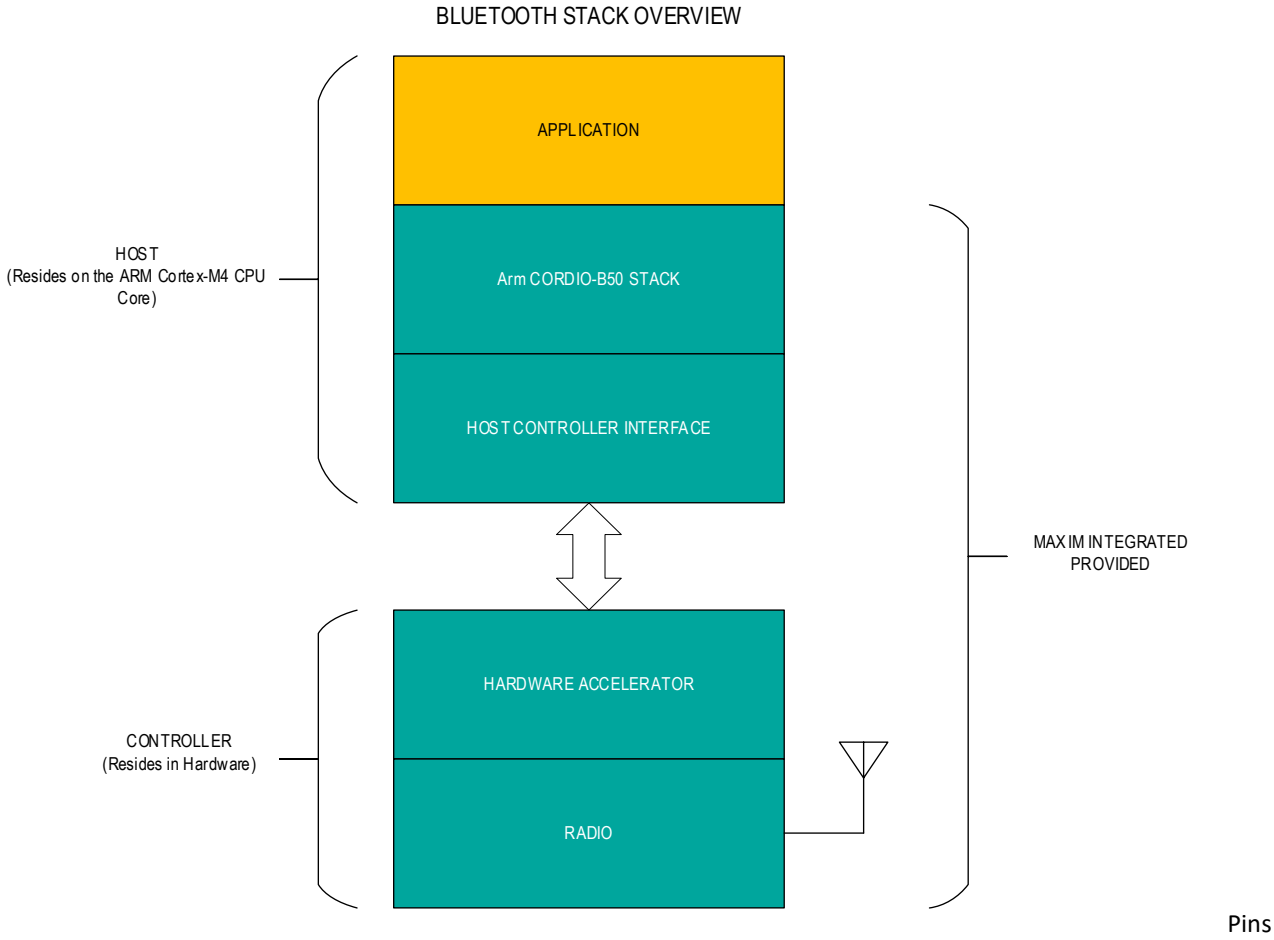
The Arm Cordio-B50 software stack interfaces to the Bluetooth link layer running on dedicated hardware. The dedicated hardware for the stack enables the ultimate in power management for IoT applications.

Cordio is a registered trademark of Arm Limited.

