# EZ-PD™ PMG1-S3 MCU architecture

**Technical reference manual**

## About this document

### Scope and purpose

EZ-PD™ Power Delivery Microcontroller Gen1 (PMG1) is a family of high-voltage USB-C Power Delivery (PD) microcontrollers (MCU). These chips include an Arm® Cortex®-M0/M0+ CPU and USB-C PD controller along with analog and digital peripherals. EZ-PD™ PMG1-S3 MCU is targeted for any embedded system that provides/consumes power to/from a high-voltage USB-C PD port and leverages the microcontroller to provide additional control capability.

### Intended audience

The document is intended for users of EZ-PD™ PMG1-S3 MCU.

# Table of contents

**About this document** 1

**Scope and purpose** 1

**Intended audience** 1

**Table of contents** 2

**Section A: Overview** 12

**1 Introduction** . . . 13
1.1 Top level architecture . . . 17
1.2 Features . . . 17
1.3 CPU system . . . 18
1.3.1 Processor . . . 18
1.3.2 Interrupt controller . . . 18
1.3.3 Direct memory access . . . 19
1.3.4 Memory . . . 19
1.3.5 Crypto . . . 19
1.4 System-wide resources . . . 19
1.4.1 Clocking system . . . 19
1.4.2 Power system . . . 19
1.4.3 Watchdog timers . . . 19
1.4.4 Voltage reference . . . 20
1.5 USB PD subsystem . . . 20
1.5.1 USB PD physical layer . . . 20
1.5.2 VCONN FET . . . 20
1.5.3 8-bit SAR ADC . . . 20
1.5.4 SBU pass-through switch and USB HS mux . . . 20
1.5.5 Under-voltage and over-voltage protection on VBUS . . . 20
1.5.6 Integrated load switch controller for provider path . . . 21
1.5.7 High-side current sense amplifier for VBUS . . . 21
1.5.8 VBUS discharge . . . 21
1.5.9 VBUS regulator . . . 21
1.5.10 NFET gate driver for VBUS . . . 21
1.6 Fixed-function digital . . . 21
1.6.1 Timer/Counter/PWM block . . . 21
1.6.2 Serial Communication Block (SCB) . . . 22
1.6.3 GPIO . . . 22
1.7 Analog systems . . . 22
1.7.1 12-bit SAR ADC . . . 22
1.7.2 Continuous time block mini . . . 22
1.7.3 Temperature sensor . . . 22
1.7.4 Low-power comparators . . . 23
1.8 Special function peripherals . . . 23
1.8.1 CAPSENSE™ . . . 23
1.8.2 USB 2.0 Full Speed device and charger detection . . . 23
1.9 Program and debug . . . 23

**2 Getting started** . . . 24
2.1 EZ-PD™ PMG1 MCU resources . . . 24
2.2 Product upgrades . . . 24

## Table of contents


**3 Document construction ..... 25**
3.1 Major sections ..... 25
3.2 Documentation conventions ..... 25
3.2.1 Register conventions ..... 25
3.2.2 Numeric naming ..... 25
3.2.3 Units of measure ..... 26
3.2.4 Acronyms ..... 27

**Section B: CPU system 31**

**4 Cortex®-M0+ CPU ..... 32**
4.1 Features ..... 32
4.2 Block diagram ..... 33
4.3 How it works ..... 33
4.3.1 Registers ..... 33
4.3.2 Operating modes ..... 35
4.3.3 Instruction set ..... 35
4.3.3.1 Address alignment ..... 37
4.3.3.2 Memory endianness ..... 37
4.3.4 Systick timer ..... 37
4.3.5 Debug ..... 37

**5 Interrupts ..... 38**
5.1 Features ..... 38
5.2 How it works ..... 38
5.3 Interrupts and exceptions – operation ..... 39
5.3.1 Interrupt and exception handling ..... 39
5.3.2 Level and pulse interrupts ..... 40
5.3.3 Exception vector table ..... 40
5.4 Exception sources ..... 41
5.4.1 Reset exception ..... 41
5.4.2 Non-maskable interrupt (NMI) exception ..... 42
5.4.3 HardFault exception ..... 42
5.4.4 Supervisor call (SVCall) exception ..... 42
5.4.5 PendSV exception ..... 42
5.4.6 SysTick exception ..... 43
5.5 Interrupt sources ..... 43
5.6 Exception priority ..... 44
5.7 Enabling/disabling interrupts ..... 45
5.8 Exception states ..... 45
5.8.1 Pending exceptions ..... 46
5.9 Stack usage for exceptions ..... 47
5.10 Interrupts and low-power modes ..... 47
5.11 Exception – initialization and configuration ..... 47
5.12 Registers ..... 48
5.13 Associated documents ..... 48

**6 DMA controller modes ..... 49**
6.1 Block diagram description ..... 50
6.1.1 Trigger sources and multiplexing ..... 51
6.1.2 Pending triggers ..... 51
6.1.3 Output triggers ..... 51

**Table of contents**


6.1.4 Channel prioritization ... 51
6.1.5 Data transfer engine ... 51
6.2 Descriptors ... 52
6.2.1 Address configuration ... 52
6.2.2 Transfer size ... 54
6.2.3 Descriptor chaining ... 55
6.2.4 Transfer mode ... 55
6.2.4.1 Single data element per trigger (OPCODE 0) ... 55
6.2.4.2 Entire descriptor per trigger (OPCODE 1) ... 58
6.2.4.3 Entire descriptor chain per trigger (OPCODE 2) ... 59
6.3 Operation and timing ... 61
6.4 Arbitration ... 62
6.5 Registers ... 64

**7 Cryptography ... 65**
7.1 Block diagram ... 66
7.2 Functional description ... 66
7.2.1 AES-128 ... 66
7.2.2 SHA-256 ... 68
7.2.3 Vector unit ... 69
7.2.4 True random number generator ... 71
7.3 Interrupts ... 72
7.4 Registers ... 72

**Section C: Memory system 73**

**8 Memory map ... 74**
8.1 Features ... 74
8.2 How it works ... 74

**Section D: System-wide resources 76**

**9 I/O system ... 77**
9.1 Features ... 77
9.2 Block diagram ... 77
9.3 GPIO drive modes ... 78
9.3.1 High-impedance analog ... 79
9.3.2 High-impedance digital ... 79
9.3.3 Resistive pull-up or resistive pull-down ... 79
9.3.4 Open drain drives high and open drain drives low ... 80
9.3.5 Strong drive ... 80
9.3.6 Resistive pull-up and resistive pull-down ... 80
9.4 Slew rate control ... 80
9.5 CMOS LVTTL level control ... 80
9.6 GPIO-OVT ... 80
9.7 High-speed I/O matrix ... 81
9.8 Firmware controlled GPIO ... 81
9.9 I/O port reconfiguration ... 81
9.10 I/O state on power up ... 82
9.11 Behavior in low-power modes ... 82
9.12 GPIO interrupt ... 82
9.12.1 Features ... 82
9.12.2 Interrupt controller block diagram ... 83

## Table of contents


9.12.3 Function and configuration ... 83
9.13 Registers ... 84

**10 Clocking system ... 85**
10.1 Block diagram ... 85
10.2 Clock sources ... 86
10.2.1 Internal main oscillator ... 86
10.2.1.1 Startup behavior ... 86
10.2.2 Internal low-speed oscillator ... 86
10.2.3 External clock ... 86
10.3 Clock distribution ... 87
10.3.1 HFCLK input selection ... 87
10.3.2 HFCLK predivider configuration ... 87
10.3.3 SYSCLK prescaler configuration ... 88
10.3.4 Peripheral clock divider configuration ... 88
10.4 Low-power mode operation ... 89
10.5 Registers ... 90

**11 Power supply and monitoring ... 91**
11.1 Block diagram ... 91
11.2 How it works ... 91
11.2.1 Regulator summary ... 92
11.2.1.1 Active digital regulator ... 92
11.2.1.2 Quiet regulator ... 92
11.2.1.3 Deep-Sleep regulator ... 92
11.3 Voltage monitoring ... 92
11.3.1 Power-on-reset ... 92
11.3.1.1 Brownout-detect ... 92
11.4 Registers ... 93

**12 Chip operational modes ... 94**
12.1 Boot ... 94
12.2 User ... 94
12.3 Privileged ... 94
12.4 Debug ... 94

**13 Power modes ... 95**
13.1 Active mode ... 96
13.2 Sleep mode ... 96
13.3 Deep-Sleep mode ... 97
13.4 Power mode summary ... 97
13.5 Low-power mode entry and exit ... 98
13.6 Registers ... 98

**14 Watchdog timer ... 99**
14.1 Features ... 99
14.2 Block diagram ... 99
14.3 How it works ... 99
14.3.1 Enabling and disabling WDT ... 100
14.3.2 WDT interrupts and low-power modes ... 100
14.3.3 WDT reset mode ... 100
14.4 Registers ... 101

**15 Reset system ... 102**

## Table of contents


15.1 Reset sources . . . 102
15.1.1 Power-on reset . . . 102
15.1.2 Brownout reset . . . 102
15.1.3 Watchdog reset . . . 102
15.1.4 Software initiated reset . . . 102
15.1.5 External reset . . . 103
15.1.6 Protection fault reset . . . 103
15.2 Identifying reset sources . . . 103
15.3 Registers . . . 103

**16 Device security . . . 104**
16.1 Features . . . 104
16.2 How it works . . . 104

**17 Trigger multiplexer block . . . 105**
17.1 Features . . . 105
17.2 Architecture . . . 105
17.2.1 Trigger multiplexer group . . . 106
17.2.2 Software triggers . . . 106
17.3 Trigger sources and destinations . . . 107
17.4 Registers . . . 110

**Section E: Digital system 111**

**18 Serial Communications Block (SCB) . . . 112**
18.1 Features . . . 112
18.2 Serial Peripheral Interface (SPI) . . . 112
18.2.1 Features . . . 112
18.2.2 General description . . . 113
18.2.3 SPI modes of operation . . . 114
18.2.3.1 Motorola SPI . . . 114
18.2.3.2 Texas Instruments SPI . . . 117
18.2.3.3 National Semiconductors SPI . . . 119
18.2.4 Easy SPI (EZSPI) protocol . . . 120
18.2.4.1 EZ address write . . . 120
18.2.4.2 Memory array write . . . 120
18.2.4.3 Memory array read . . . 121
18.2.4.4 Configuring SCB for EZSPI mode . . . 122
18.2.5 SPI registers . . . 122
18.2.6 SPI interrupts . . . 123
18.2.7 Enabling and initializing SPI . . . 123
18.2.8 Internally and externally clocked SPI operations . . . 125
18.2.8.1 Non-EZ mode of operation . . . 126
18.2.8.2 EZ Mode of operation . . . 127
18.3 UART . . . 128
18.3.1 Features . . . 128
18.3.2 General description . . . 128
18.3.3 UART modes of operation . . . 129
18.3.3.1 Standard protocol . . . 129
18.3.3.2 SmartCard (ISO 7816) . . . 133
18.3.3.3 IrDA . . . 134
18.3.4 UART registers . . . 135

**Table of contents**


18.3.5 UART interrupts . . . 135
18.3.6 Enabling and initializing UART . . . 135
18.4 Inter integrated circuit (I2C) . . . 137
18.4.1 Features . . . 137
18.4.2 General description . . . 137
18.4.3 Terms and definitions . . . 138
18.4.4 I2C modes of operation . . . 139
18.4.4.1 Write transfer . . . 139
18.4.4.2 Read transfer . . . 140
18.4.5 EZI2C protocol . . . 141
18.4.6 Memory array write . . . 141
18.4.7 Memory array read . . . 141
18.4.8 I2C registers . . . 142
18.4.9 I2C interrupts . . . 143
18.4.10 Enabling and initializing the I2C . . . 144
18.4.11 Internal and external clock operation in I2C . . . 145
18.4.11.1 I2C non-EZ operation mode . . . 145
18.4.11.2 EZ operation mode . . . 146
18.4.12 Wake up from Sleep . . . 146
18.4.13 Master mode transfer examples . . . 147
18.4.13.1 Master transmit . . . 147
18.4.13.2 Master receive . . . 148
18.4.14 Slave mode transfer examples . . . 149
18.4.14.1 Slave transmit . . . 149
18.4.14.2 Slave receive . . . 150
18.4.15 EZ slave mode transfer example . . . 151
18.4.15.1 EZ slave transmit . . . 151
18.4.15.2 EZ slave receive . . . 152
18.4.16 Multi-master mode transfer example . . . 153
18.4.16.1 Multi-master – Slave not enabled . . . 153
18.4.16.2 Multi-master – Slave enabled . . . 154

**19 Timer, Counter, and PWM . . . 155**
19.1 Features . . . 155
19.2 Block diagram . . . 156
19.2.1 Enabling and disabling counter in TCPWM block . . . 156
19.2.2 Clocking . . . 157
19.2.3 Events based on trigger inputs . . . 157
19.2.4 Output signals . . . 158
19.2.4.1 Signals upon trigger conditions . . . 159
19.2.4.2 Interrupts . . . 159
19.2.4.3 Outputs . . . 160
19.2.5 Power modes . . . 160
19.3 Modes of operation . . . 161
19.3.1 Timer mode . . . 162
19.3.1.1 Block diagram . . . 162
19.3.1.2 How it works . . . 162
19.3.1.3 Configuring counter for timer mode . . . 164
19.3.2 Capture mode . . . 164
19.3.2.1 Block diagram . . . 164

**Table of contents**

19.3.2.2 How it works . . . 165
19.3.2.3 Configuring counter for capture mode . . . 166
19.3.3 Quadrature decoder mode . . . 166
19.3.3.1 Block diagram . . . 166
19.3.3.2 How it works . . . 167
19.3.3.3 Configuring counter for quadrature mode . . . 169
19.3.4 PWM mode . . . 169
19.3.4.1 Block diagram . . . 169
19.3.4.2 How it works . . . 170
19.3.4.3 Other configurations . . . 171
19.3.4.4 Kill feature . . . 172
19.3.4.5 Configuring counter for PWM mode . . . 172
19.3.5 Pulse Width Modulation with dead time mode . . . 173
19.3.5.1 Block diagram . . . 173
19.3.5.2 How it works . . . 173
19.3.5.3 Configuring counter for PWM with dead time mode . . . 174
19.3.6 Pulse width modulation pseudo-random mode . . . 175
19.3.6.1 Block diagram . . . 175
19.3.6.2 How it works . . . 175
19.3.6.3 Configuring counter for pseudo-random PWM mode . . . 176
19.4 Registers . . . 177

**Section F: USB power and data 178**

**20 USB Power Delivery (USB PD) . . . 179**
20.1 Features . . . 179

**21 USB Full Speed (USB FS) . . . 180**
21.1 Features . . . 180
21.2 Block diagram . . . 181
21.2.1 USB physical layer (USB PHY) . . . 181
21.2.2 Serial interface engine (SIE) . . . 181
21.2.3 Arbiter . . . 182
21.2.3.1 SIE interface module . . . 182
21.2.3.2 CPU/DMA interface block . . . 182
21.2.3.3 Memory interface . . . 182
21.2.3.4 Arbiter logic . . . 182
21.3 How it works . . . 183
21.3.1 USB physical layer (USB PHY) . . . 183
21.3.1.1 Power scheme . . . 183
21.3.1.2 VBUS detection . . . 184
21.3.1.3 USB PHY isolation logic . . . 185
21.3.1.4 USB D+ pin pull-up enable logic . . . 185
21.3.1.5 Transmitter and receiver logic . . . 185
21.3.1.6 Battery charger detection (BCD) . . . 186
21.3.1.7 Link power management (LPM) . . . 187
21.3.2 Endpoints . . . 187
21.3.3 Transfer types . . . 188
21.3.4 Interrupts . . . 188
21.3.4.1 Data endpoint interrupt events . . . 188
21.3.4.2 Control endpoint (EP0) interrupt event . . . 188
21.3.4.3 Arbiter interrupt event . . . 188

**Table of contents**

21.3.4.4 USB start of frame (SOF) event . . . 190
21.3.4.5 USB bus reset event . . . 190
21.3.4.6 Link power management (LPM) event . . . 190
21.3.5 DMA support . . . 191
21.4 Logical transfer modes . . . 191
21.4.1 Store and forward mode . . . 194
21.4.1.1 No DMA access . . . 194
21.4.1.2 Manual DMA access . . . 196
21.4.2 Cut through mode . . . 198
21.4.3 Control endpoint logical transfer . . . 201
21.5 Registers . . . 203

**Section G: Analog system 206**

**22 12-bit SAR ADC . . . 207**
22.1 Features . . . 207
22.2 Analog Multiplexed Bus (AMUX BUS) . . . 208
22.3 Block diagram . . . 209
22.4 How it works . . . 209
22.4.1 SAR ADC core . . . 209
22.4.1.1 Single-ended and differential mode . . . 209
22.4.1.2 Input range . . . 210
22.4.1.3 Result data format . . . 210
22.4.1.4 Negative input selection . . . 211
22.4.1.5 Resolution . . . 211
22.4.1.6 Acquisition time . . . 212
22.4.1.7 SAR ADC clock . . . 212
22.4.1.8 SAR ADC timing . . . 213
22.4.2 SARMUX . . . 213
22.4.2.1 Analog routing . . . 214
22.4.2.2 Analog interconnection . . . 215
22.4.3 SARREF . . . 220
22.4.3.1 Reference options . . . 220
22.4.3.2 Bypass capacitors . . . 220
22.4.3.3 Input range versus reference . . . 221
22.4.4 SARSEQ . . . 221
22.4.4.1 Averaging . . . 223
22.4.4.2 Range detection . . . 223
22.4.4.3 Double buffer . . . 223
22.4.4.4 Injection channel . . . 224
22.4.5 Interrupt . . . 225
22.4.5.1 End-of-scan interrupt (EOS_INTR) . . . 225
22.4.5.2 Overflow interrupt . . . 226
22.4.5.3 Collision interrupt . . . 226
22.4.5.4 Injection end-of-conversion interrupt (INJ_EOC_INTR) . . . 226
22.4.5.5 Range detection interrupts . . . 226
22.4.5.6 Saturate detection interrupts . . . 227
22.4.5.7 Interrupt cause overview . . . 227
22.4.6 Trigger . . . 227
22.4.6.1 External trigger configuration . . . 228
22.4.7 SAR ADC status . . . 229

**Table of contents**


22.4.8 Low-power mode . . . 229
22.4.9 System operation . . . 230
22.4.9.1 SARMUX analog routing . . . 230
22.4.9.2 Global SARSEQ configuration . . . 232
22.4.9.3 Channel configurations . . . 232
22.4.9.4 Channel enables . . . 233
22.4.9.5 Interrupt masks . . . 233
22.4.9.6 Trigger . . . 233
22.4.9.7 Retrieve data after each interrupt . . . 234
22.4.9.8 Injection conversions . . . 234
22.4.10 Temperature sensor configuration . . . 234
22.5 Registers . . . 235

**23 Continuous Time Block mini (CTBm) . . . 237**
23.1 Features . . . 237
23.2 Block diagram . . . 237
23.3 How it works . . . 238
23.3.1 Power mode configuration . . . 238
23.3.2 Output strength configuration . . . 238
23.3.3 Compensation . . . 239
23.3.4 Switch control . . . 240
23.3.4.1 Input configuration . . . 240
23.3.4.2 Output configuration . . . 241
23.3.4.3 Comparator mode . . . 242
23.3.4.4 Comparator configuration . . . 242
23.3.4.5 Comparator interrupt . . . 242
23.3.4.6 Deep-Sleep mode operation . . . 243
23.4 Registers . . . 245

**24 Low-power comparator . . . 246**
24.1 Features . . . 246
24.2 Block diagram . . . 247
24.3 How it works . . . 247
24.3.1 Input configuration . . . 247
24.3.2 Output and interrupt configuration . . . 248
24.3.3 Power mode and speed configuration . . . 249
24.3.4 Hysteresis . . . 250
24.3.5 Wakeup from low-power modes . . . 250
24.3.6 Comparator clock . . . 251
24.3.7 Offset trim . . . 251
24.4 Registers . . . 251

**25 Temperature sensor . . . 252**
25.1 Features . . . 252
25.2 How it works . . . 252
25.3 Temperature sensor configuration . . . 253
25.4 Algorithm . . . 254
25.5 Registers . . . 254

**26 CAPSENSE™ . . . 255**

**Section H: Program and debug 256**

**27 Program and debug Interface . . . 257**

## Table of contents


27.1 Features . . . 257
27.2 Functional description . . . 257
27.3 Serial wire debug (SWD) interface . . . 258
27.3.1 SWD timing details . . . 259
27.3.2 ACK details . . . 260
27.3.3 Turnaround (Trn) period details . . . 260
27.4 Cortex®-M0+ debug and access port (DAP) . . . 260
27.4.1 Debug port (DP) registers . . . 261
27.4.2 Access port (AP) registers . . . 261
27.5 Programming the EZ-PD™ PMG1-S3 MCU device . . . 262
27.5.1 SWD port acquisition . . . 262
27.5.1.1 Primary and secondary SWD pin pairs . . . 262
27.5.1.2 SWD port acquire sequence . . . 262
27.5.2 SWD programming mode entry . . . 263
27.5.3 SWD programming routines executions . . . 263
27.6 EZ-PD™ PMG1-S3 MCU SWD debug interface . . . 263
27.6.1 Debug control and configuration registers . . . 263
27.6.2 Breakpoint Unit (BPU) . . . 264
27.6.3 Data Watchpoint (DWT) . . . 264
27.6.4 Debugging the EZ-PD™ PMG1-S3 MCU device . . . 264
27.7 Registers . . . 265

**28 Nonvolatile memory Programming . . . 266**
28.1 Features . . . 266
28.2 Functional description . . . 266
28.3 System call implementation . . . 267
28.4 Blocking and non-blocking system calls . . . 267
28.4.1 Performing a system call . . . 268
28.5 System calls . . . 269
28.5.1 Silicon ID . . . 270
28.5.2 Configure clock . . . 271
28.5.3 Load flash bytes . . . 272
28.5.4 Write row . . . 273
28.5.5 Program row . . . 274
28.5.6 Erase all . . . 275
28.5.7 Checksum . . . 276
28.5.8 Write protection . . . 277
28.5.9 Non-blocking write row . . . 278
28.5.10 Non-blocking program row . . . 279
28.5.11 Resume non-blocking . . . 280
28.6 System call status . . . 281
28.7 Non-blocking system call pseudo code . . . 282

**References 284**

**Terminology 285**

# Section A: Overview

This section encompasses the following chapters:

- **"Introduction"** on page 13
- **"Getting started"** on page 24
- **"Document construction"** on page 25

## Revision history

| Document version | Date of release | Description of changes |
|---|---|---|
| ** | 2021-12-01 | Initial release |
| *A | 2022-03-04 | Removed confidential status<br>Updated USB PD in **Figure 1-2**<br>Updated USB PD in **"EZ-PD™ PMG1-S3 MCU USB Power Delivery block diagram"** on page 178<br>Updated **"USB Power Delivery (USB PD)"** on page 179<br>Updated pins in **Figure 23-1**, **Table 23-4**, and **Table 23-5** |

# 1 Introduction

EZ-PD™ Power Delivery Microcontroller Gen1 (PMG1) is a family of high-voltage USB-C Power Delivery (PD) microcontrollers (MCU). These chips include an Arm® Cortex®-M0/M0+ CPU and USB-C PD controller along with analog and digital peripherals. EZ-PD™ PMG1-S3 MCU is targeted for any embedded system that provides/consumes power to/from a high-voltage USB-C PD port and leverages the microcontroller to provide additional control capability.

**Figure 1-1** shows the EZ-PD™ PMG1 MCU family segmentation.

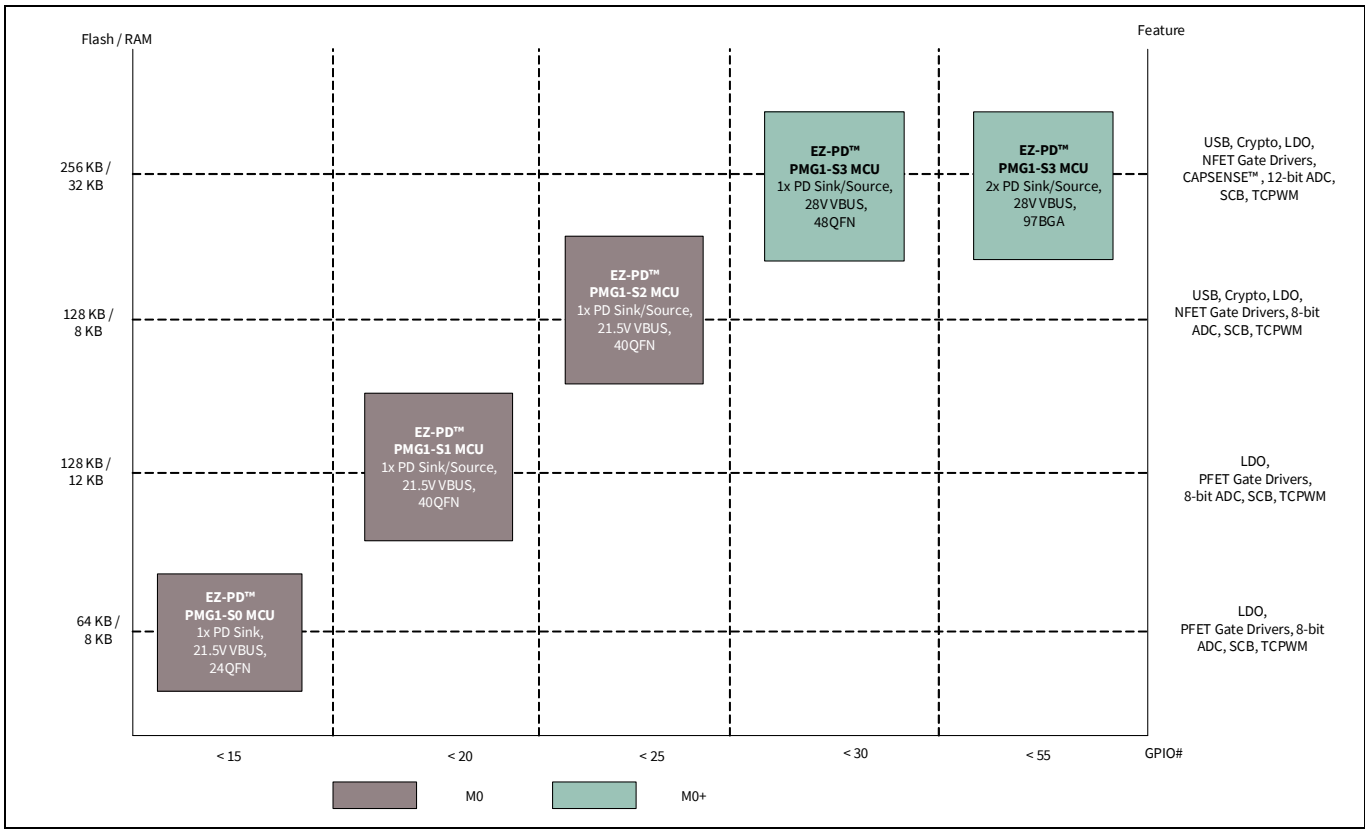



**Figure 1-1. EZ-PD™ PMG1 MCU family segmentation**

**Table 1-1** shows the comparison of features of different MCUs of the EZ-PD™ PMG1 MCU family. The rest of the document discusses EZ-PD™ PMG1-S3 MCU in detail.

**Table 1-1. EZ-PD™ PMG1 MCU family feature comparison**

| Subsystem or range | Item | EZ-PD™ PMG1-S0 MCU | EZ-PD™ PMG1-S1 MCU | EZ-PD™ PMG1-S2 MCU | EZ-PD™ PMG1-S3 MCU |
|---|---|---|---|---|---|
| CPU and memory subsystem | Core | Arm® Cortex®-M0 | Arm® Cortex®-M0 | Arm® Cortex®-M0 | Arm® Cortex®-M0+ |
| | Max. Freq (MHz) | 48 | 48 | 48 | 48 |
| | Flash (KB) | 64 | 128 | 128 | 256 |
| | SRAM (KB) | 8 | 12 | 8 | 32 |
| Power Delivery | Power Delivery ports | 1 | 1 | 1 | 1 port for 48-QFN<br>2 ports for 97-BGA |
| | Role | Sink | DRP | DRP | DRP |
| | MOSFET gate drivers | 1x PFET | 2x PFET | 2x NFET | Flexible 2x NFET |
| | Fault protections | VBUS OVP and UVP | VBUS OVP, UVP, and OCP.<br>SCP and RCP (for source configuration only) | VBUS OVP, UVP and OCP | VBUS OVP, UVP and OCP.<br>SCP and RCP (for source configuration only) |
| USB | Integrated Full Speed USB 2.0 device with Billboard Class support | No | No | Yes | Yes |
| Voltage range | Supply (V) | VDDD (2.7 - 5.5)<br>VBUS (4 - 21.5) | VSYS (2.75 - 5.5)<br>VBUS (4 - 21.5) | VSYS (2.7 - 5.5)<br>VBUS (4 - 21.5) | VSYS (2.8 - 5.5)<br>VBUS (4 - 28) |
| | IO (V) | 1.71 - 5.5 | 1.71 - 5.5 | 1.71 - 5.5 | 1.71 – 5.5 |
| Digital | SCB (configurable as I2C/UART/SPI) | 2 | 4 | 4 | 7 for 48-QFN (out of which only 5 can be configured as SPI and UART)<br>8 for 97-BGA |
| | TCPWM block (configurable as timer, counter or pulse width modulator) | 4 | 2 | 4 | 7 for 48-QFN<br>8 for 97-BGA |
| | Hardware authentication block (Crypto) | No | No | Yes (AES-128/192/256, SHA1, SHA2-224, SHA2-256, PRNG, CRC) | Yes (AES-128, SHA2-256, TRNG, vector unit) |

**Table 1-1. EZ-PD™ PMG1 MCU family feature comparison** (continued)

| Subsystem or range | Item | EZ-PD™ PMG1-S0 MCU | EZ-PD™ PMG1-S1 MCU | EZ-PD™ PMG1-S2 MCU | EZ-PD™ PMG1-S3 MCU |
|---|---|---|---|---|---|
| Analog | ADC | 2x 8-bit SAR | 1x 8-bit SAR | 2x 8-bit SAR | 2x 8-bit SAR<br>1x 12-bit SAR |
| | On-chip temperature sensor | Yes | Yes | Yes | Yes |
| Direct memory access (DMA) | DMA | No | No | No | Yes |
| GPIO | Max # of I/Os | 12 (10+2 OVT) | 17 (15+2 OVT) | 20 (18+2 OVT) | 26 (24+2 OVT) for 48-QFN<br>50 (48+2 OVT) for 97-BGA |
| Charging standards | Charging source | – | BC 1.2, AC | BC 1.2, AC | BC 1.2, AC, AFC and Quick Charge 3.0 |
| | Charging sink | BC 1.2, Apple Charging (AC) | BC 1.2, AC | BC 1.2, AC | BC 1.2, AC |
| ESD protection | ESD protection | Yes (Up-to ± 8-kV contact discharge, up-to ±15-kV air discharge, human body model, and charged device model) | Yes (Human body model and charged device model) | Yes (Up-to ± 8-kV contact discharge, up-to ±15-kV air discharge human body model, and charged device model) | Yes (Human body model and charged device model) |
| Packages | Package options | 24 QFN (4 × 4 mm, 0.5 mm pitch) | 40-QFN (6 × 6 mm, 0.5 mm pitch) | 40-QFN(6 × 6 mm, 0.5 mm pitch) | 48-QFN(6 × 6 mm, 0.5 mm pitch)<br>97BGA(6 × 6 mm, 0.5 mm and 0.65 mm pitch) |

## Introduction

EZ-PD™ PMG1-S3 MCU is a single/dual port USB Type-C and PD controller. The controller complies with the latest USB Type-C 1.3 and Power Delivery (PD) 3.0 standards. EZ-PD™ PMG1-S3 MCU expands the EZ-PD™ PMG1 MCU family with 256-KB Flash, 32-KB SRAM, 50 GPIOs, full-speed USB device controller, a Crypto engine for authentication, analog resources (ADC, opamp, comparators, CAPSENSE™), and dual Type-C PD ports. It is targeted for device, dock, and accessory applications and is available in QFN and BGA packages. It can also be used in dual-role-port (DRP) and downstream facing port (DFP) applications. It uses Infineon' proprietary M0S8 technology with a 32-bit, 48-MHz Arm® Cortex®-M0+ processor with 256-KB flash and integrates two Type-C transceivers including the Type-C termination resistors Rp and Rd. EZ-PD™ PMG1-S3 MCU devices have these characteristics:

- Supports upto two USB Type-C ports
- High-performance, 32-bit single-cycle Cortex®-M0+ CPU core
- Configurable timer, counter, and PWM blocks
- Configurable serial communication blocks (SCB) to support $I^2C$, SPI, and UART
- Integrated oscillator eliminating the need for an external clock
- Integrated UFP (Rd) and current sources (Rp) termination resistors
- Integrated dead battery termination resistors for DRP applications
- Integrated legacy charging protocols charger block
- High-voltage protection on CC and SBU lines
- Integrated VCONN FETs
- Hardware Crypto block enables Authentication
- Full-speed USB device controller supporting Billboard device class
- Integrated Load Switch in the VBUS provider path and capable of detecting OC (over current), RC (reverse current), UV (under voltage), OV (over voltage), and SC (short circuit) events

This document describes each function block of the EZ-PD™ PMG1-S3 MCU device in detail. This information will help designers to create system-level designs.

## 1.1 Top level architecture

**Figure 1-2** shows the major components of the EZ-PD™ PMG1-S3 MCU architecture.

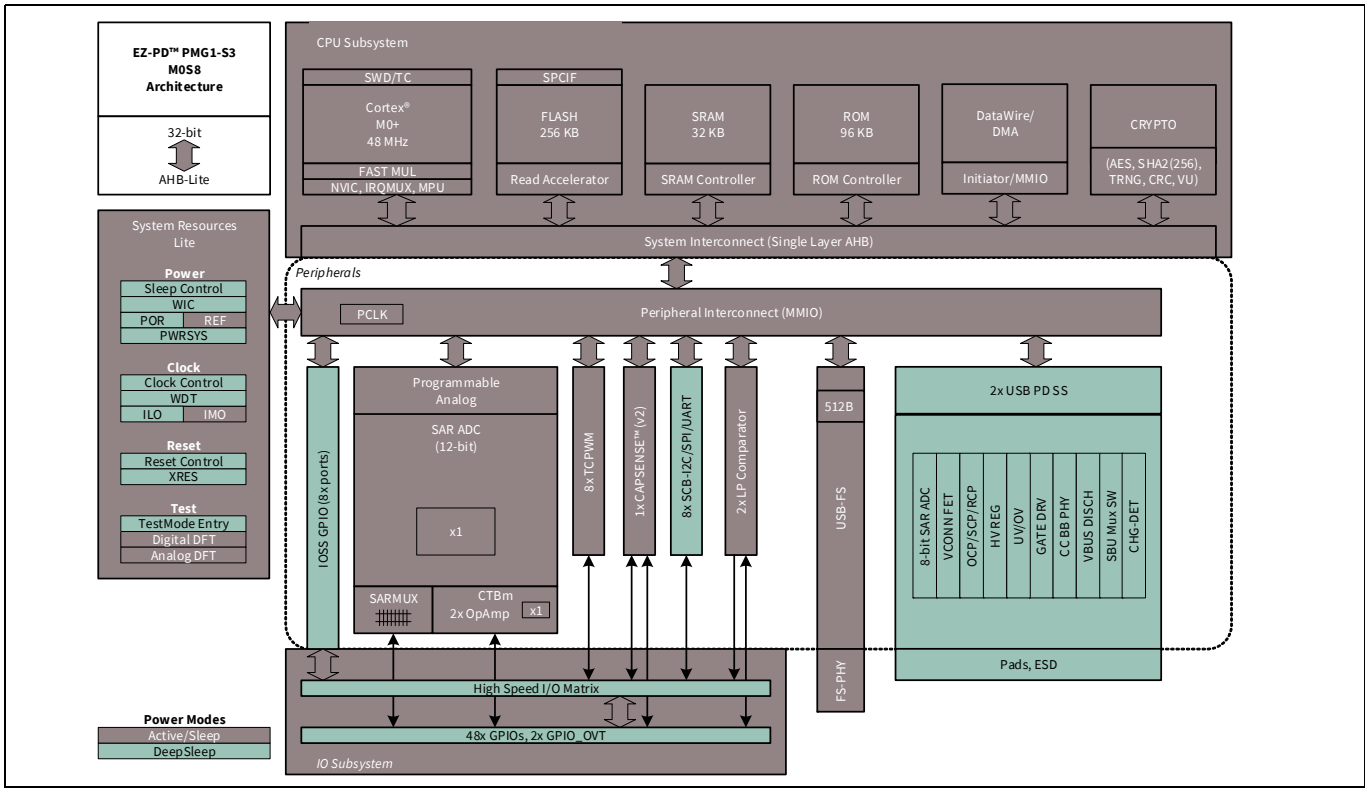



**Figure 1-2. EZ-PD™ PMG1-S3 MCU block diagram**

## 1.2 Features

The EZ-PD™ PMG1-S3 MCU has these major components:

- 32-bit MCU subsystem
- 48-MHz Arm® Cortex®-M0+ CPU with DMA
- Memory
  - 256-KB flash
  - 32-KB SRAM and
  - 96-KB ROM with PD code
- Type-C/PD blocks
  - Up to two Type-C/PD blocks each with a baseband transceiver
  - Two integrated VBUS NFET gate drivers
  - Slew rate control to limit the inrush current on the gate drivers configured to be used in VBUS provider or consumer path
  - Integrated USB Power-Delivery (USB-PD) 3.0 support
  - Supports 28 V Extended Power Range (EPR)
  - High-voltage (28 V) regulator and VBUS discharge
  - Configurable VBUS over-voltage protection (OVP), overcurrent protection (OCP), short-circuit protection (SCP) and reverse-current (RCP) protection
  - VCONN FETs with OCP
  - Two integrated 3:1 SBU analog muxes for alternate modes (Display Port and Thunderbolt) on the 97-BGA part

- Interfaces
  - Up to eight run-time reconfigurable serial communication blocks (SCBs) configurable as $I^2C$, SPI or UART
  - Up to eight timer/counter pulse-width modulators (TCPWMs)
- Programmable GPIO pins
  - Up to 50 GPIO pins
  - Any GPIO pin can be CAPSENSE™, analog, or digital
  - Programmable drive modes, strengths, and slew rates
- Integrated analog blocks
  - Two 8-bit SAR ADCs
  - One 12-bit SAR ADC
  - Two opamps
  - Two LP Comparators
- Capacitive sensing
  - Infineon CAPSENSE™ Sigma-Delta (CSD) provides best-in-class signal-to-noise ratio (SNR) (>5:1) and water tolerance
  - Infineon-supplied software component makes capacitive sensing design easy
  - Automatic hardware tuning (SmartSense)
- Hardware Crypto engine for Secure FW boot and Signed FW update
- USB Full-speed device
- Charger detect block
- Power
  - VSYS (2.8 V to 5.5 V)
  - VBUS (4 V to 28 V)
  - Independent supply voltage pin for GPIO that allows 1.71 V to 5.5 V signaling on the I/Os
- Packages
  - 48-QFN
  - 97-BGA
- Software tool
  - ModusToolbox™ software

## 1.3 CPU system

### 1.3.1 Processor

The Cortex® M0+ in EZ-PD™ PMG1-S3 MCU is a 32-bit MCU, which is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set, which enables fully compatible binary upwards migration of code to higher performance processors such as the Cortex® M3 and M4. The MCU includes a hardware multiplier, which provides a 32-bit result in one cycle. The subsystem contains 256 KB of flash organized in two banks of 128 KB. The two-bank flash allows the system to maintain two copies of the firmware and update one copy while continuing to execute from the other copy.

### 1.3.2 Interrupt controller

The CPU subsystem of EZ-PD™ PMG1-S3 MCU includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode. The Arm® Cortex®-M0+ CPU provides a Non Maskable Interrupt (NMI) input, which is made available to the user when it is not in use for system functions requested by the user.

### 1.3.3 Direct memory access

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

### 1.3.4 Memory

The EZ-PD™ PMG1-S3 MCU memory subsystem consists of a 256 KB flash module 32 KB SRAM, 8 KB of boot SROM, and 88 KB of user SROM.

### 1.3.5 Crypto

The EZ-PD™ PMG1-S3 MCU Crypto block provides cryptography functionality. It includes hardware acceleration blocks for AES (Advanced Encryption Standard) block cipher, SHA-1 (Secure Hash Algorithm) and SHA-2 hash, CRC (Cyclic Redundancy Check), and true random number generation.

## 1.4 System-wide resources

### 1.4.1 Clocking system

The clock system for the EZ-PD™ PMG1-S3 MCU controller consists of the internal main oscillator (IMO) and an internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock.

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the EZ-PD™ PMG1-S3 MCU. The default IMO frequency is 48 MHz. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a 32-kHz low-power, low accuracy oscillator and is used to generate clocks for peripheral operation in Deep- Sleep power mode.

### 1.4.2 Power system

EZ-PD™ PMG1-S3 MCU is capable of operating from three possible external power sources: VSYS, VBUS_C_P0, and VBUS_C_P1. The VSYS input supports operation in the range 2.8 V to 5.5 V. This VSYS range is acceptable for DFP and DRP applications. The MCU has three power modes - Active (default mode), Sleep, and Deep-Sleep. The transitions between these modes are managed by the power system. A separate power domain VDDIO is provided for the GPIOs. The VCCD pin, the output of the core (1.8 V) regulator, is brought out only for connecting a 0.1-µF capacitor for stability. This pin is not supported as a power supply.

In Active mode, the CPU runs with all the peripherals and other subsystems powered. In Sleep mode, the CPU clock is gated (turned off). In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention (states retained); the main system clock (IMO) is OFF while the low-frequency clock (ILO) is ON and the low-frequency peripherals are in operation. Multiple internal regulators are available in the system to support power supply in different power modes.

The V5V supply is used to power Type-C cable markers through the VConn switches and supports operation in the range 4.85 V to 5.5 V. The MCU cannot be powered through this supply.

### 1.4.3 Watchdog timers

The EZ-PD™ PMG1-S3 MCU device has one 16-bit watchdog timer, which is capable of automatically resetting the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality.

### 1.4.4 Voltage reference

The EZ-PD™ PMG1-S3 MCU reference system generates all internally required references. To allow better signal to noise ratios (SNR) and better absolute accuracy, it is possible to bypass the internal reference using a GPIO pin or to use an external reference for the 12-bit SAR ADC. The internal reference at the pin may be buffered by using one of the on-chip op amps and used as an external reference.

## 1.5 USB PD subsystem

This subsystem provides the interface to the Type-C USB port. This sub-system comprises of USB PD physical layer, VCONN FETs, ADC, SBU pass-through switch and SBU Mux, under-voltage and over-voltage protection on VBUS, load switch controller with integrated FET (5 V/3 A) for the provider path, high-side current sense amplifier for VBUS, VBUS discharge, VBUS regulator, NFET gate drivers for VBUS. This sub-system also includes HBM ESD support on the Type-C port.