Arm Cordio-B50 features the following:

- C library for linking directly into an application development tool
- Host selects the PHY it needs to use at any given time enabling long range or higher bandwidth only when required
- LE 1M
- LE Coded S = 2
- LE Coded S = 8
- LE 2M
- Bluetooth 5 advertising extension support for enabling next generation Bluetooth beacons
- Larger packets and advertising channel offloading
- Packets up to 255 octets long
- Advertising packet chaining
- Advertising sets
- Periodic advertising
- High-duty cycle non-connectable advertising
- Sample applications using standard profiles built on the Arm Cordio-B50 software framework

Figure 24-1: MAX32655 Bluetooth Stack Overview



Pins

The interface has two device pins to connect to the off-chip user-provided antenna. The ANT device pin is the radio frequency signal pin and it should be routed through the ANT THRU device pin to the antenna. The ANT device pin is a 50Ω source impedance driver.

24.4 Configuration

The Radio and Hardware Accelerator requires a 32MHz external crystal specified in the data sheet. The CPU core hosting the Arm Cordio-B50 stack must run at a core speed greater than 32MHz.

Perform the following steps to enable the Bluetooth radio:

1. Set *GCR_BTLELDOCTRL.ldowen* and set to *GCR_BTLELDOCTRL.ldowoen* 1 to enable the internal Bluetooth LDO.
2. Set *GCR_CLKCTRL.x32M_en* to 1.
3. Wait for *GCR_CLKCTRL.x32M_rdy* to be set to 1. Do not access any Bluetooth registers beforehand.

24.4.1 Kick Starting the ERFO

24.5 Documentation

The Arm Codio-B50 Stack Product Sheet and Profiles are available online at:

https://www.arm.com/products/silicon-ip-wireless/wpan-software

The Maxim MAX32655 Low-Power Arm Micro Toolchain for both iOS and Windows is available online at:

https://www.maximintegrated.com/en/products/microcontrollers/MAX32655.html/tb_tab2

This toolchain includes documentation of the Arm Cordio-B50 Stack and profile examples.

25. Secure Boot

The device provides a secure boot feature, which employs a digital signature feature to guarantee the integrity and authenticity of program memory before any execution of user software.

The provided debug access port allows programming and debug capability through the JTAG interface.

All versions of the bootloader provide the ability to permanently block read/write access to program memory.

Bootloader features:

- Application image digital signature guarantees integrity and authenticates program code.
- Manufacturer and customer root key management.
- Debugger and loading done through JTAG interface or SWD interface.
- Trusted secure boot provides automatic program memory verification and authentication before execution after every reset.

Root of Trust Starts with trusted software and the DeepCover® secure microcontroller hardware. The secure boot software stored in the microcontroller ROM is considered inherently trusted (i.e., the root of trust) because it cannot be modified. If this early software can be modified without control, trust cannot be guaranteed. "Early" means it is the first piece of software executed when the microcontroller is powered on. Hence, the requirement for inherent trustworthiness of this initial software. If this software is trustworthy, then it can be used for verifying the signature of the application before relinquishing the control of the microcontroller.

At power-on, the device's microcontroller starts running the root-of-trust code from ROM. This code's primary task is to start the application code after successful verification of its signature. Verification of the signature is done using a public key previously loaded in the microcontroller using the device-specific method.

These protected microcontrollers can still be used in their usual manner by developers for writing software, loading, and executing the software through the JTAG, and debugging.

25.1 Development Tools

The information in this chapter is provided for completeness for customers who wish to understand the secure boot process. However, Maxim Integrated provides a complete software development kit that builds a complete application image. It is highly recommended that customers contact Maxim Integrated to get the latest productivity enhancing tools.

The dedicated pins and features of the bootloader are shown in *Table 25-1*.

Table 25-1: MAX32655 Bootloader Instances

Part Number	Activation Pins	Digital Signature Algorithm	Secure Boot
MAX32655GXG+	There are no dedicated bootloader activation pins. The secure boot is configured directly through the JTAG interface. RV_TCK: P1.0 RV_TMS: P1.1 RV_TDI: P1.2 RV_TDO: P1.3	ECDSA-256	Yes