### 1.5.1 USB PD physical layer

The USB PD sub-system contains the USB PD physical layer block and supporting circuits. The USB PD physical layer consists of a transmitter and receiver that communicate BMC encoded data over the CC channel per the PD 3.0 standard. All communication is half-duplex. The physical layer or PHY practices collision avoidance to minimize communication errors on the channel.

USB PD block includes termination resistors Rp and Rd as required by the USB Type-C spec. Rp and Rd resistors are required to implement connection detection, plug orientation detection and for establishment of the USB source/sink roles. The Rp resistor is implemented as a current source. The dead battery Rd resistor on CC pins is required for sink termination detection and charging in the unpowered state.

### 1.5.2 VCONN FET

EZ-PD™ PMG1-S3 MCU has a power supply input, V5V pin, for providing power to EMCA cables through integrated VCONN FETs. There are two VCONN FETs in MCU to power either CC1 or CC2 pins. These are power supply input VCONN_Source pins for providing power to EMCA cables through these VCONN FETs. These FETs can provide a maximum of 1.5 W power over VCONN with voltage range 4.85 V to 5.5 V on the CC1 and CC2 pins for the EMCA cables. At any given time, only one VCONN FET will be enabled on the non-CC line. The FETs have over current protection and support 700 mΩ RDSon resistance.

### 1.5.3 8-bit SAR ADC

The ADC is a low-footprint 8-bit SAR ADC available for general purpose A-D conversion applications in the chip. The ADC can be accessed from the GPIOs through an on-chip analog mux. In EZ-PD™ PMG1-S3 MCU, one ADC is instantiated per PD port.

### 1.5.4 SBU pass-through switch and USB HS mux

EZ-PD™ PMG1-S3 MCU is integrated with two SBU switches connects SBU1 and SBU2 pins of the Type-C connector to AUX of a DisplayPort or LSx of Thunderbolt and UART debug pins.

### 1.5.5 Under-voltage and over-voltage protection on VBUS

The chip supports dedicated comparators for detecting over-voltage and under-voltage faults. The thresholds for both UV and OV are made programmable through firmware.

### 1.5.6 Integrated load switch controller for provider path

EZ-PD™ PMG1-S3 MCU Load Switch controller supports the following features:

The VBUS provider path FET has capability to soft turn on to avoid the very high inrush currents, to avoid the reliability issues for the Power FET. Refer to the **device datasheet** for the soft turn on specification.

EZ-PD™ PMG1-S3 MCU has capability to detect the VBUS over current above the PD contract value for provider path. By default, the OCP threshold is set at 30% higher than the PD contract current. The part automatically shuts off the provider path when the OCP event is triggered. OCP thresholds can be configured using firmware. Contact Infineon support for changing the thresholds.

EZ-PD™ PMG1-S3 MCU can detect reverse current whenever VBUS at Type-C connector is greater than VBUS at provider side regulator. After recognizing the RCP event, the provider FET is turned off to isolate the provider and connector VBUS. It has three distinct RCP mechanisms.

EZ-PD™ PMG1-S3 MCU can detect short circuit faults on the Type-C VBUS. It has two programmable thresholds of 6 A and 10 A.

### 1.5.7 High-side current sense amplifier for VBUS

EZ-PD™ PMG1-S3 MCU supports over-current (OC) protection on the VBUS path using a current sense amplifier. The voltage drop across this precision resistor is monitored to sense the magnitude of current on the VBUS path and compare against a programmable threshold.

### 1.5.8 VBUS discharge

EZ-PD™ PMG1-S3 MCU supports VBUS discharge circuitry inside. After cable removal detection, the chip will discharge the residual charge and bring the floating VBUS back to 0.8 V.

### 1.5.9 VBUS regulator

EZ-PD™ PMG1-S3 MCU has three input power supplies - VSYS, VBUS_C_P0 and VBUS_C_P1. A regulator operating on these three power supplies will derive the chip operating supply. The VSYS always takes priority over VBUS_C_P0/ P1. In absence of VSYS, the regulator powers the chip from VBUS_C_P0/P1 whichever is present.

### 1.5.10 NFET gate driver for VBUS

EZ-PD™ PMG1-S3 MCU supports consumer side external power FET drivers with soft start capability to limit inrush current. Only NFET external drivers are supported.

## 1.6 Fixed-function digital

### 1.6.1 Timer/Counter/PWM block

The Timer/Counter/PWM(TCPWM) block consists of a 16-bit counter with user-programmable period length. The functionality of these counters can be synchronized. Each block has a capture register, period register, and compare registers. EZ-PD™ PMG1-S3 MCU has up to eight TCPWMs. They can be used as internal timers by firmware or for providing PWM based functions on the GPIOs.

### 1.6.2 Serial Communication Block (SCB)

The EZ-PD™ PMG1-S3 MCU has upto eight SCBs, which can each implement a serial communication interface as $I^2C$, universal asynchronous receiver/transmitter (UART), or serial peripheral interface (SPI). The SCBs are configurable to $I^2C$/SPI/UART. For more details see the **device datasheet**.

The features of SCB include:

- Standard $I^2C$ multi-master and slave functionality compatible with the standard NXP Semiconductor I2C Specification V3.0
- Standard SPI master and slave function with Motorola, TI, and National Semiconductor (MicroWire) modes
- Standard UART transmitter and receiver function

### 1.6.3 GPIO

EZ-PD™ PMG1-S3 MCU 97-BGA has 50 GPIOs that include dedicated GPIOs, SCBs, and SWD pins which can also be used as GPIOs. EZ-PD™ PMG1-S3 MCU 48-QFN package has 26 GPIOs. Every GPIO in EZ-PD™ PMG1-S3 MCU has the following characteristics:

- Eight drive strength modes
- Input threshold select (CMOS or LVTTL)
- Individual control of input and output disables
- Selectable slew rates for dV/dt related noise control to improve EMI
- Hold mode to latch the previous state (used to retain I/O state in Deep Sleep mode)
- Interrupt generation - edge and level triggered

## 1.7 Analog systems

### 1.7.1 12-bit SAR ADC

The EZ-PD™ PMG1-S3 MCU device has a configurable 12-bit 1-Msps SAR ADC. The ADC provides three internal voltage references (VDDA, VDDA /2, and VREF) and an external reference through a GPIO pin. The SAR hardware sequencer is available, which scans multiple channels without CPU intervention.

### 1.7.2 Continuous time block mini

The continuous time block mini (CTBm) provides continuous time functionality at the entry and exit points of the analog subsystem. The CTBm has two highly configurable and high-performance opamps with a switch routing matrix. The opamps can also work in comparator mode. The EZ-PD™ PMG1-S3 MCU device has one such CTBm block. The block allows open-loop opamp, linear buffer, and comparator functions to be performed without external components. PGAs, voltage buffers, filters, and trans-impedance amplifiers can be realized with external components. The CTBm block can work in Active, Sleep, and Deep-Sleep modes.

### 1.7.3 Temperature sensor

EZ-PD™ PMG1-S3 MCU has an on-chip temperature sensor which consists of a diode biased by a current source that can be disabled to save power. The diode is calibrated during production to achieve ±5% maximum deviation from accuracy (typical ±1%). Since the measured temperature is the on-chip temperature of the diode, the diode is placed in close proximity to the SAR ADC to allow more accurate measurement.

### 1.7.4 Low-power comparators

The EZ-PD™ PMG1-S3 MCU device has a pair of low-power comparators, which can operate in all device power modes. This functionality allows the CPU and other system blocks to be disabled while retaining the ability to monitor external voltage levels during low-power modes. Both comparator inputs can come from the GPIO pins, or one can come from an internal signal via AMUXBUS also.

## 1.8 Special function peripherals

### 1.8.1 CAPSENSE™

CAPSENSE™ is supported on 16 pins in EZ-PD™ PMG1-S3 MCU via a CAPSENSE™ Sigma-Delta (CSD) block that can be connected to any pin via the analog mux buses that any GPIO pin can be connected to. CAPSENSE™ function can thus be provided on any pin or group of pins in a system under software control.

### 1.8.2 USB 2.0 Full Speed device and charger detection

EZ-PD™ PMG1-S3 MCU has one USB 2.0 FS device to support billboard class, and HID class applications for firmware download. The charger detection block connected to the DP/DM pins allows EZ-PD™ PMG1-S3 MCU to detect conventional battery chargers conforming to BC 1.2, Apple Charger, QC3.0 and AFC specifications. The QC and AFC protocols are supported for source only.

## 1.9 Program and debug

EZ-PD™ PMG1-S3 MCU devices support programming and debug features of the device via the on-chip SWD interface.

# 2 Getting started

## 2.1 EZ-PD™ PMG1 MCU resources

This chapter provides the complete list of EZ-PD™ PMG1 MCU resources that will help you get started with the device and design your applications with them. If you are new to EZ-PD™ PMG1 MCU, there is a wealth of data at **www.infineon.com** to help you to select the right EZ-PD™ PMG1 MCU device and quickly and effectively integrate it into your design.

The following is an abbreviated list of EZ-PD™ PMG1 MCU resources:

- Overview: **EZ-PD™ PMG1 MCU webpage**
- Datasheets describe and provide electrical specifications for each device family.
- Application notes and code examples cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples, which can be opened from ModusToolbox™ software.
- Technical reference manuals (TRMs) provide detailed descriptions of the architecture and registers in each device family.
- Development tools
  - **ModusToolbox™** software is a free integrated design environment (IDE). It enables you to design hardware and firmware systems concurrently with EZ-PD™ PMG1 MCU devices. In addition, ModusToolbox™ software includes a device selection tool to select devices for ModusToolbox™ software projects.
  - EZ-PD™ PMG1 MCU prototyping kits offer an easy-to-use, inexpensive platform to build EZ-PD™ PMG1 MCU-based systems.
- Additional resources: Visit the **EZ-PD™ PMG1 MCU** webpage for additional resources such as IBIS, BSDL models, CAD library files, and programming specifications.
- Technical support
  - Forum: See if your question is already answered by fellow developers of the EZ-PD™ PMG1 MCU community.
  - Infineon support: Visit our support page or contact a local sales representative. Free support for products is available online at **www.infineon.com**. Resources include application notes, CRM technical support email, knowledge base articles, and application support engineers.

## 2.2 Product upgrades

Infineon provides scheduled upgrades and version enhancements for the EZ-PD™ PMG1-Sx MCU software development kit (SDK) which includes firmware and tools for the EZ-PD™ PMG1 MCU family. Upgrades can be downloaded from the Software and drivers section of **www.infineon.com**. Critical updates to system documentation are provided in the Documentation section.

# 3 Document construction

This document has the following sections:


- **"CPU system"** on page 31
- **"Memory system"** on page 73
- **"System-wide resources"** on page 76
- **"Digital system"** on page 111
- **"USB power and data"** on page 178
- **"Analog system"** on page 206
- **"Program and debug"** on page 256


## 3.1 Major sections

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

- Section – Presents the top-level architecture, how to get started, and conventions and overview information about any particular area that inform the reader about the construction and organization of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These provide the detailed information about the topic.
- Terminology – Defines the specialized terminology used in this technical reference manual (TRM). Terminology terms are presented in bold, italic font throughout.
- Registers technical reference manual – Supplies all device register details summarized in the technical reference manual. These are additional documents.

## 3.2 Documentation conventions

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

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

### 3.2.1 Register conventions

Register conventions are detailed in the EZ-PD™ PMG1-S3 MCU registers TRM.

### 3.2.2 Numeric naming

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

### 3.2.3 Units of measure

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

**Table 3-1. Units of measure**

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

### 3.2.4 Acronyms

This table lists the acronyms used in this document.

**Table 3-2. Acronyms**

| Acronym | Description |
|---|---|
| ABUS | analog output bus |
| AC | alternating current |
| ADC | analog-to-digital converter |
| AES | advanced encryption standard |
| AHB | AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm® data transfer bus |
| API | application programming interface |
| APOR | analog power-on reset |
| BMC | Bi-Phase Mark Coding |
| BR | bit rate |
| BRA | bus request acknowledge |
| BRQ | bus request |
| CAN | controller area network |
| CC | Configuration Channel |
| CI | carry in |
| CMP | compare |
| CO | carry out |
| CPU | central processing unit |
| CRC | cyclic redundancy check |
| CT | continuous time |
| DAC | digital-to-analog converter |
| DC | direct current |
| DFP | Downstream Facing Port |
| DI | digital or data input |
| DMA | direct memory access |
| DNL | differential nonlinearity |
| DO | digital or data output |
| DRP | dual-role-power |
| DSI | digital signal interface |
| DSM | deep-sleep mode |
| ECO | external crystal oscillator |
| EEPROM | electrically erasable programmable read only memory |
| EMCA | Electronically Marked Cable Assembly |
| EMIF | external memory interface |
| FB | feedback |
| FIFO | first in first out |
| FRS | Fast Role Swap |

**Table 3-2. Acronyms** (continued)

| Acronym | Description |
|---|---|
| FSR | full scale range |
| GPIO | general purpose I/O |
| HCI | host-controller interface |
| HFCLK | high-frequency clock |
| $I^2C$ | inter-integrated circuit |
| IDE | integrated development environment |
| ILO | internal low-speed oscillator |
| IMO | internal main oscillator |
| INL | integral nonlinearity |
| I/O | input/output |
| IOR | I/O read |
| IOW | I/O write |
| IRES | initial power on reset |
| IRA | interrupt request acknowledge |
| IRQ | interrupt request |
| ISR | interrupt service routine |
| IVR | interrupt vector read |
| LRb | last received bit |
| LRB | last received byte |
| LSb | least significant bit |
| LSB | least significant byte |
| LUT | lookup table |
| MISO | master-in-slave-out |
| MMIO | memory mapped input/output |
| MOSI | master-out-slave-in |
| MSb | most significant bit |
| MSB | most significant byte |
| OCP | overcurrent protection |
| OVP | overvoltage protection |
| OVT | over voltage tolerant |
| PC | program counter |
| PCH | program counter high |
| PCL | program counter low |
| PD | power down |
| PGA | programmable gain amplifier |
| PM | power management |
| POR | power-on reset |
| PPOR | precision power-on reset |

**Table 3-2. Acronyms** (continued)

| Acronym | Description |
|---|---|
| PRNG | pseudo random number generation |
| PRS | pseudo random sequence |
| PSRR | power supply rejection ratio |
| PSSDC | power system sleep duty cycle |
| PWM | pulse width modulator |
| RAM | random-access memory |
| RCP | reverse current protection |
| RETI | return from interrupt |
| RF | radio frequency |
| ROM | read only memory |
| RW | read/write |
| SAR | successive approximation register |
| SC | switched capacitor |
| SCB | serial communication block |
| SCP | short circuit protection |
| SIE | serial interface engine |
| SIO | special I/O |
| SE0 | single-ended zero |
| SHA | secure hash algorithm |
| SNR | signal-to-noise ratio |
| SOF | start of frame |
| SOI | start of instruction |
| SOP | Start of Packet |
| SP | stack pointer |
| SPD | sequential phase detector |
| SPI | serial peripheral interconnect |
| SPIM | serial peripheral interconnect master |
| SPIS | serial peripheral interconnect slave |
| SRAM | static random-access memory |
| SROM | supervisory read only memory |
| SSADC | single slope ADC |
| SSC | supervisory system call |
| SYSCLK | system clock |
| SWD | serial wire debug |
| TC | terminal count |
| TD | transaction descriptors |
| TRNG | true random number generation |
| UART | universal asynchronous receiver/transmitter |

**Table 3-2. Acronyms** (continued)

| Acronym | Description |
|---|---|
| UFP | Upstream Facing Port |
| USB | universal serial bus |
| USBIO | USB I/O |
| UVP | undervoltage protection |
| USB PD | USB Power Delivery |
| WCO | watch crystal oscillator |
| WDT | watchdog timer |
| WDR | watchdog reset |
| XRES | external reset |
| XRES_N | external reset, active low |

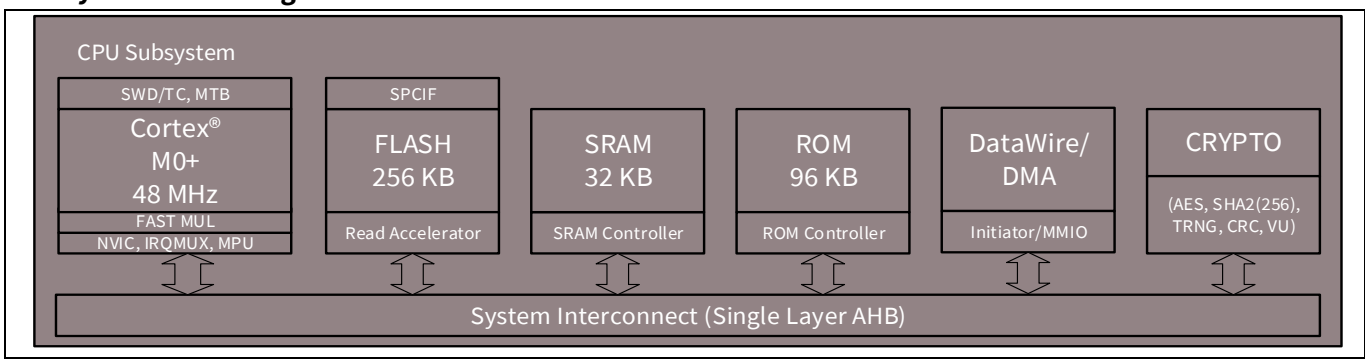
# Section B: CPU system

This section encompasses the following chapters:

- **“Cortex®-M0+ CPU”** on page 32
- **“Interrupts”** on page 38
- **“DMA controller modes”** on page 49
- **“Cryptography”** on page 65

## Top level architecture

### CPU system block diagram



# 4 Cortex®-M0+ CPU

The EZ-PD™ PMG1-S3 MCU Arm® Cortex®-M0+ core is a 32-bit CPU optimized for low-power operation. It has an efficient three-stage pipeline, a fixed 4-GB memory map, and supports the ARMv6-M Thumb instruction set. The Cortex®-M0+ also features a low-latency interrupt service routine (ISR) entry and exit. The Cortex®-M0+ processor includes a number of other components that are tightly linked to the CPU core. These include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

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

## 4.1 Features

The EZ-PD™ PMG1-S3 MCU Cortex®-M0+ has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Supports the Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Implements design time configurable memory protection unit (MPU)
- Supports unprivileged and privileged mode execution
- Supports optional vector table offset register (VTOR)
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

## 4.2 Block diagram

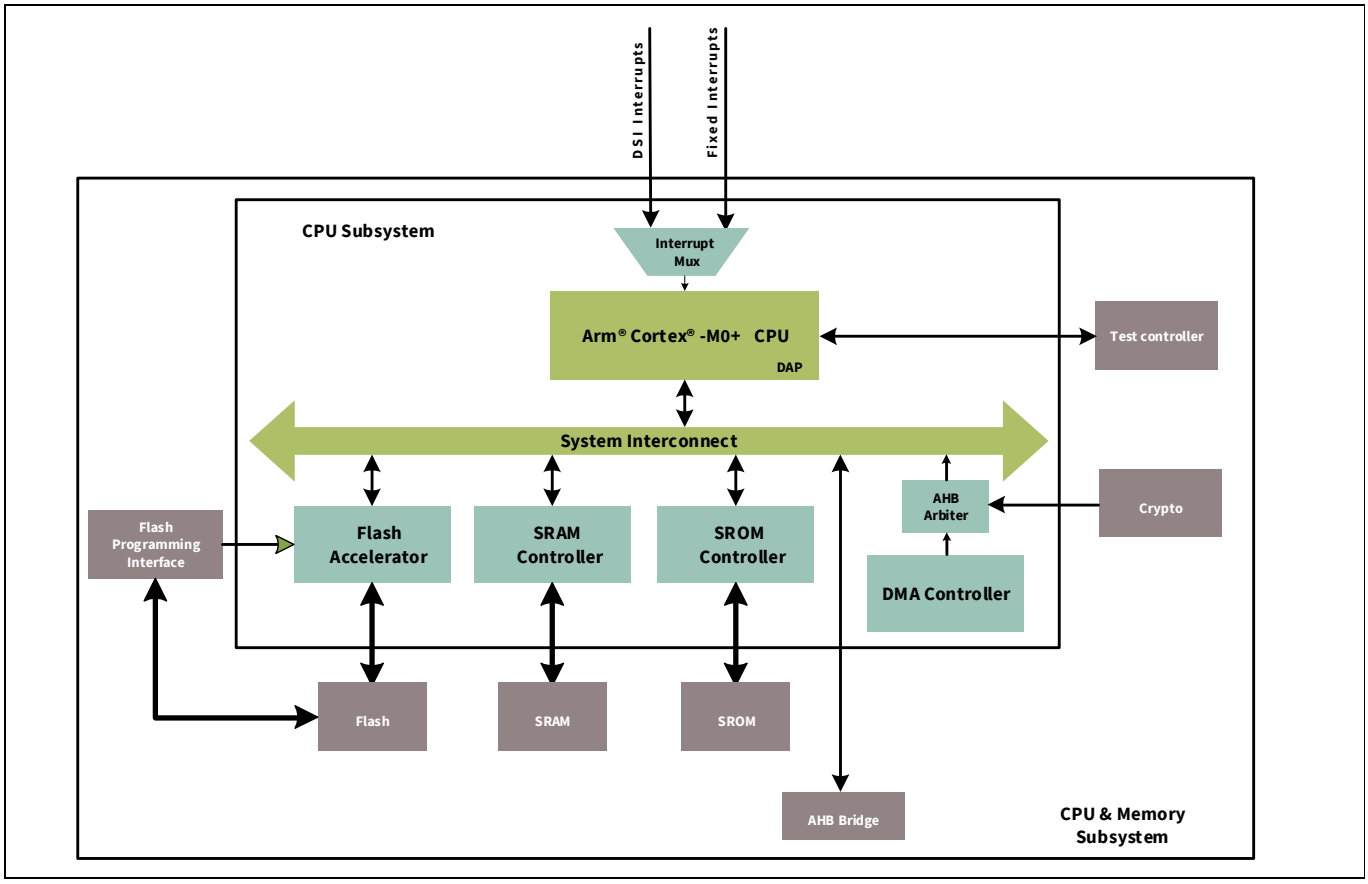



**Figure 4-1. EZ-PD™ PMG1-S3 MCU CPU subsystem block diagram**

## 4.3 How it works

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

The processor supports two operating modes (see **"Operating modes"** on page 35). It has a single-cycle 32-bit multiplication instruction.

### 4.3.1 Registers

The Cortex®-M0+ has 16 32-bit registers, as **Table 4-1** shows:

- R0 to R12 – General-purpose registers. R0 to R7 can be accessed by all instructions; the other registers can be accessed by a subset of the instructions.
- R13 – Stack pointer (SP). There are two stack pointers, with only one available at a time. In thread mode[1)], the CONTROL register indicates the stack pointer to use, main stack pointer (MSP) or process stack pointer (PSP).
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

1) Thread mode is an Arm® processor mode. It is used to execute the application software. The processor enters this mode when it comes out of reset. For details, see the **Cortex®-M0+ generic user guide**.

**Table 4-1. Cortex®-M0+ registers**

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

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

b) Bit[24] is the T bit and is loaded from bit[0] of the reset vector.

Table 4-2 shows how the PSR bits are assigned.

**Table 4-2. Cortex®-M0+ PSR bit assignments**

| Bit | PSR register | Name | Usage |
|---|---|---|---|
| 31 | APSR | N | Negative flag |
| 30 | APSR | Z | Zero flag |
| 29 | APSR | C | Carry or borrow flag |
| 28 | APSR | V | Overflow flag |
| 27–25 | – | – | Reserved |
| 24 | EPSR | T | Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception. |

**Table 4-2. Cortex®-M0+ PSR bit assignments** (continued)

| Bit | PSR register | Name | Usage |
|---|---|---|---|
| 23–6 | – | – | Reserved |
| 5–0 | IPSR | N/A | Exception number of current ISR:<br>0 = thread mode<br>1 = reserved<br>2 = Non maskable interrupt (NMI)<br>3 = HardFault<br>4 – 10 = reserved<br>11 = SVCall<br>12, 13 = reserved<br>14 = PendSV<br>15 = SysTick<br>16 = IRQ0<br>…<br>47 = IRQ31 |

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset are disabled. See the **"Interrupts"** on page 38 for a list of exceptions.

### 4.3.2 Operating modes

The Cortex®-M0+ processor supports two operating modes:

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

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

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

### 4.3.3 Instruction set

The Cortex®-M0+ implements a version of the Thumb instruction set. For details, see the **Cortex®-M0+ generic user guide**.

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

**Table 4-3. Thumb instruction set**

| Mnemonic | Brief description |
|---|---|
| ADCS | Add with Carry |
| ADD{S} | Add |
| ADR | PC-relative Address to Register |
| ANDS | Bit wise AND |

**Table 4-3. Thumb instruction set** (continued)

| Mnemonic | Brief description |
|---|---|
| ASRS | Arithmetic Shift Right |
| B{cc} | Branch {conditionally} |
| BICS | Bit Clear |
| BKPT | Breakpoint |
| BL | Branch with Link |
| BLX | Branch indirect with Link |
| BX | Branch indirect |
| CMN | Compare Negative |
| CMP | Compare |
| CPSID | Change Processor State, Disable Interrupts |
| CPSIE | Change Processor State, Enable Interrupts |
| DMB | Data Memory Barrier |
| DSB | Data Synchronization Barrier |
| EORS | Exclusive OR |
| ISB | Instruction Synchronization Barrier |
| LDM | Load Multiple registers, increment after |
| LDR | Load Register from PC-relative address |
| LDRB | Load Register with word |
| LDRH | Load Register with half-word |
| LDRSB | Load Register with signed byte |
| LDRSH | Load Register with signed half-word |
| LSLS | Logical Shift Left |
| LSRS | Logical Shift Right |
| MOV{S} | Move |
| MRS | Move to general register from special register |
| MSR | Move to special register from general register |
| MULS | Multiply, 32-bit result |
| MVNS | Bit wise NOT |
| NOP | No Operation |
| ORRS | Logical OR |
| POP | Pop registers from stack |
| PUSH | Push registers onto stack |
| REV | Byte-Reverse word |
| REV16 | Byte-Reverse packed half-words |
| REVSH | Byte-Reverse signed half-word |
| RORS | Rotate Right |
| RSBS | Reverse Subtract |
| SBCS | Subtract with Carry |

**Table 4-3. Thumb instruction set** (continued)

| Mnemonic | Brief description |
|---|---|
| SEV | Send Event |
| STM | Store Multiple registers, increment after |
| STR | Store Register as word |
| STRB | Store Register as byte |
| STRH | Store Register as half-word |
| SUB{S} | Subtract |
| SVC | Supervisor Call |
| SXTB | Sign extend byte |
| SXTH | Sign extend half-word |
| TST | Logical AND based test |
| UXTB | Zero extend a byte |
| UXTH | Zero extend a half-word |
| WFE | Wait For Event |
| WFI | Wait For Interrupt |

#### 4.3.3.1 Address alignment

An aligned access is an operation where a word-aligned address is used for a word or multiple word access, or where a half-word-aligned address is used for a half-word access. Byte accesses are always aligned. No support is provided for unaligned accesses on the Cortex®-M0+ processor. Any attempt to perform an unaligned memory access operation result in a HardFault exception.

#### 4.3.3.2 Memory endianness

The EZ-PD™ PMG1-S3 MCU Arm® Cortex®-M0+ uses little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

### 4.3.4 Systick timer

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

### 4.3.5 Debug

EZ-PD™ PMG1-S3 MCU Arm® Cortex®-M0+ contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

# 5 Interrupts

The Arm® Cortex®-M0+ (CM0+) CPU in EZ-PD™ PMG1-S3 MCU supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or ISR being executed by the CPU. EZ-PD™ PMG1-S3 MCU provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

## 5.1 Features

EZ-PD™ PMG1-S3 MCU supports the following interrupt features:

- Supports 32 interrupts
- NVIC integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

## 5.2 How it works

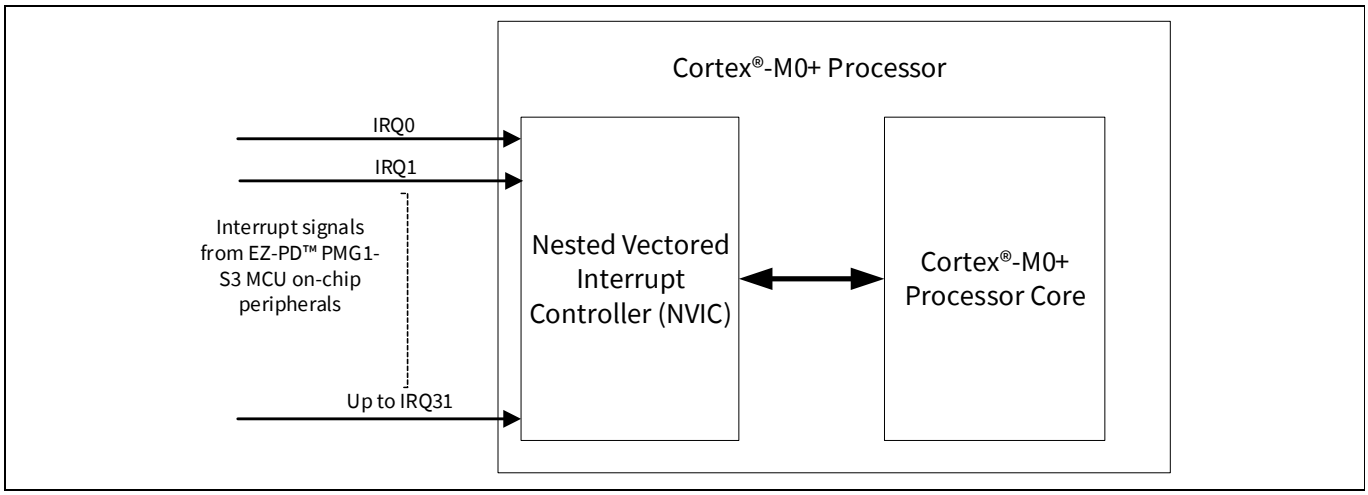



**Figure 5-1. Interrupts block diagram**

**Figure 5-1** shows the interaction between interrupt signals and the Cortex®-M0+ CPU. EZ-PD™ PMG1-S3 MCU supports 32 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in **Figure 5-1** because they are part of CM0+ core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

## 5.3 Interrupts and exceptions – operation

### 5.3.1 Interrupt and exception handling

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

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

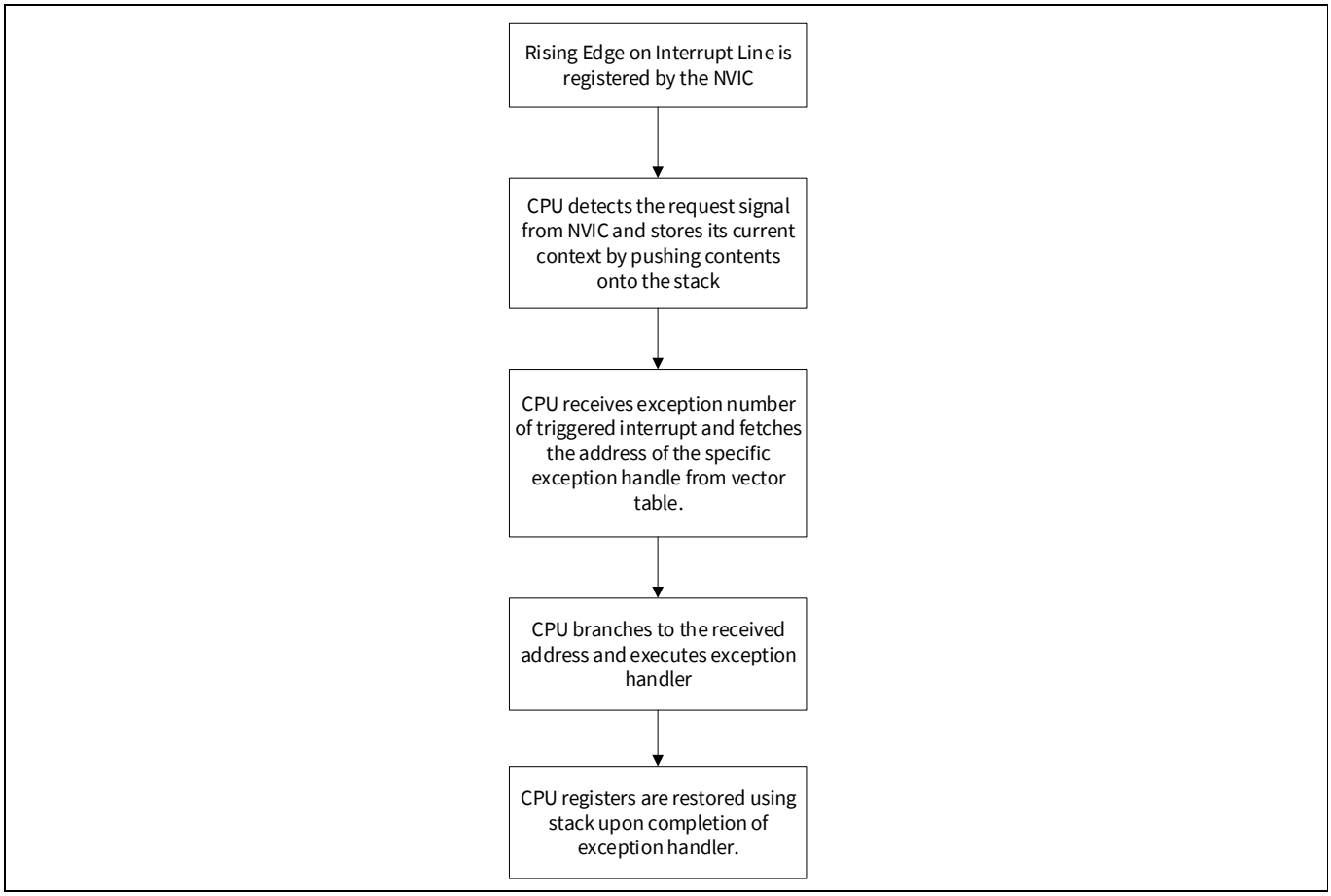



**Figure 5-2. Interrupt handling when triggered**

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

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

### 5.3.2 Level and pulse interrupts

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

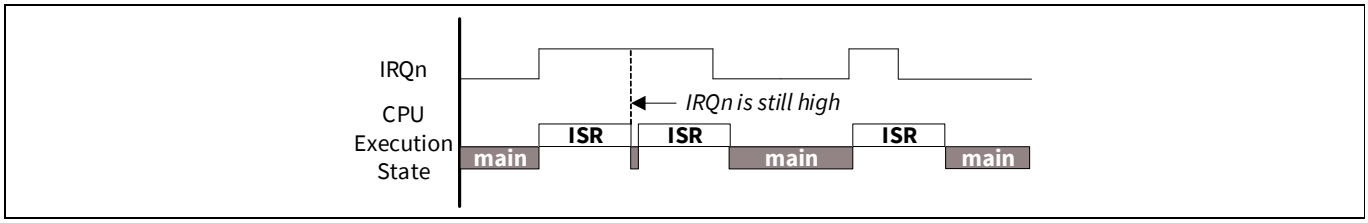



**Figure 5-3. Level interrupts**

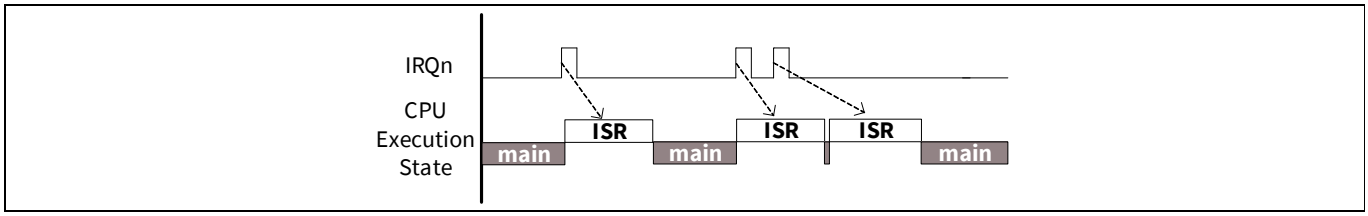



**Figure 5-4. Pulse interrupts**

Figure 5-3 and Figure 5-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

1. On a rising edge event of the interrupt signal, the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the interrupt requests have not yet been serviced by the CPU.
2. The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
3. When the ISR is being executed by the CPU, one or more rising edges of the interrupt signal are logged as a single pending request. The pending interrupt is serviced again after the current ISR execution is complete (see Figure 5-4 for pulse interrupts).
4. If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. Figure 5-3 illustrates this for level triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

### 5.3.3 Exception vector table

The exception vector table (see Table 5-1), stores the entry point addresses for all exception handlers in EZ-PD™ PMG1-S3 MCU. The CPU fetches the appropriate address based on the exception number.

**Table 5-1. EZ-PD™ PMG1-S3 MCU exception vector table**

| Exception number | Exception | Exception priority | Vector address |
|---|---|---|---|
| – | – | – | Base_Address – Can be 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM) |
| 1 | Reset | –3, the highest | Base_Address + 0x04 |
| 2 | Non Maskable | –2 | Base_Address + 0x08 |
| 3 | HardFault | –1 | Base_Address + 0x0C |
| 4–10 | Reserved | NA | Base_Address + 0x10 to Base_Address + 0x28 |
| 11 | Supervisory Call | Configurable (0–3) | Base_Address + 0x2C |

**Table 5-1. EZ-PD™ PMG1-S3 MCU exception vector table** (continued)

| Exception number | Exception | Exception priority | Vector address |
|---|---|---|---|
| 12–13 | Reserved | NA | Base_Address + 0x30 to Base_Address + 0x34 |
| 14 | PendSupervisory | Configurable (0–3) | Base_Address + 0x38 |
| 15 | System Timer (SysTick) | Configurable (0–3) | Base_Address + 0x3C |
| 16 | External Interrupt | Configurable (0–3) | Base_Address + 0x40 |
| … | … | … | … |
| 47 | External Interrupt | Configurable (0–3) | Base_Address + 0xBC |

In **Table 5-1**, the first word (4 bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. In EZ-PD™ PMG1-S3 MCU, the vector table can be configured to be located either in flash memory or SRAM. This configuration is done by writing to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. When the VECT_IN_RAM bit field is '1', CPU fetches exception handler addresses from the SRAM vector table location. When this bit field is '0' (reset state), the vector table in flash memory is used for exception address fetches. You must set the VECT_IN_RAM bit field as part of the device boot code to configure the vector table to be in SRAM. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

The exception sources (exception numbers 1 to 15) are explained in **Exception sources**. The exceptions marked as Reserved in **Table 5-1** are not used in EZ-PD™ PMG1-S3 MCU, though they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 43) are explained in **Interrupt sources**.

## 5.4 Exception sources

This section describes the different exception sources listed in **Table 5-1** (exception numbers 1 to 15).

### 5.4.1 Reset exception

Device reset is treated as an exception in EZ-PD™ PMG1-S3 MCU. It is always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of the SROM. The boot code and other data in SROM memory are programmed by Infineon, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in **Table 5-1**) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is deasserted.

### 5.4.2 Non-maskable interrupt (NMI) exception

Non-maskable interrupt (NMI) is the highest priority exception other than reset. It is always enabled with a fixed priority of –2. There are two ways to trigger an NMI exception in EZ-PD™ PMG1-S3 MCU:

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0P_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System call NMI exception:** This exception is used for nonvolatile programming operations in EZ-PD™ PMG1-S3 MCU such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

### 5.4.3 HardFault exception

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

### 5.4.4 Supervisor call (SVCall) exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0+ in EZ-PD™ PMG1-S3 MCU uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the **“Chip operational modes”** on page 94 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in EZ-PD™ PMG1-S3 MCU. The application developer must define the SVCall exception handler according to the end application requirements.

The priority of a SVCall exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_11[31:30] of the System Handler Priority Register 2 (CM0P_SHPR2). When the SVC instruction is executed, the SVCall exception enters the pending state and waits to be serviced by the CPU. The SVCALLPENDED bit in the System Handler Control and State Register (CM0P_SHCSR) can be used to check or modify the pending status of the SVCall exception.

### 5.4.5 PendSV exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, CM0P_ICSR. On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSV- CLR bit in the Interrupt Control State Register, CM0P_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_14[23:22] of the System Handler Priority Register 3 (CM0P_SHPR3). See the **Armv6-M architecture reference manual** for more details.

### 5.4.6 SysTick exception

CM0+ CPU in EZ-PD™ PMG1-S3 MCU supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various time keeping purposes such as an RTOS tick timer, high- speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick Exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0P_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_15[31:30] of the System Handler Priority Register 3 (CM0P_SHPR3). The SysTick exception can always be generated in software at any instant by writing a '1' to the PENDSTSETb bit in the Interrupt Control State Register, CM0P_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a '1' to the PENDST-CLR bit in the Interrupt Control State Register, CM0P_ICSR.

## 5.5 Interrupt sources

EZ-PD™ PMG1-S3 MCU supports 32 interrupts (IRQ0–IRQ31, or exception numbers 16-48) from peripherals. **Table 5-2** lists the source of each interrupt. EZ-PD™ PMG1-S3 MCU provides flexible sourcing options for each of the 32 interrupt lines. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM, serial communication block, and interrupts from GPIO ports. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. These interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously. See the **"I/O system"** on page 77 for details on GPIO interrupts.

**Table 5-2. List of EZ-PD™ PMG1-S3 MCU interrupt sources**

| Interrupt No. | Cortex®-M0+ exception No. | Interrupt source |
|---|---|---|
| NMI | 2 | |
| IRQ0 | 16 | GPIO Interrupt All Ports |
| IRQ1 | 17 | WDT or Temp (WDT only in DeepSleep) |
| IRQ2 | 18 | SCB0 (Serial Communication Block 0) |
| IRQ3 | 19 | SCB1 (Serial Communication Block 1) |
| IRQ4 | 20 | SCB2 (Serial Communication Block 2) |
| IRQ5 | 21 | SCB3 (Serial Communication Block 3) |
| IRQ6 | 22 | SCB4 (Serial Communication Block 4) |
| IRQ7 | 23 | SCB5 (Serial Communication Block 5) |
| IRQ8 | 24 | SCB6 (Serial Communication Block 6) |
| IRQ9 | 25 | SCB7 (Serial Communication Block 7) |
| IRQ10 | 26 | LPCOMP trigger interrupt |
| IRQ11 | 27 | CTBm Interrupt (all CTBms) |
| IRQ12 | 28 | Ganged USBPD[0] Interrupt |
| IRQ13 | 29 | Ganged USBPD[1] Interrupt |
| IRQ14 | 30 | CSD #0 (Primarily CAPSENSE™) |
| IRQ15 | 31 | SPC |
| IRQ16 | 32 | DMA interrupt |
| IRQ17 | 33 | Crypto Interrupt |

**Table 5-2. List of EZ-PD™ PMG1-S3 MCU interrupt sources** (continued)

| Interrupt No. | Cortex®-M0+ exception No. | Interrupt source |
|---|---|---|
| IRQ18 | 34 | 12-bit SAR ADC |
| IRQ19 | 35 | TCPWM Counter #0 |
| IRQ20 | 36 | TCPWM Counter #1 |
| IRQ21 | 37 | TCPWM Counter #2 |
| IRQ22 | 38 | TCPWM Counter #3 |
| IRQ23 | 39 | TCPWM Counter #4 |
| IRQ24 | 40 | TCPWM Counter #5 |
| IRQ25 | 41 | TCPWM Counter #6 |
| IRQ26 | 42 | TCPWM Counter #7 |
| IRQ27 | 43 | USB Start of Frame |
| IRQ28 | 44 | USB EP1-EP8 data |
| IRQ29 | 45 | USB EP1-EP8 data |
| IRQ30 | 46 | Synchronous USBPD[0] Interrupts |
| IRQ31 | 48 | Synchronous USBPD[1] Interrupts |

## 5.6 Exception priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. EZ-PD™ PMG1-S3 MCU provides flexibility in choosing priority values for different exceptions. All exceptions except Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In EZ-PD™ PMG1-S3 MCU, lower priority numbers represent higher priorities. This means that the Reset, NMI, and HardFault exceptions have the highest priorities. The other exceptions can be assigned a configurable priority level between 0 and 3.