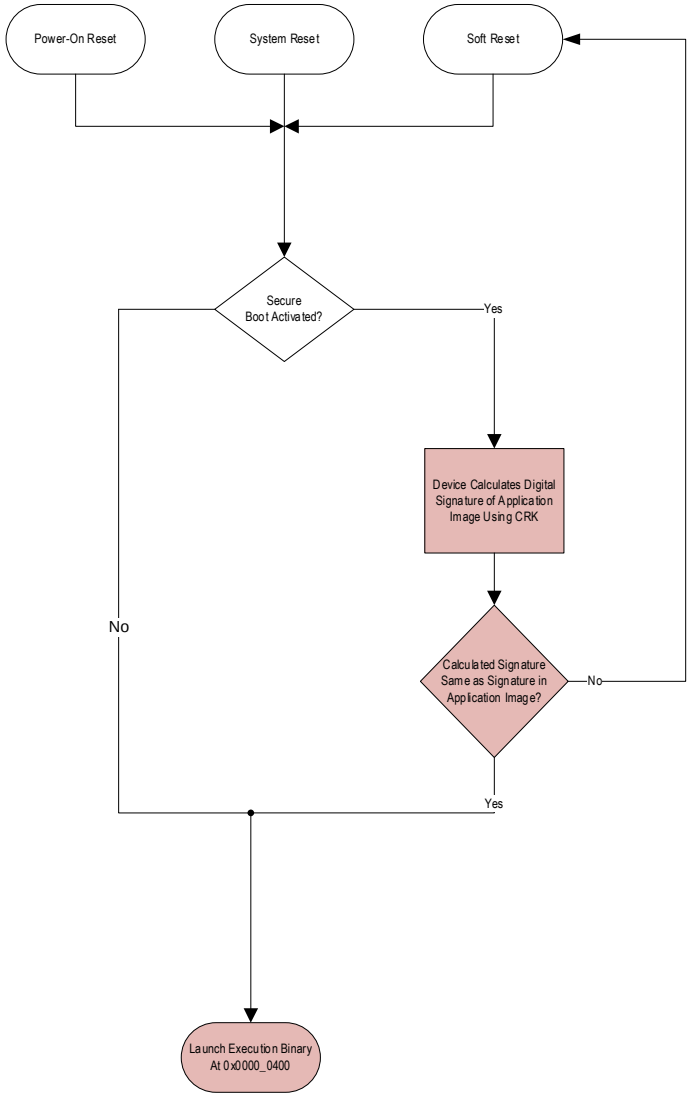
25.2 Secure Boot

Following a reset, the device executes a secure boot from ROM to verify the integrity of the code in flash memory. This establishes the chain of trust that application execution relies on; the secure boot guarantees to the applications the platform they are running on is authorized and trustworthy. The secure boot also guarantees that the applications running on the platform are only the authorized ones, so only trusted applications can run on the platform.

Before executing program code, the secure ROM calculates the digital signature of the application image in flash memory using the previously loaded CRK. If the calculated value matches the digital signature stored at the end of the application image, the device exits the bootloader and begins execution of the user code.

If the calculated value does not match the digital signature, either by an error in creation of the application image or corruption of flash memory, the device enters the shutdown mode (remain looping in the reset state) until a new SCP packet is sent that erases program memory.

Figure 25-1: MAX32655 Secure Boot Flow

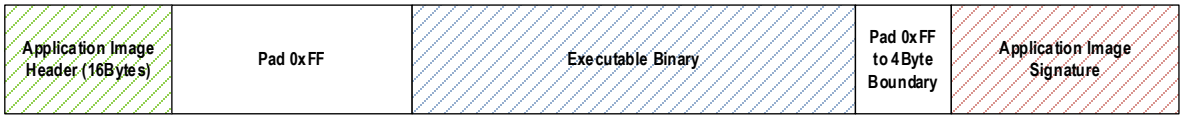


25.3 Building the Application Image

The application image is the data that gets physically programmed into program memory, regardless of the transmission protocol.

The application image has a specific format shown in *Figure 25-2*.

Figure 25-2: MAX32655 Application Image



The application image contains multiple parts:

- Application image header, containing information about length of the execution binary and identification of the cryptographic protection method.
- Executable binary, which is the compiled customer code.
- A digital signature or checksum to verify the integrity and/or authenticate the application image header and the executable binary.
- Padding as needed to the ensure the start and end of the executable binary align with the required boundaries.