EZ-PD™ PMG1-S3 MCU supports nested exceptions in which a higher priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as active exception. The CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0+ CPU in EZ-PD™ PMG1-S3 MCU allows nesting of up to four exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1 to 32 are explained in **"Exception sources"** on page 41. The priority of the 32 interrupts (IRQ0 to IRQ31) can be configured by writing to the Interrupt Priority registers (CM0P_IPR). This is a group of five 32-bit registers with each register storing the priority values of four interrupts, as given in **Table 5-3**. The other bit fields in the register are not used.

**Table 5-3. Interrupt priority register bit definitions**

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

## 5.7 Enabling/disabling interrupts

The NVIC provides registers to individually enable and disable the 32 interrupts (EZ-PD™ PMG1-S3 MCU) in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that interrupt line. The Interrupt Set-Enable Register (CM0P_ISER) and the Interrupt Clear-Enable Register (CM0P_ICER) are used to enable and disable the interrupts respectively. These registers are 32-bit wide and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. **Table 5-4** shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

**Table 5-4. Interrupt enable/disable registers**

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

## 5.8 Exception states

Each exception can be in one of the following states.

**Table 5-5. Exception states**

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

CM0P_ISER and CM0P_ICER registers are applicable only for the interrupts (IRQ0–IRQ31). These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in **"Exception sources"** on page 41.

The PRIMASK register in Cortex®-M0+ (CM0+) CPU can be used as a global exception enable register to mask all the configurable priority exceptions irrespective of whether they are enabled. Configurable priority exceptions include all the exceptions except Reset, NMI, and HardFault listed in **Table 5-1**. They can be configured to a priority level between 0 and 3, 0 being the highest priority and 3 being the lowest priority. When the PM bit (bit 0) in PRIMASK register is set, none of the configurable priority exceptions can be serviced by the CPU, though they can be in the pending state waiting to be serviced by the CPU after the PM bit is cleared.

The Interrupt Control State Register (CM0P_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0P_ICSR store the exception number for the current executing exception. This value is zero if the CPU is not executing any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Control State Register (ICSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0P_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0P_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ18) is in a pending state.

### 5.8.1 Pending exceptions

When a peripheral generates an interrupt request signal to the NVIC or an exception event occurs, the corresponding exception enters the pending state. When the CPU starts executing the corresponding exception handler routine, the exception is changed from the pending state to the active state.

The NVIC allows software pending of the 19 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending Register (CM0P_ISPR) and the Interrupt Clear-Pending Register (CM0P_ICPR) are used to set and clear the pending status of the interrupt lines. These registers are 32 bits wide, and each bit corresponds to the same numbered interrupt. **Table 5-6** shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

**Table 5-6. Interrupt set pending/clear pending registers**

| Register | Operation | Bit value | Comment |
|---|---|---|---|
| Interrupt Set-Pending Register (CM0P_ISPR) | Write | 1 | To put an interrupt to pending state |
| | | 0 | No effect |
| | Read | 1 | Interrupt is pending |
| | | 0 | Interrupt is not pending |
| Interrupt Clear-Pending Register (CM0P_ICPR) | Write | 1 | To clear a pending interrupt |
| | | 0 | No effect |
| | Read | 1 | Interrupt is pending |
| | | 0 | Interrupt is not pending |

Setting the pending bit when the same bit is already set results in only one execution of the ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0P_ISER register.

Note that the CM0P_ISPR and CM0P_ICPR registers are used only for the 19 peripheral interrupts (exception numbers 16–34). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in **“Exception sources”** on page 41.

## 5.9 Stack usage for exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its general-purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the Program and Status Register (PSR), Return Address, Link Register (LR or R14), R12, R3, R2, R1, and R0. Cortex®-M0+ has two stack pointers – MSP and PSP. Only one of the stack pointers can be active at a time. When in thread mode, the Active Stack Pointer bit in the Control register is used to define the current active stack pointer. When in handler mode, the MSP is always used as the stack pointer. The stack pointer in Cortex®-M0+ always grows downwards and points to the address that has the last pushed data.

When the CPU is in thread mode and an exception request comes, the CPU uses the stack pointer defined in the control register to store the general-purpose register contents. After the stack push operations, the CPU enters handler mode to execute the exception handler. When another higher priority exception occurs while executing the current exception, the MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the **"Cortex®-M0+ CPU"** on page 32 for details.

The Cortex®-M0+ uses two techniques, tail chaining and late arrival, to reduce latency in servicing exceptions. These techniques are not visible to the external user and are done as part of the internal processor architecture (**infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0419c/index.html**).

## 5.10 Interrupts and low-power modes

EZ-PD™ PMG1-S3 MCU allows device wakeup from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the device to enter Active mode when one or more wakeup sources generate an interrupt signal. After entering Active mode, the ISR of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0+ CPU, triggers the transition into Sleep and Deep-Sleep modes. The sequence of entering the different low-power modes is detailed in the **"Power modes"** on page 95. The device low-power modes have two categories of fixed- function interrupt sources:

- Fixed-function interrupt sources that are available in the Active and Deep-Sleep modes (watchdog timer interrupt, $I^2C$ interrupts, USB PD wakeup and GPIO interrupts)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

## 5.11 Exception – initialization and configuration

This section covers the steps involved in initializing and configuring exceptions in EZ-PD™ PMG1-S3 MCU.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required – either in flash memory or SRAM. This configuration is done by writing either a '1' (SRAM vector table) or '0' (flash vector table) to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. This register write is done as part of the device initialization code.
   It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.
   a) Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
   b) Define the exception handler function and write the address of the function to the exception vector table. **Table 5-1** gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
   c) Set up the exception priority, as explained in **"Exception priority"** on page 44.

d) Enable the exception, as explained in **"Enabling/disabling interrupts"** on page 45.

## 5.12 Registers

**Table 5-7. Interrupts registers**

| Register name | Description |
|---|---|
| CM0P_ISER | Interrupt Set-Enable Register |
| CM0P_ICER | Interrupt Clear Enable Register |
| CM0P_ISPR | Interrupt Set-Pending Register |
| CM0P_ICPR | Interrupt Clear-Pending Register |
| CM0P_IPR | Interrupt Priority Registers |
| CM0P_ICSR | Interrupt Control State Register |
| CM0P_AIRCR | Application Interrupt and Reset Control Register |
| CM0P_SCR | System Control Register |
| CM0P_CCR | Configuration and Control Register |
| CM0P_SHPR2 | System Handler Priority Register 2 |
| CM0P_SHPR3 | System Handler Priority Register 3 |
| CM0P_SHCSR | System Handler Control and State Register |
| CM0P_SYST_CSR | Systick Control and Status |
| CPUSS_CONFIG | CPU Subsystem Configuration |
| CPUSS_SYSREQ | System Request Register |

## 5.13 Associated documents

- **Armv6-M architecture reference manual** – This document describes the Arm® Cortex®-M0+ architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

# 6 DMA controller modes

The DMA controller provides DataWire (DW) and Direct Memory Access (DMA) functionality. The DMA controller has the following features:

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

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

The operation modes are:

- Mode 0: Single data element per trigger
- Mode 1: All data elements per trigger
- Mode 2: All data elements per trigger and automatically trigger chained descriptor

The data transfer specifics, such as source and destination address locations and the size of the transfer, are specified by a descriptor structure. Each channel has an independent descriptor structure.

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

## 6.1 Block diagram description

The DMA transfers data to and from memory, peripherals, and registers. These transfers occur independent of the CPU. The DMA can transfer up to 65,536 data elements in one transfer. These data elements can be 8-bit, 16-bit, or 32-bit wide. The DMA starts each transaction through an external trigger that can come from a DMA channel (including itself), another DMA channel, a peripheral, or the CPU. The DMA is best used to offload data transfer tasks from the CPU.

**Figure 6-1** gives an overview of the DMA controller at a block level.

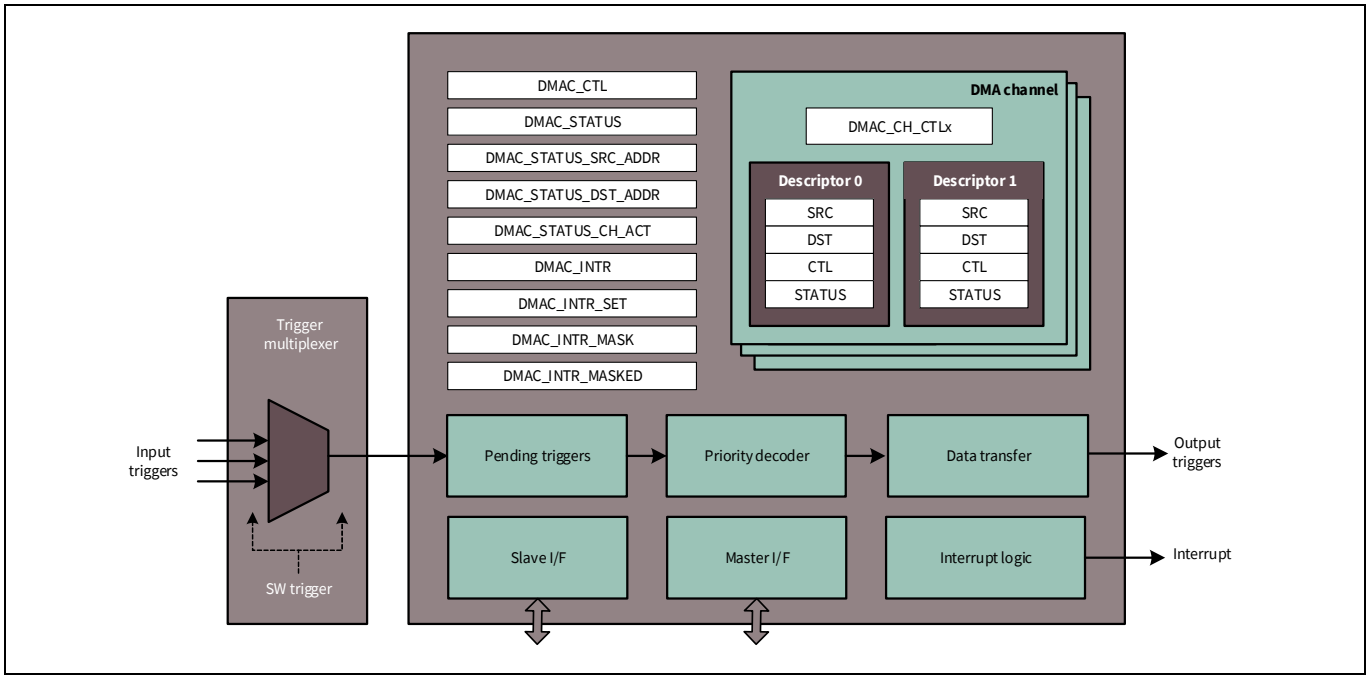



**Figure 6-1. DMA block diagram**

Every DMA channel has two descriptors, which are responsible for configuring parameters specific to the transfer, such as source address, destination address, and data width. The transfer initiation in the DMA channel is on a trigger event. The trigger signals can come from different peripherals in the device, including the DMA itself. The DMA controller has two bus interfaces, the master interface and the slave interface. Master I/F is an AHB-Lite bus master, which allows the DMA controller to initiate AHB-Lite data transfers to the source and destination locations. The DMA is the bus master in the master interface. This is the interface through which all DMA transfers are accomplished.

The DMA configuration registers and descriptors are accessed and reconfigured through the slave interface. Slave I/F is an AHB-Lite bus slave, which allows the EZ-PD™ PMG1-S3 MCU CPU to access the DMA controller's control/status registers and to access the descriptor structure. CPU is generally the master for this bus.

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

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

### 6.1.1 Trigger sources and multiplexing

Every DMA channel has an input and output trigger associated with it. The input trigger can come from any peripheral, CPU, or a DMA channel itself. The input trigger is used to trigger a DMA transfer, as defined by the **"Transfer mode"** on page 55. A 'logic high', on the trigger input will trigger the DMA channel. The minimum width of this 'logic high' is two system clock cycles. The deactivation setting configures the nature of trigger deactivation. The output trigger signals the completion of a transfer. This signal can be used as a trigger to a DMA channel or as a digital signal to the digital interconnect. The trigger input can come from different sources and is routed through a **"Trigger multiplexer block"** on page 105.

### 6.1.2 Pending triggers

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

The pending triggers are registered in the status register (DMAC_STATUS_CH_ACT).

### 6.1.3 Output triggers

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

### 6.1.4 Channel prioritization

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

### 6.1.5 Data transfer engine

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

- State 0 - Default State: This is the idle state of the DMA controller, where it waits for a trigger condition to initiate transfer.
- State 1 - Load Descriptor: When a trigger condition is encountered and priority is resolved, the data transfer engine enters the load descriptor state. In this state, the active descriptor (SRC, DST, and CTL) is loaded into the DMA controller to initiate the transfer. The DMAC_STATUS, DMAC_STATUS_SRC_ADDR and DMAC_STATUS_DST_ADDR, and STATUS_CH_ACT will also reflect the currently active status.
- State 2 - Loading data from source: The data transfer engine uses the master I/F to load data from the source location.
- State 3 - Storing data at destination: The data transfer engine uses the master I/F to store data to the destination location. Depending on the Transfer mode, State 2 and 3 may be performed multiple times.

- State 4 - Storing Descriptor: The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor.
- State 5 - Wait for Trigger Deactivation: If the trigger deactivation condition is specified as two cycles, this condition is met after two cycles of the trigger activation. If it was set to ‘wait indefinitely’, the DMA controller will remain in this state until the trigger signal has gone low.
- State 6 - Storing Descriptor Response: In this phase, the data transfer according to the descriptor is completed and an interrupt may be generated if it was configured to do so. The Response field in DMAC_DESCR_PING_STATUS or DMAC_DESCR_PONG_STATUS is also populated and the state transitions to State 0.

## 6.2 Descriptors

The data transfer between a source and a destination in a channel is configured using a descriptor. Each channel in the DMA has two descriptors named PING and PONG descriptors (also called Descriptor 0 and Descriptor 1 in this document). A descriptor is a set of four 32-bit registers that contain the configuration for the transfer in the associated channel. **Figure 6-2** shows the structure of a descriptor.

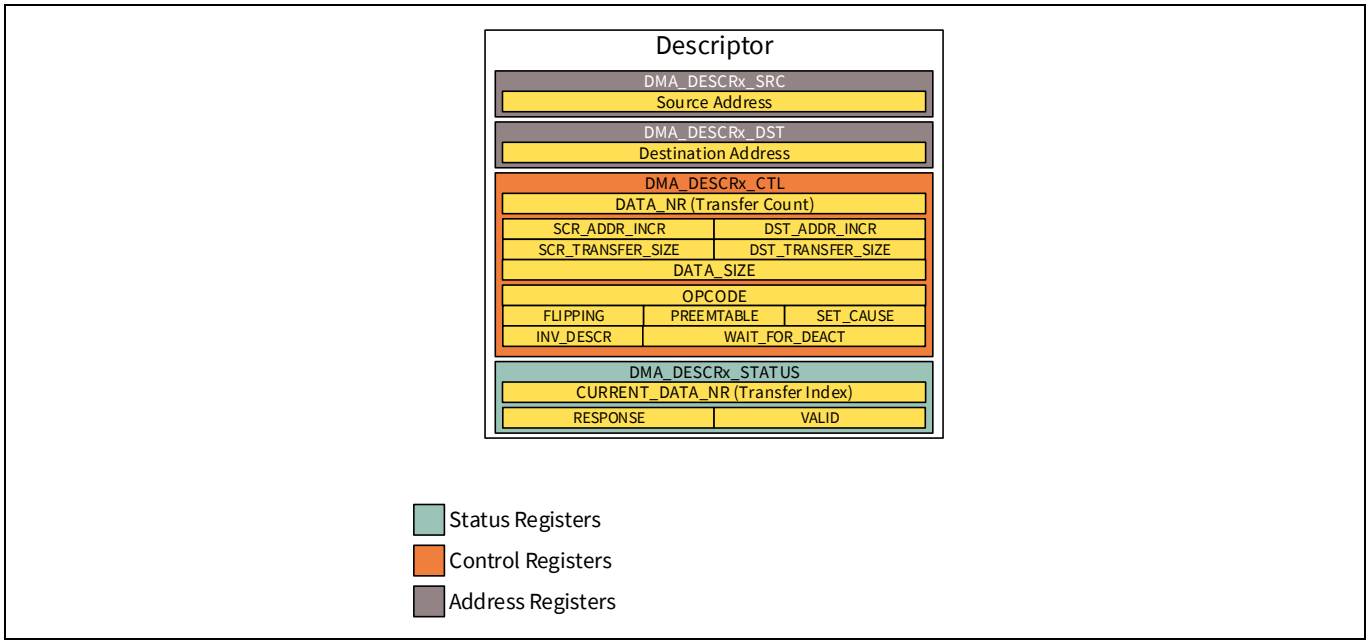



**Figure 6-2. Descriptor structure**

### 6.2.1 Address configuration

**Figure 6-3** demonstrates the use of the descriptor settings for the address configuration of a transfer.

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

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

### DMA controller modes

**Source address increment (SCR_ADDR_INC):** This is a bit setting in the control register, which determines if a source address is incremented between each data element transfer. This feature is enabled when the source of the data is a buffer and each transfer element needs to be fetched from subsequent locations in the memory. In this case, the Source Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the SCR_TRANSFER_SIZE setting described in **"Transfer size"** on page 54.

**Destination address increment (DST_ADDR_INC):** This is a bit setting in the control register, which determines if a destination address is incremented between each element transfer. This feature is enabled when the destination of the data is a buffer and each transfer element needs to be transferred to subsequent locations in the memory. In this case, the Destination Address register sets only the base address and subsequent transfers are incremental on this. The size of address increments are determined based on the DST_TRANSFER_SIZE setting described in **"Transfer size"** on page 54.

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

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

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

**Trigger type (WAIT_FOR_DEACT):** When the DMA transfer based on the descriptor is completed, the data transfer engine checks the state of trigger deactivation. This is corresponding to State 5 of the data transfer engine. See **"Data transfer engine"** on page 51. The type of DMA input trigger will determine when the trigger signal is considered deactivated. The DMA transfer is activated when the trigger is activated, but the transfer is not considered complete until the trigger state is deactivated. This field is used to synchronize the controller's data transfer(s) with the agent that generated the trigger.

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

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

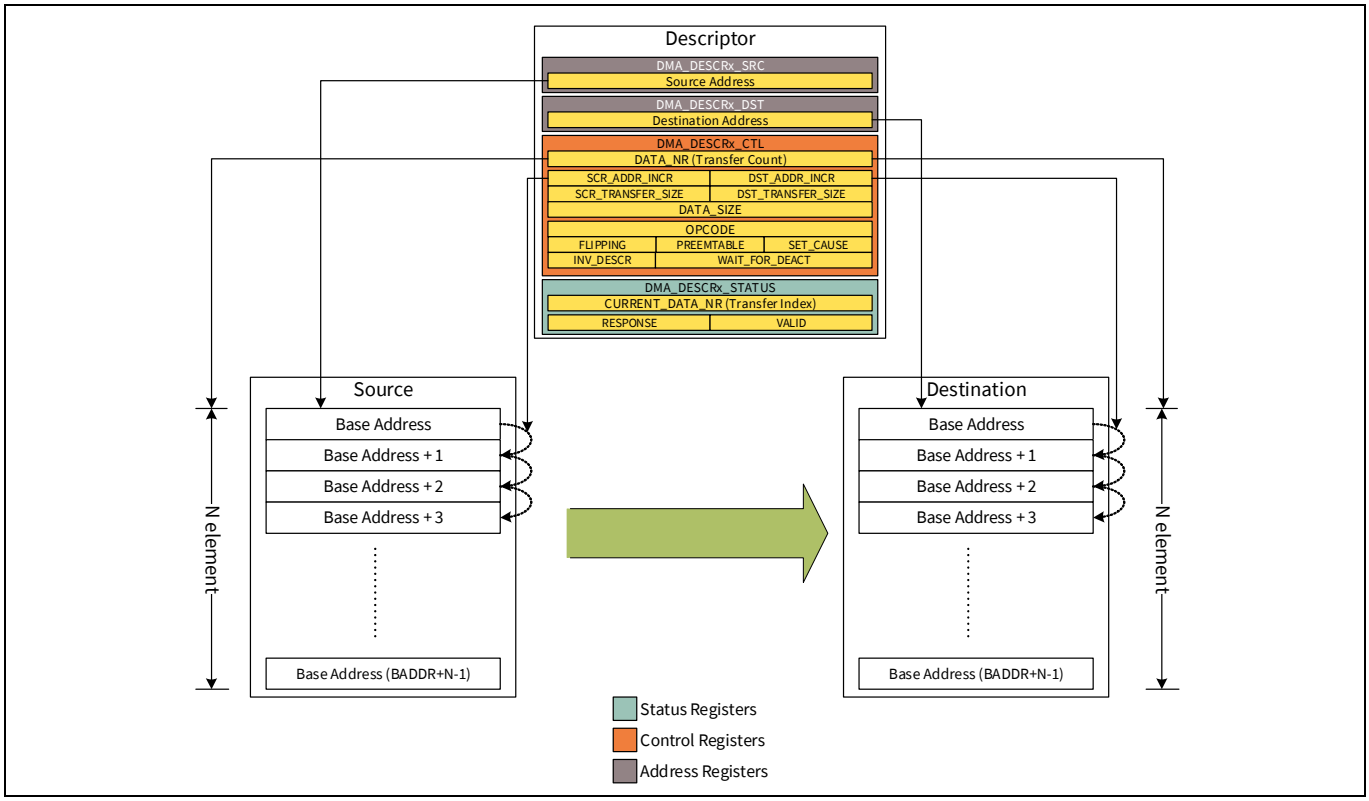



**Figure 6-3. DMA transfer: Address configuration**

### 6.2.2 Transfer size

The transfer word width for a transfer can be configured using the transfer/data size parameter in the descriptor. The settings are diversified into source transfer size, destination transfer size, and data size. The data size parameter (DATA_SIZE) sets the width of the bus for the transfer. The source and destination transfer sizes, set by SCR_TRANSFER_SIZE and DST_TRANSFER_SIZE, can have a value of either the DATA_SIZE or 32 bit. DATA_SIZE can be set to a 32-bit, 16-bit, or 8-bit setting.

The data width of most EZ-PD™ PMG1-S3 MCU peripheral registers is 4 bytes (32 bit); therefore, SCR_TRANSFER_SIZE or DST_TRANSFER_SIZE should typically be set to 32 bit when DMA is using a peripheral as its source or destination.

The source and destination transfer size for the DMA component must match the addressable width of the source and destination, regardless of the width of data that needs to be moved. The DATA_SIZE parameter will correspond to the width of the actual data. For example, if a 16-bit PWM is used as a destination for DMA data, the DST_TRANSFER_SIZE must be set to 32 bit to match the width of the PWM register, because the peripheral register width for the TCPWM block (and most EZ-PD™ PMG1-S3 MCU peripherals) is always 32 bit. However, in this example the DATA_SIZE for the destination may still be set to 16 bit because the 16-bit PWM only uses 2 bytes of data. SRAM and flash are 8-bit, 16-bit, or 32-bit addressable and can use any source and destination transfer sizes to match the needs of the application.

**Table 6-1** summarizes the possible combinations of the transfer size settings and its description.

**Table 6-1. Transfer size settings and descriptions**

| DATA_SIZE | SCR_TRANSFER_SIZE | DST_TRANSFER_SIZE | Typical usage | Description |
|---|---|---|---|---|
| 8-bit | 8-bit | 8-bit | Memory to Memory | No data manipulation |
| 8-bit | 32-bit | 8-bit | Peripheral to Memory | Higher 24 bits from the source dropped |

**Table 6-1. Transfer size settings and descriptions** (continued)

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

### 6.2.3 Descriptor chaining

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

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

### 6.2.4 Transfer mode

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

#### 6.2.4.1 Single data element per trigger (OPCODE 0)

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

**Figure 6-4** shows a typical use case of this transfer. Here a UART receive (RX) register is the source and the destination is a peripheral register such as an SPI transmit (TX) register. The trigger is from the DMA request signal of the UART. When the trigger is received, the transfer engine will load data from the UART Rx register and store the lower eight bits to the SPI TX register. Successive triggers will result in the same behavior because the descriptor will be rerun.

### DMA controller modes

Note how the source and destination data widths are assigned as 32 bit. This is because all accesses to peripheral registers in EZ-PD™ PMG1-S3 MCU must be 32 bit. Because the valid data width is only eight bits, the DATA_SIZE is maintained as eight bit.

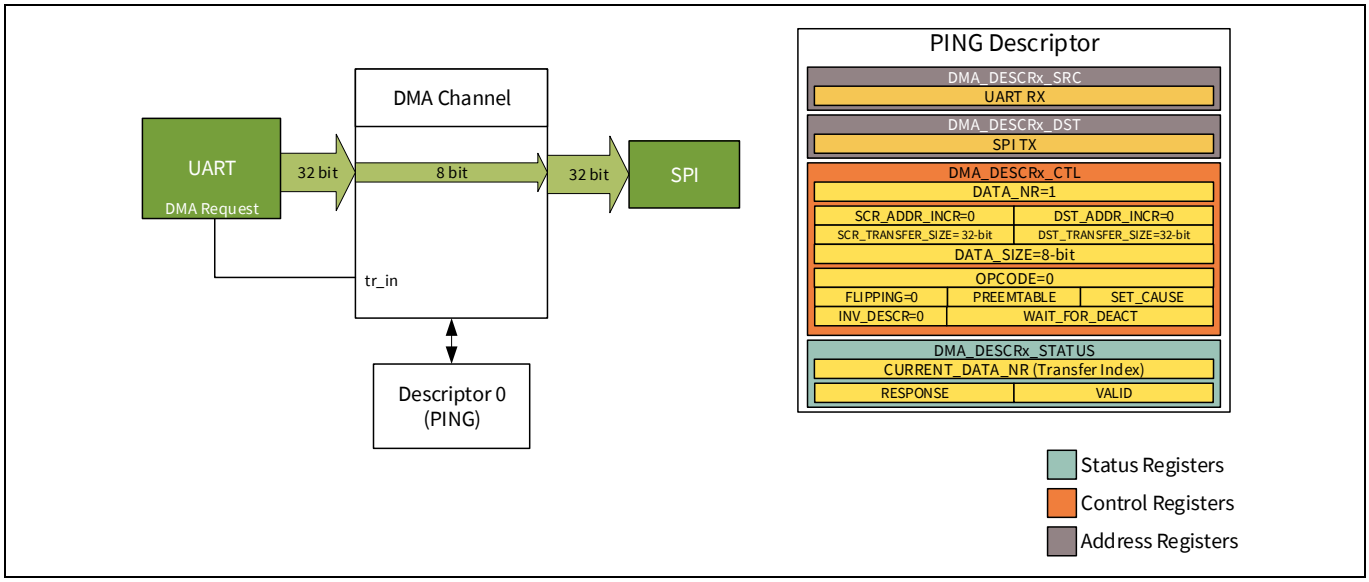



**Figure 6-4. OPCODE 0: Simple DMA transfer from peripheral to peripheral**

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

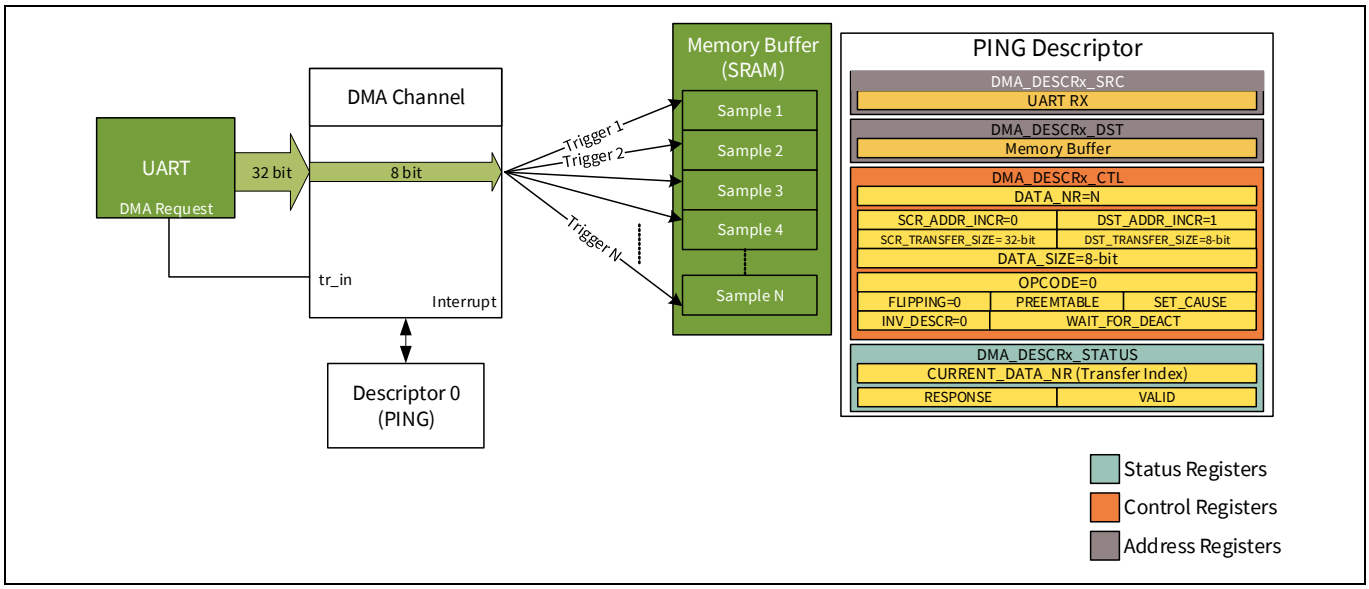



**Figure 6-5. OPCODE 0: Transfer with destination address increment feature**

### DMA controller modes

A similar use case is shown in **Figure 6-6**. This demonstrates the use of the PING and PONG descriptors. On completion of the PING descriptor, the controller will flip to execute the PONG descriptor. Thus, two buffer transfers are achieved in sequence. However, note that the transfers are still done at one element transfer for every trigger.

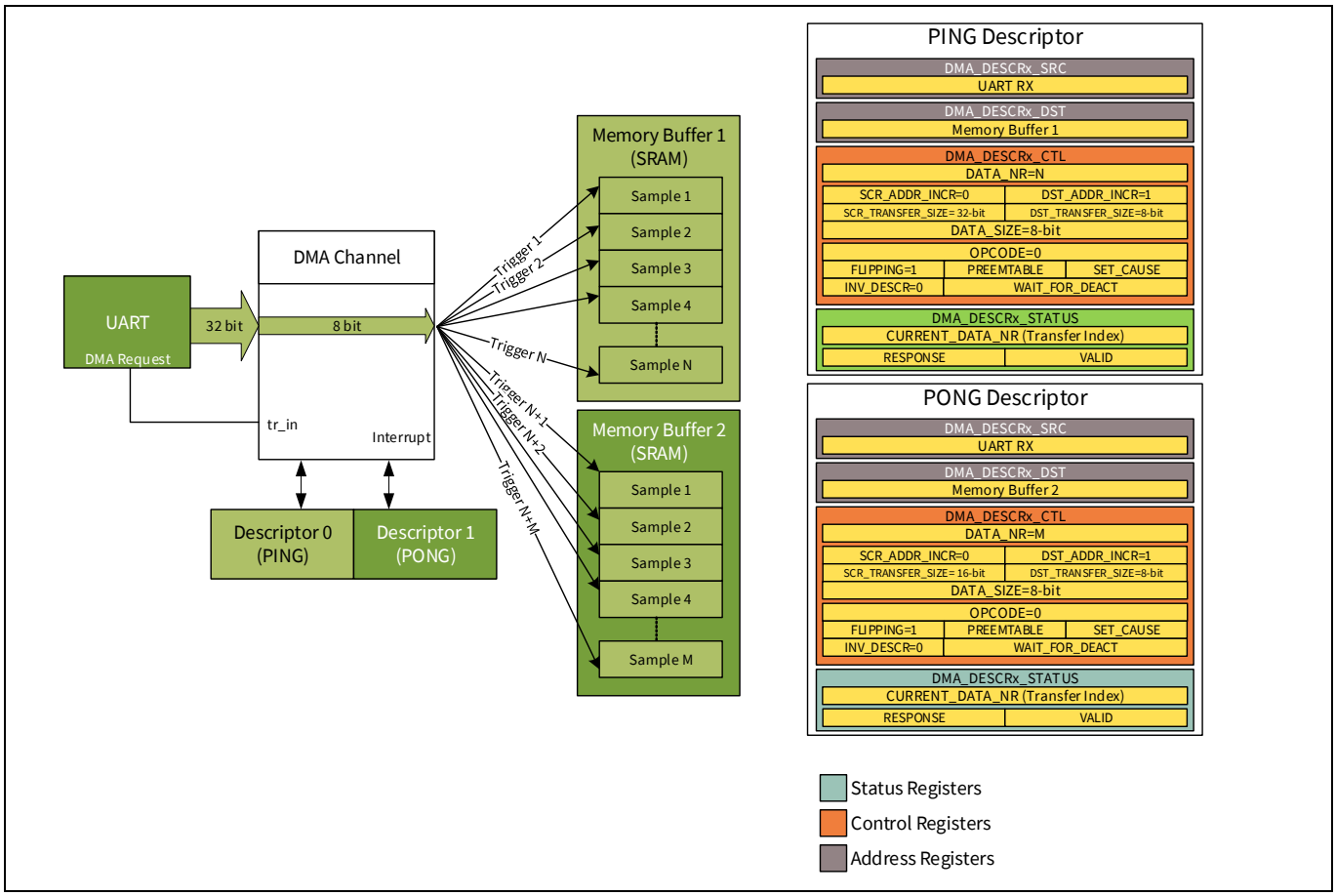



**Figure 6-6. DMA transfer using flipping feature**

### 6.2.4.2 Entire descriptor per trigger (OPCODE 1)

In this mode of operation, the DMA transfers multiple data elements from a source location to a destination location in one trigger. In OPCODE 1, the controller executes the entire descriptor in a single trigger. This type of functionality is useful in memory-to-memory buffer transfers. When the trigger condition is encountered, the transfer is continued until the descriptor is completed.

**Figure 6-7** shows an OPCODE 1 transfer, which transfers the entire contents of the source buffer into the destination buffer. The entire transfer is part of a single PING descriptor and is completed on a single trigger.

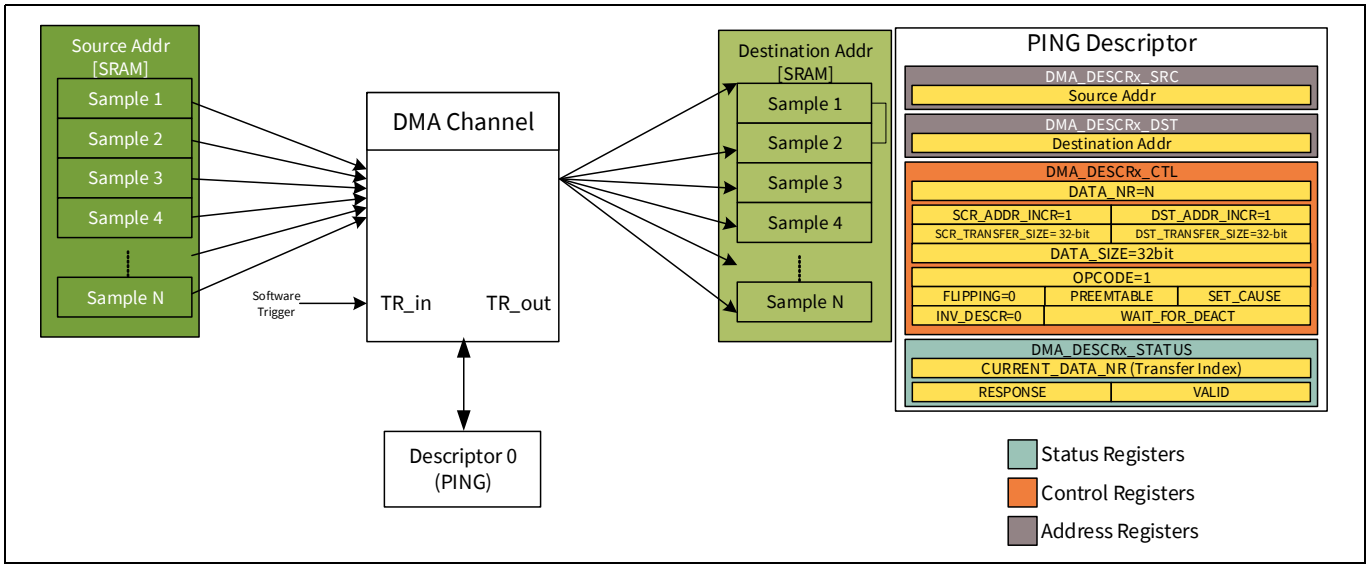



**Figure 6-7. DMA transfer example with OPCODE 1**

### 6.2.4.3 Entire descriptor chain per trigger (OPCODE 2)

OPCODE 2 is always used in conjunction with the FLIPPING field. When OPCODE 2 is used with FLIPPING enabled in a PING descriptor, a single trigger can execute a PING descriptor and automatically flip to the PONG descriptor and execute that too. If the PONG descriptor is also provided with an OPCODE 2, then the cycling between PING and PONG will continue until one of the descriptors are invalidated or changed by the CPU. **Figure 6-8** shows a case where the PING and PONG descriptors are configured for OPCODE 2 operation and on the second iteration of the PING register, FLIPPING is disabled by the CPU.

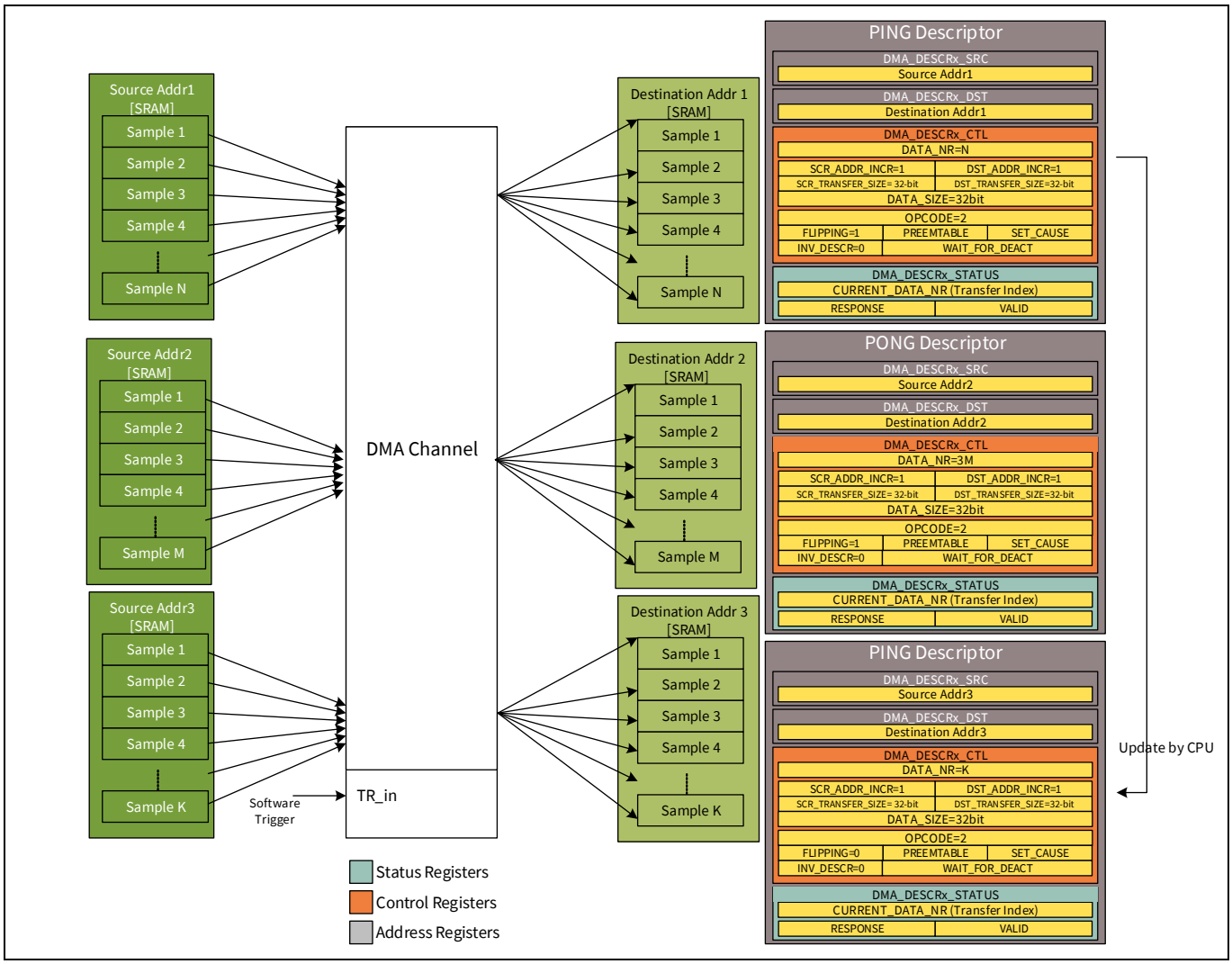



**Figure 6-8. DMA transfer example with OPCODE 2**

The OPCODE 2 transfer mode can be customized to implement distinct use cases. **Figure 6-9** illustrates one such use case. Here, the source data can come from two different locations which are not consecutive memory. The destination is a data structure that is in consecutive memory locations. One source is the Timer 2, which holds a timing data and the other source is a PWM compare register. Both the data is stored in consecutive locations in memory.