A detailed breakdown of the application image is detailed in *Table 25-2*

Table 25-2: MAX32655 Application Image Structure

Element	Absolute Start Address within Application Image	Length (Range)	Value
*Synchronization pattern	0x0000 0000	16 bytes	0x516A000100000001461A8CF5BF1421E4
Digital Signature Address	0x0000 0010	1 byte	Located at the end of the executable binary
*Padding	0x0000 0014	(0x0000_0014 - 0x0000_03FF)	0x0xFF (fixed)
*Executable Binary Start Address	0x0000 0400	Var	0x0000_0400 (fixed)
Terminal padding to 4 Byte boundary, if necessary	End of executable binary	0 1B 2B #B	N/A 0xFF 0xFFFF 0xFFFFFF
*Application Image Signature	End of executable binary + terminal padding	256 bits	ECDSA256 signature

* The customer must ensure the start of the executable binary is at address 0x0000 0400.

25.4 Root Key Management

The secure boot performs authentication using the encryption type shown in *Table 25-1*. The keys are stored in non-volatile memory accessible to the secure boot ROM.

The secure boot PKI is based on a Maxim Integrated Root Authority. The manufacturer root key (MRK) is stored in the ROM. The chips are delivered to customers without any customer key.

Once the software is finalized and fully tested, and then audited and approved by a certification laboratory or an internal validation authority, it must be released. Releasing software for a secure boot requires one additional, crucial step: the binary executable code signature. Signing the code, in fact, "seals" the code (it cannot be further modified without those modifications being detected) and authenticates it (the identity of the approver is well established). The code is sealed because, if modified, the associated signature becomes invalid – the integrity of the digital signature is no longer complete. The code is also authenticated because it was signed by a unique, undisclosed, private key by its owner.

The MAX32655 implements a simpler personalization method that requires only the CRK stored in internal, non-volatile memory. A trusted environment is needed to ensure that the intended public key is not replaced by a rogue key because the root of trust cannot verify this key. This key also must be internally protected to ensure there is no integrity issue with that key.

25.4.1 Manufacturer Root Key (MRK)

The MRK pair is owned by Maxim Integrated. It is not used on this device.

25.4.2 Customer Root Key (CRK)

Devices are delivered to customers without any customer key. Customers generate their own CRK keypair using (typically) PCI PTS compliant tools. The public key, used for the digital signature, must be signed by Maxim Integrated. The signing procedure is described in the *CRK Certification* section.

The customer must protect the secret part of its RSA key pair. Maxim Integrated cannot guarantee the chain of trust if this key is compromised. Maxim Integrated recommends the usage of a Host Security Module (HSM) or an equivalent hardware device providing physical protection to the secret part of the RSA key pair and complying with usual (PCI-PTS, FIPS 140-2) keys protection requirements.

25.4.3 CRK Certification

The exchange of the unsigned CRK from the customer, and the return of the signed key must be done securely in procedure known as the PGP key ceremony:

1. The designated customer contact person sends his PGP key by email to his Maxim Integrated Business Manager and confirms it through another channel (skype, phone call).
2. The Maxim Integrated Business Manager sends by email the crk.certificate PGP public key.
3. The customer sends the CRK public key value to Maxim Integrated by email:
    a. The CRK is inserted in the body email.
    b. The CRK is in hexadecimal format; the number 8903214_{10} is coded as $87DA2E_{16}$.
    c. The email address where to send the public key is *crk.certificate@maximintegrated.com*
    d. The email shall explicitly contain the customer name and device part number.
    e. The email shall be signed by the sender, using PGP.
4. Maxim Integrated sends an email signed by the crk.certificate PGP key to the designated customer contact. This email has an attached zip file containing the secure boot initialization script and the secure activation script.

25.5 Program Loading

Programming is done through the device JTAG port, usually using a scripting program such as GDB. Once the binary executable is compiled, and the application image is built, the CRK must first be loaded onto the device.

Maxim Integrated provides a Secure Boot Initialization script and a Secure Boot Activation script to be loaded immediately before and after, respectively, the application Image. The procedure used is:

1. Activate JTAG interface.
2. Execute the Secure Boot Initialization script.
3. Load the application image.
4. Execute the Secure Boot Activation script.
5. Reset the device.

The secure boot feature is now enabled, and the device now performs a secure boot following all resets.

26. Revision History

Table 26-1: Revision History

REVISON NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	03/2021	Initial Release	-