## DMA controller modes

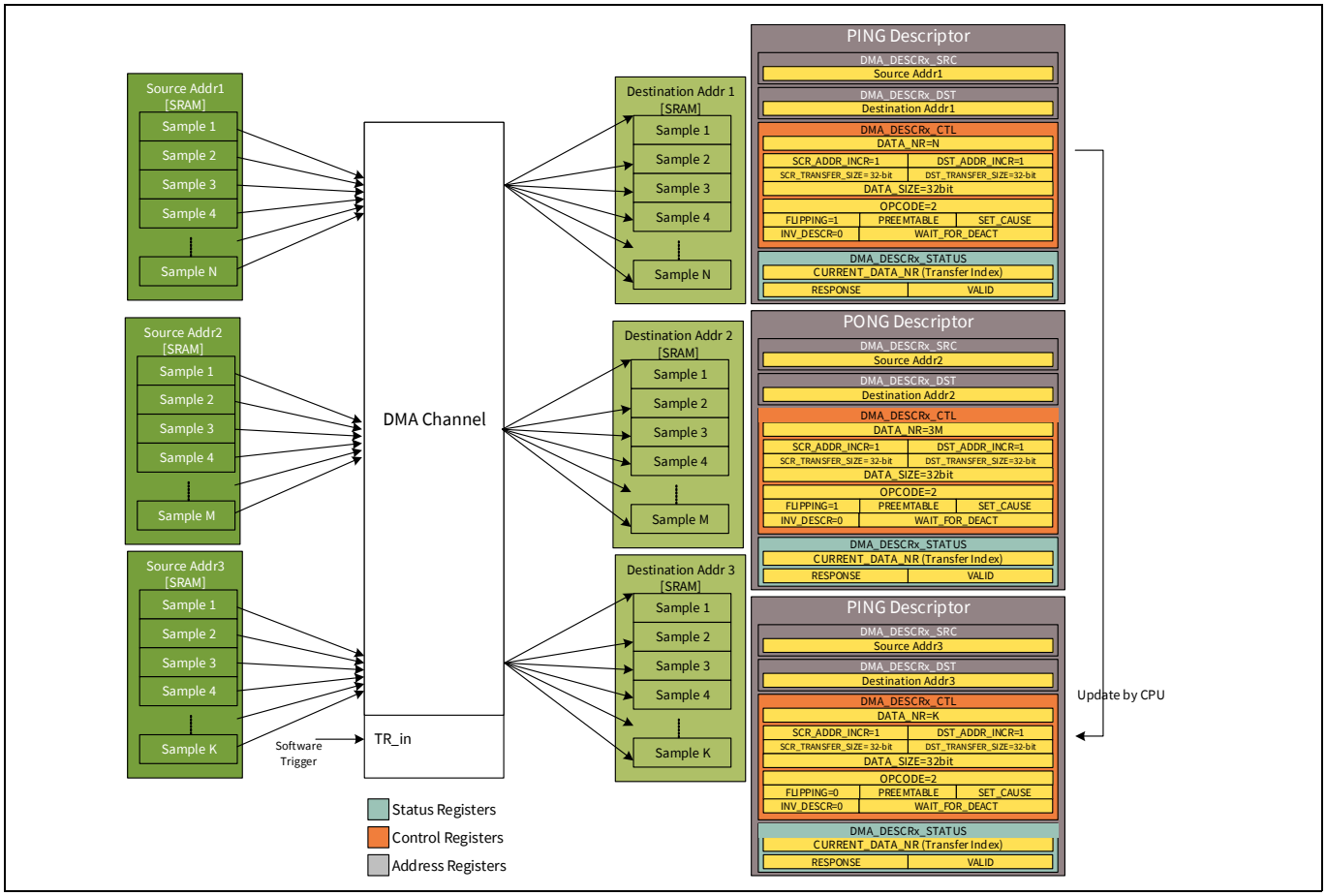



**Figure 6-9. OPCODE 2: Multiple sources to memory**

## 6.3 Operation and timing

**Figure 6-10** shows the DMA controller design with a trigger, data, or interrupt flow superimposed on it.

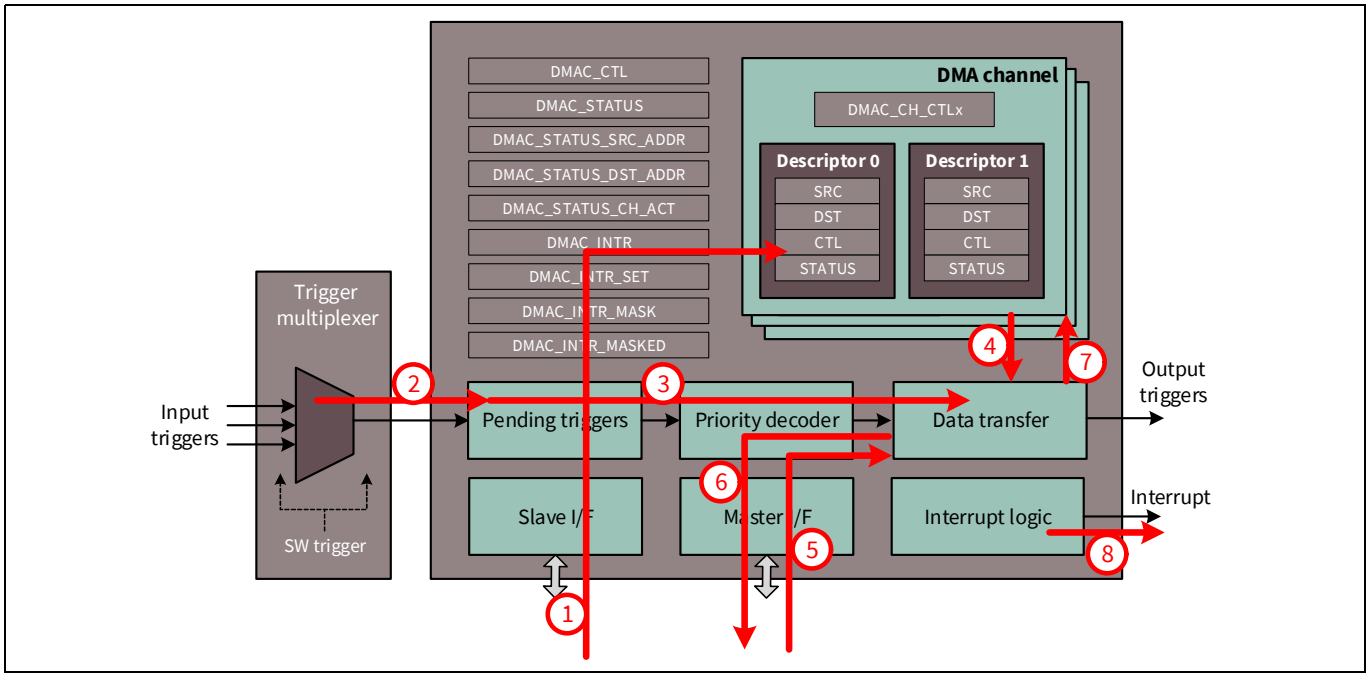



**Figure 6-10. Operational flow**

The flow exemplifies the steps that are involved in a DMA controller data transfer:

1. The main CPU programs the descriptor structure for a specific channel. It also programs the DMA register that selects a specific system trigger for the channel.
2. The channel's system trigger is activated.
3. Priority decoding determines the highest priority activated channel.
4. The data transfer engine accepts the activated channel and uses the channel identifier to load the channel's descriptor structure. The descriptor structure specifies the channel's data transfers.
5. The data transfer engine uses the master I/F to load data from the source location.
6. The data transfer engine uses the master I/F to store data to the destination location. In a single element (opcode 0) transfer, steps 5 and 6 are performed once.

In a multiple element descriptor (opcode 1 or 2) transfer, steps 5 and 6 may be performed multiple times in sequence to implement multiple data element transfers.

1. The data transfer engine updates the channel's descriptor structure to reflect the data transfer and stores it in the descriptor SRAM.
2. If all the data transfers as specified by a descriptor channel structure have completed, an interrupt may be generated (this is a programmable option).

The DMA controller data transfer steps can be classified as either: initialization, concurrent, or sequential steps:

- Initialization: This includes step 1, which programs the descriptor structures. This step is done for each descriptor structure. It is performed by the main CPU and is NOT initiated by an activated channel trigger.
- Concurrent: This includes steps 2 and 3. These steps are performed in parallel for each channel.
- Sequential: This includes steps 4 through 8. These steps are performed sequentially for each activated channel. As a result, the DMA controller throughput is determined by the time it takes to perform these steps. This time consists of two parts: the time spent by the controller (to load and store the descriptor) and the time spent on the bus infrastructure. The latter time is dependent on the latency of the bus (determined by arbiter and bridge components) and the target memories/peripherals.

### DMA controller modes

When transferring single data elements, it takes 12 clock cycles to complete one full transfer under the assumption of no wait states on the AHB-Lite bus. The equation for number of cycles to complete a transfer in this mode is:

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

No of cycles = (12 + LOAD wait states + STORE wait states) + (N–1)*(3 + LOAD wait states + STORE wait states)

## 6.4 Arbitration

The AHB bus of the device has two masters: the CPU and the DMA controller. All peripherals and memory connect to the bus through slave interfaces. There are dedicated slave interfaces for flash memory and RAM with their own arbiters.

The peripheral registers all connect to a single slave interface through a bridge into a dedicated arbiter. The DMA controller's slave interface, which is used to access the DMA controller's control registers, all connect through another slave interface. **Figure 6-11** illustrates this architecture.

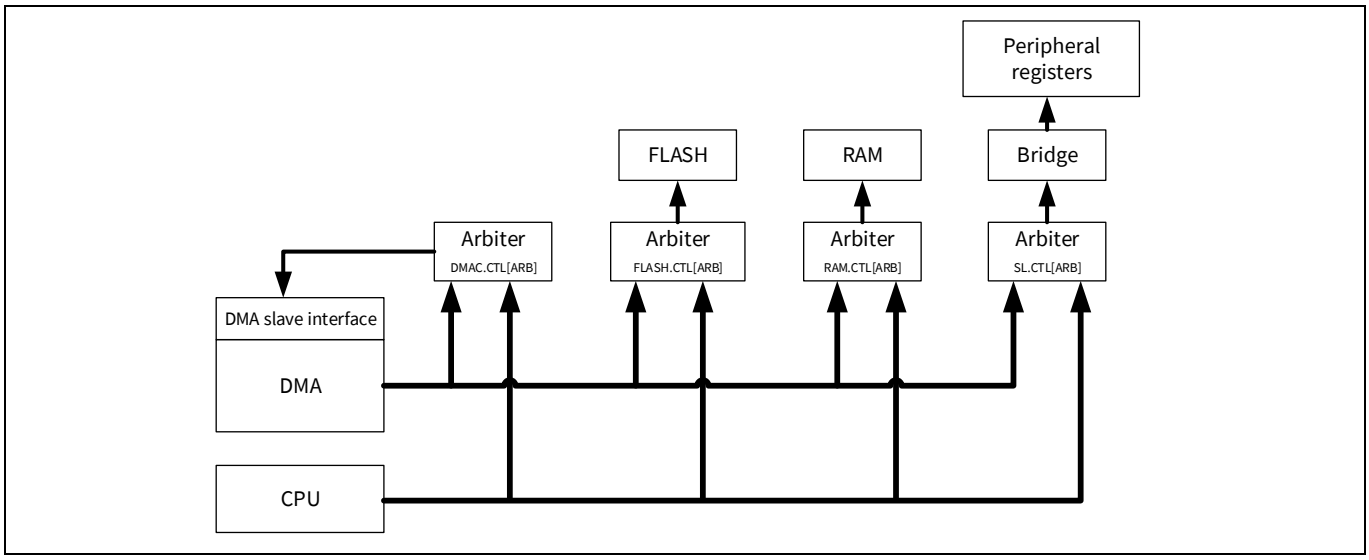



**Figure 6-11. EZ-PD™ PMG1-S3 MCU bus architecture**

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

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

## DMA controller modes

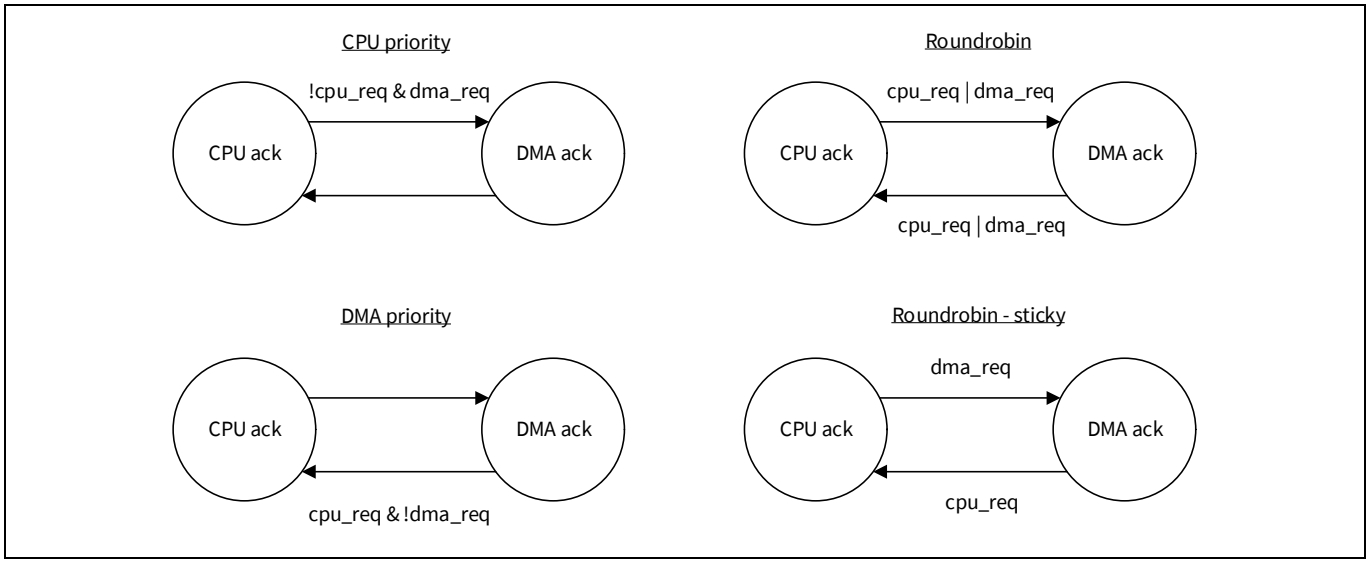



**Figure 6-12. Arbitration models**

## 6.5 Registers

**Table 6-2. DMA registers**

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

# 7 Cryptography

EZ-PD™ PMG1-S3 MCU Cryptography accelerator has following features:

- AES-128 component to accelerate block cipher functionality. This component supports forward encryption of a single 128-bit block with a 128-bit key.
- SHA-256 component to accelerate hash functionality. This component supports message schedule calculation for a 512-bit message chunk and processing of a 512-bit message chunk.
- Vector unit (VU) component to accelerate asymmetric key cryptography (e.g. RSA and ECC). This component supports large integer multiplication, addition, etc.
- True Random Number Generator (TRNG) component based on a set of ring oscillators. The TRNG includes a HW health monitor.

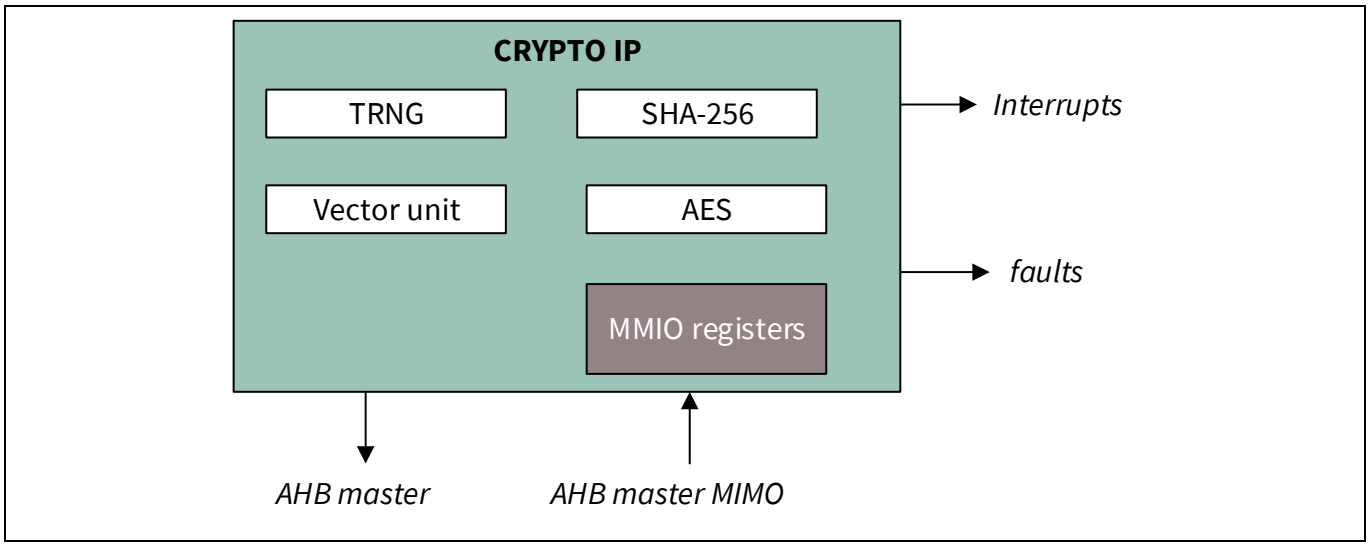



**Figure 7-1. Cryptography block diagram**

## 7.1 Block diagram

This diagram shows the major components of Cryptography accelerator in EZ-PD™ PMG1-S3 MCU.

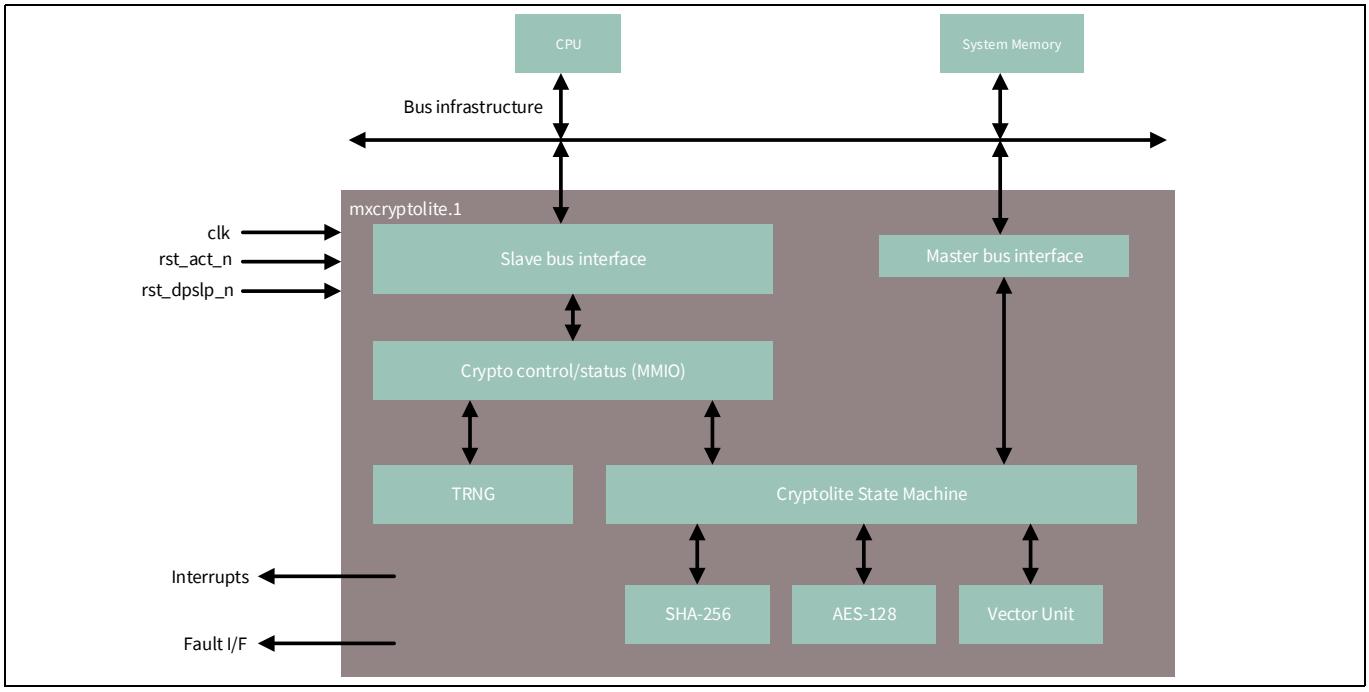



**Figure 7-2. Block diagram**

## 7.2 Functional description

### 7.2.1 AES-128

The AES-128 block cipher function is a member of the AES block cipher family. It encrypts a 128-bit block with a 128-bit key.

A block cipher mode of operation uses the AES-128 block cipher function in a specific configuration to provide message confidentiality and authenticity. The CRYPTO component provides AES-128 block cipher function HW acceleration. A specific block cipher mode of operation is implemented using SW and HW acceleration. Typically, a message is broken up into multiple 128-bit blocks.

The AES-128 block cipher function performs 10 rounds. Each round updates the internal cipher state. Each round has a dedicated 128-bit round key, which is derived from the provided 128-bit key.

The EZ-PD™ PMG1-S3 MCU AES-128 block cipher function HW acceleration consists of a single HW function:

- Encrypt function. Encrypt a 128-bit block P (plaintext) with a 128-bit secret key SK into a 128-bit block C (ciphertext).

A write to the AES_DESCR MMIO register starts a 128-bit AES encryption in ECB (Electronic Code Block) mode. The AES_DESCR MMIO register expects a pointer to a three 32-bit word structure in memory. **The pointer must be 32-bit word aligned.**

The CRYPTO component retrieves the three control words that specify the HW function.

- Word 0: Pointer to a 16 * 8-bit secret key SK. The secret key is a HW function input. The pointer must be 32-bit word aligned.
- Word 1: Pointer to a 16 * 8-bit block P. The block is a HW function input. The pointer must be 32-bit word aligned.
- Word 2: Pointer to a 16 * 8-bit block C. The block is a HW function output. The pointer must be 32-bit word aligned.

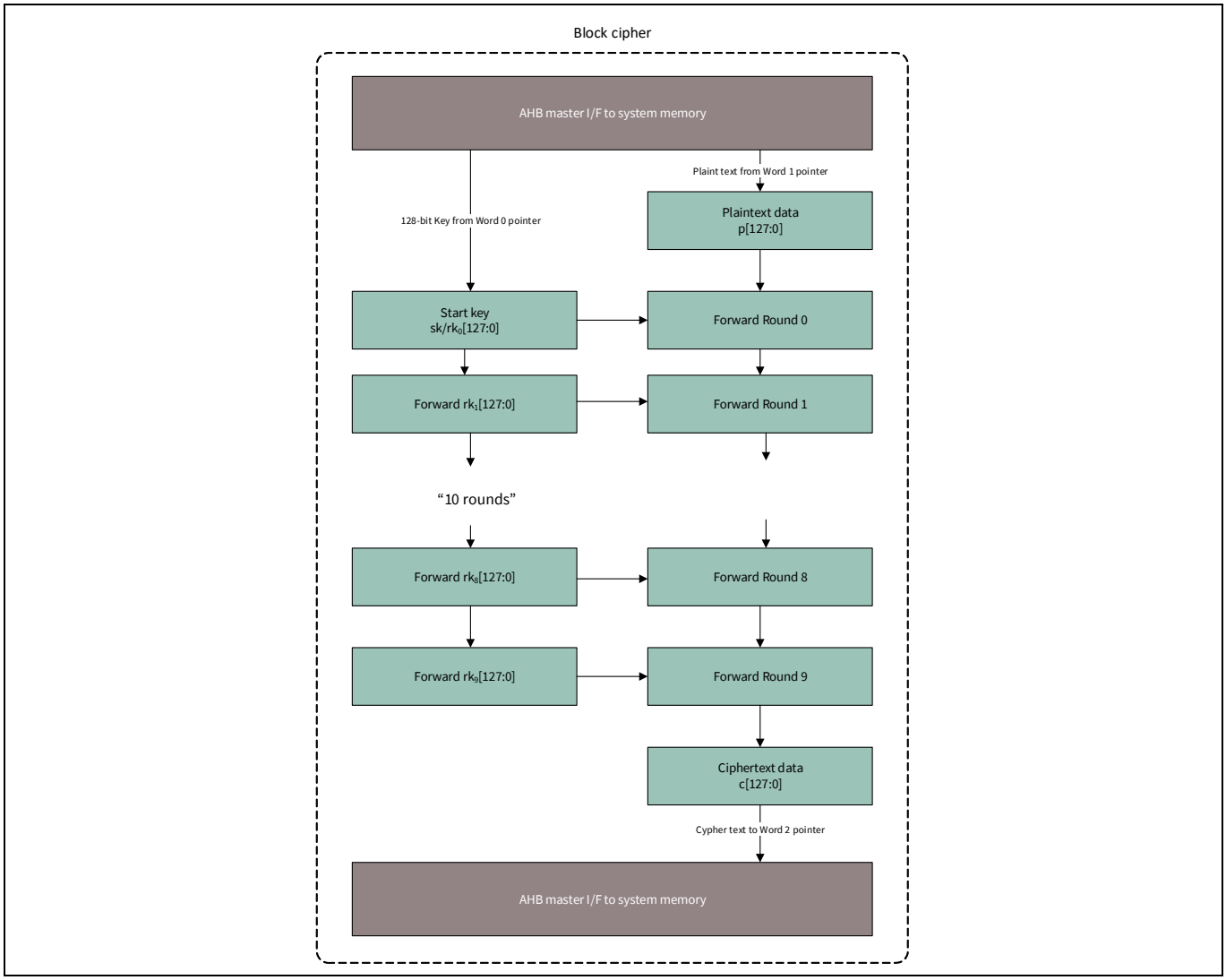



**Figure 7-3. AES data flow**

### Endianness

Since AES-128 block cipher function operates on 8-bit Bytes, no special endianness considerations are applicable HW point of view.

Regarding the alignment of data in the SRAM, the NIST specifies that both message and key are expressed in **big-endian BYTEWISE** format. For example, in the NIST example, the key is mentioned as:

2B7E1516 28AED2A6 ABF71588 09CF4F3C

For EZ-PD™ PMG1-S3 MCU Crypto block, this key is aligned in the memory as follows:

| ADDRESS | Key |
|---|---|
| 0x00 | 2B |
| 0x01 | 7E |
| 0x02 | 15 |
| 0x03 | 16 |
| 0x04 | 28 |
| 0x05 | AE |
| 0x06 | D2 |

| ADDRESS | Key |
|---|---|
| 0x07 | A6 |
| 0x08 | AB |
| 0x09 | F7 |
| 0x0A | 15 |
| 0x0B | 88 |
| 0x0C | 09 |
| 0x0D | CF |
| 0x0E | 4F |
| 0x0F | 3C |

Similar to Key shown in above table, 128-bit input message is also stored in memory in same way.

### 7.2.2 SHA-256

The SHA-256 hash function is a member of the SHA-2 hash family. To calculate a message's hash value (also known as message digest), a message is broken up into multiple 512-bit message chunks. The SHA-256 hash function processes a single message chunk.

The SHA-256 hash function has an internal 256-bit (eight 32-bit words) hash state H[j] (with j = 0, ..., 7). For the first message chunk, the start hash state is initialized with SHA-256 pre-defined values. For each successive message chunk, the start hash state is the previous message chunk's end hash state. The final message chunk's end hash state is the message's hash value.

The SHA-256 hash function performs 64 rounds of a compression function. Each compression function round updates the internal hash state. Each round has a dedicated 32-bit round constant K[i] and a dedicated 32-bit message schedule value W[i]. The round constants have SHA-256 pre-defined values; i.e. they are fixed.

The message schedule values are created from the 512-bit message chunk; i.e. they are message specific.

The EZ-PD™ PMG1-S3 MCU Crypto SHA-256 hash function HW acceleration consists of two HW functions:

1. Message schedule function. Creation of 64 32-bit message schedule values W[i] (with = 0, ..., 63) from a 512-bit message chunk M.
2. Process function. SHA-256 processing of a 256-bit hash state by performing 64 rounds of a compression function. The process inputs are a 256-bit hash state and a message schedule W (as created by the message schedule function). The process output is an updated 256-bit hash state. Note that the round constants K are an implicit, rather than explicit process input (they are hardwired in the process function).

To process a single message chunk, one message schedule HW function is performed, followed by one process HW function.

A write to the SHA_DESCR MMIO register starts a HW function. The SHA_DESCR MMIO register expects a pointer to a three 32-bit word structure in memory. The pointer must be 32-bit word aligned.

The CRYPTO component retrieves the three control words that specify the HW function. The first word (word 0) is generic, the other two words (words 1 and 2) are HW function specific.

- Word 0's Bit position 28 is used to specify the HW function. If WORD0[28] is '0', a message schedule HW function is performed. If WORD0[28] is '1', a process HW function is performed.

For a message schedule function, words 1 and 2 are defined as follows:

- Word 1: Pointer to a 16 * 32-bit (512 bit) message chunk M. The message chunk is a HW function input. The pointer must be 32-bit word aligned.

- Word 2: Pointer a 64 * 32-bit (2048 bit) message schedule array W. The message schedule is a HW function output. The pointer must be 32-bit word aligned.

For a process function, words 1 and 2 are defined as follows:

- Word 1: Pointer to a 8 * 32-bit (256 bit) hash state H. The hash state is a HW function input AND a HW function output (the input is overwritten by the output). The pointer must be 32-bit word aligned.
- Word 2: Pointer a 64 * 32-bit (2048 bit) message schedule array W. The message schedule is a HW function input. The pointer must be 32-bit word aligned.

#### Endianness

The SHA-256 hash function operates on 32-bit words and assumes a big-endian memory layout of these words in memory.

- When a message chunk's 8-bit Bytes are used to construct the message schedule's 32-bit words W[i]. A little to big endian conversion (“LE2BE”) is processed by HW.
- When a final hash value's 32-bit words H[j] are used as a message digest (assuming the message digest is considered a Byte stream). A big to little endian conversion (“BE2LE”) is required by SW.

### 7.2.3 Vector unit

The vector unit (VU) support long integer operations that are used for asymmetric key cryptography such as RSA or ECC.

The operations have a single destination operand (DST) and one or two source operands (SRC0 and SRC1). Each operand has a specific bit size. This bit size is specified in 32-bit word multiples. A value of x specifies a bit size of (x+1) * 32 bits.

The VU supports the following long integer operations:

- MUL operation. This operation is a regular long integer multiplication operation.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] * SRC1[src1_bit_size-1:0]
- XMUL operation. This operation is a carry-less long integer multiplication operation.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] xmul SRC1[src1_bit_size-1:0]
- ADD operation. This operation is a long integer addition operation.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] + SRC1[src1_bit_size-1:0]
- SUB operation. This operation is a long integer subtraction operation.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] - SRC1[src1_bit_size-1:0]
- XOR operation. This operation is a long integer exclusive-or operation.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] ^ SRC1[src1_bit_size-1:0]
- MOV operation. This operation moves/copies a long integer.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0]
- LSR1 operation. This operation shifts a long integer one bit position to the right.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] >> 1
- LSL1 operation. This operation shifts a long integer one bit position to the left.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] << 1
- LSR operation. This operation shifts a long integer a specific number of bit positions to the right.
  - DST[dst_bit_size-1:0] = SRC0[src0_bit_size-1:0] >> “bit shift amount”
- COND_SUB operation. This operation is a conditional long integer subtraction.
  - DST[dst_bit_size-1:0] = (SRC0[src0_bit_size-1:0] >= SRC1[src1_bit_size-1:0]) ? (SRC0[src0_bit_size-1:0] - SRC1[src1_bit_size-1:0]) : SRC0[src0_bit_size-1:0]

Note that the operation’s “greater than equal” operator uses the source bit sizes (not the destination bit size) and the operation’s “subtraction” operator uses the destination bit size.

A write to the VU_DESCR MMIO register starts a VU operation. The VU_DESCR MMIO register expects a pointer to a four 32-bit word structure in memory. The pointer must be 32-bit word aligned.

### Cryptography

The CRYPTO component retrieves the four control words that specify the HW function. The first word (word 0) is generic, the other three words (words 1, 2 and 3) are HW function specific.

- Word 0:
  - Bit positions 31:28 specify the operation.

**Table 7-1. Vector unit opcodes**

| Opcode | Operation |
|---|---|
| 4'h0 | Multiplication (MUL) |
| 4'h1 | Addition (ADD) |
| 4'h2 | Subtraction (SUB) |
| 4'h3 | Exclusive or (XOR) |
| 4'h4 | Binary multiplication (XMUL) |
| 4'h5 | Logical shift right by 1 (LSR1) |
| 4'h6 | Logical shift left by 1 (LSL1) |
| 4'h7 | Logical shift right (LSR) |
| 4'h8 | Conditional subtraction (COND_SUB) |
| 4'h9 | Move (MOV). |
| 4'ha to 4'hf | Reserved. |

  - Bit positions 24:16 specify the word length (minus 1) of the destination operand. A 9-bit word length allows for up to 512 32-bit word (16384-bit) destination operand.
  - Bit positions 15:8 specify the word length (minus 1) of source operand 1 (if present).
  - Bit positions 7: 0 specify the word length (minus 1) of source operand 0 (if present). An 8-bit word length allows for up to 256 32-bit word (8192-bit) source operands.

Note that for reserved opcodes (> 9), no change in HW state and few cycles STATUS.BUSY will be high. This is because, crypto HW does not know if VU opcode is undefined at the point of VU_start initiated by MMIO write, so crypto will be busy until it fetches the opcode from descriptor location and becomes idle (STATUS.BUSY is cleared to '0' when undefined opcode is detected).

For MUL, XMUL, ADD, SUB, XOR, COND_SUB operations, words 1, 2 and 3 are defined as follows:

- Word 1: Pointer to word 0 (least significant word) of source operand 0. The pointer must be 32-bit word aligned.
- Word 2: Pointer to word 0 (least significant word) of source operand 1. The pointer must be 32-bit word aligned.
- Word 3: Pointer to word 0 (least significant word) of the destination operand. The pointer must be 32-bit word aligned.

For LSR1, LSL1, MOV operations, words 1, 2 and 3 are defined as follows:

- Word 1: Pointer to word 0 (least significant word) of the source operand. The pointer must be 32-bit word aligned.
- Word 2: Not used.
- Word 3: Pointer to word 0 (least significant word) of the destination operand. The pointer must be 32-bit word aligned.

For LSR operations, words 1, 2 and 3 are defined as follows:

- Word 1: Pointer to word 0 (least significant word) of the source operand. The pointer must be 32-bit word aligned.
- Word 2: Shift amount. Bit positions 12:0 specify the shift amount.

- Word 3: Pointer to word 0 (least significant word) of the destination operand. The pointer must be 32-bit word aligned.

**Note on operand sizes:**

- Note that source and destination operands have dedicated size specifications. The destination operand may be larger than the source operands. Typically, this is useful for multiplication operations.
- When a destination operand size is smaller than the operation result size, the more significant bits of the operation result are ignored. When a destination operand size is larger than the operation result size, the operation result is '0' extended.

Typically, the source operands and destination operand occupy different memory location. However, to limit memory allocation, it may be beneficial to:

- Use the same memory locations for the two source operands. This is can be done without any restrictions. E.g., the two source operands of an ADD operation are the same to double the source operand.
- Use the same memory locations for a source operand and the destination operand. This comes with restrictions that are operation dependent. Specifically, the following operations allow for the use of the same memory locations for a source operand and the destination operand:
  - ADD, SUB, XOR, MOV, LSR1, LSL1 operations.

The following operations don't allow for the use of the same memory locations for a source operand and the destination operand:

  - MUL, XMUL, LSR, COND_SUB operations.

Note that, the VU operates on 32-bit words multiples and assumes a little-endian memory layout of these words in memory. The MXS40 platform uses a little-endian memory layout. As a result, no endianness conversion is required.

### 7.2.4 True random number generator

The true random number generator component (TRNG) generates true random numbers. The bit size of these generated TRNG numbers is fixed to 32-bits in EZ-PD™ PMG1-S3 MCU.

The TRNG relies on up to six ring oscillators to provide physical noise sources. A ring oscillator consists of a series of inverters connected in a feedback loop to form a ring. Due to (temperature) sensitivity of the inverter delays, jitter is introduced on a ring's oscillating signal. The jittered oscillating signal is sampled to produce a "digitized analog signal" (DAS). This is done for all multiple ring oscillators.

To increase entropy and to reduce bias in DAS bits, the DAS bits are further post-processed. Post-processing involves two steps:

- An optional reduction step (over up to six ring oscillator DAS bits and over one or multiple DAS bit periods) to increase entropy.
- An optional "von Neumann correction" step to reduce a '0' or '1' bias.

This correction step processes pairs of reduction bits as produced by the previous step. Given two reduced bits r0 and r1 (with r0 being produced before r1), the correction step is defined as follows:

  - {r0, r1} = {0, 0}: no bit is produced
  - {r0, r1} = {0, 1}: a '0' bit is produced (bit r0)
  - {r0, r1} = {1, 0}: a '1' bit is produced (bit r0)
  - {r0, r1} = {1, 1}: no bit is produced

In other words, the correction step only produces a bit on a '0' to '1' or '1' to '0' transition. Note that for a random input bit sequence, the correction step produces an output bit sequence of roughly ¼ the frequency of the input bit sequence (the input reduction bits are processed in non-overlapping pairs and only half of the pair encodings result in an output bit).

### Cryptography

Post-processing produces bit samples that are considered true random bit samples. The true random bit samples are shifted into a register, to provide random values of 32 bits.

As a result of high switching activity, ring oscillators consume a significant amount of power. Therefore, when the TRNG functionality is disabled, the ring is “broken” to prevent switching. When the TRNG functionality is enabled, a ring oscillator initially has predictable behavior. However, over time, infinitesimal environmental (temperature) changes cause an increasing deviation from this predictable behavior.

- DURING the initial delay, the ring oscillator is NOT a reliable physical noise source.
- AFTER an initial delay, the same ring oscillator will show different oscillation behavior and provides a reliable physical noise source.
- Therefore, the DAS bits can be dropped during an initialization period (TR_CTL.INIT_DELAY[]).

The following figure gives an overview of the TRNG component.

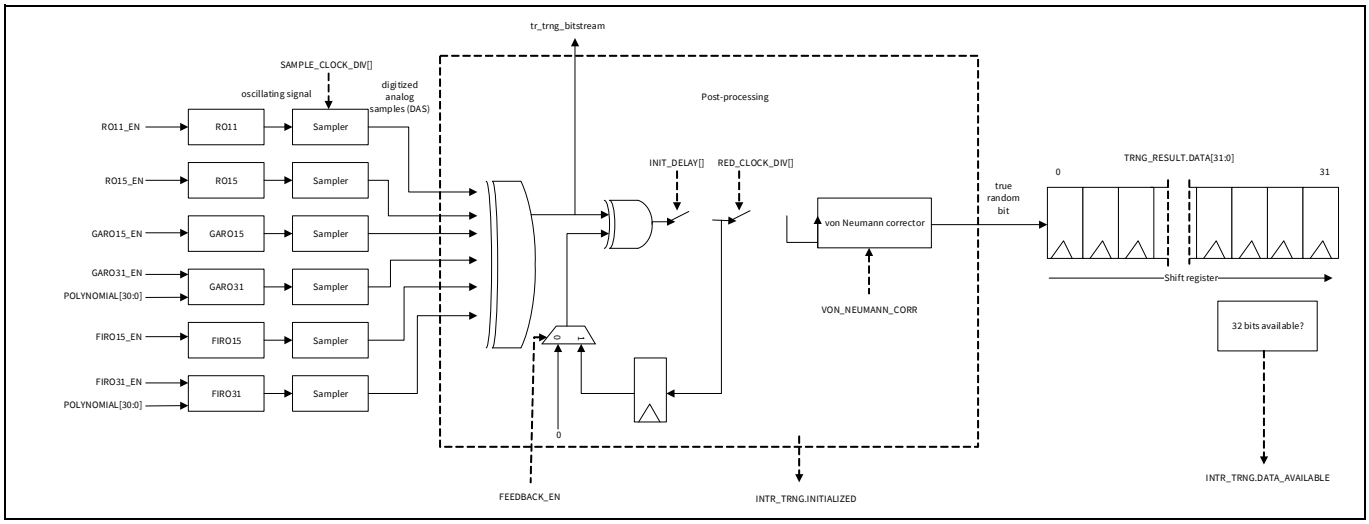



**Figure 7-4. TRNG overview**

## 7.3 Interrupts

The Soft IP provides two interrupts:

- Interrupt_trng: This interrupt is used to report TRNG behaviour.

This interrupt is generated for below scenarios:

1. TRNG is initialized.
2. TRNG data is available.
3. TRNG Repetition Count test fails.
4. TRNG Adaptive Proportion test fails.

- interrupt_error: This interrupt is used to report irregular/erroneous behavior.

This interrupt is generated for below scenarios:

1. An AHB-Lite bus error on AHB master I/F.
2. An ECC error (correctable or non-correctable) is detected on a read transfer. Note that the IP will not correct correctable ECC errors, as this would introduce a critical timing path. (Note that ECC error is reported only when design time configuration parameter ECC_PRESENT is set to ‘1’ and also SW programming register ECC_CTL.CHECK_EN is set to ‘1’).

## 7.4 Registers

See EZ-PD™ PMG1-S3 MCU registers TRM for detailed list of registers.

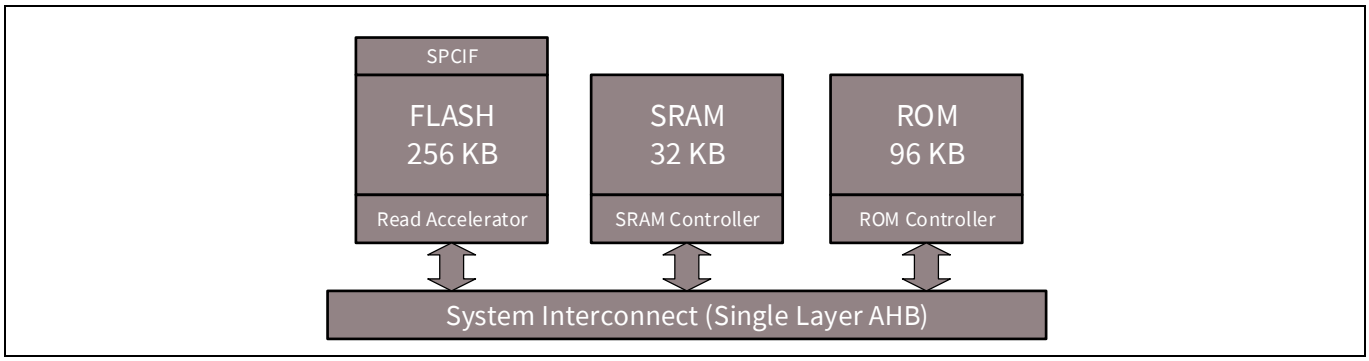
# Section C: Memory system

This section presents the following chapter:

- **"Memory map"** on page 74

## Top level architecture

### Memory system block diagram



# 8 Memory map

All EZ-PD™ PMG1-S3 MCU memory (flash, SRAM, and SROM) and all registers are accessible by the CPU and in most cases by the debug system. This chapter contains an overall map of the addresses of the memories and registers.

## 8.1 Features

The EZ-PD™ PMG1-S3 MCU memory system has the following features:

- 256 KB flash, 32 KB RAM
- 8 KB boot SROM containing boot and configuration routines
- 88 KB user SROM containing commonly used firmware routines
- Arm® Cortex®-M0+ 32-bit linear address space, with regions for code, SRAM, peripherals, and CPU internal registers
- Flash is mapped to the Cortex®-M0+ code region
- SRAM is mapped to the Cortex®-M0+ SRAM region
- Peripheral registers are mapped to the Cortex®-M0+ peripheral region
- The Cortex®-M0+ Private Peripheral Bus (PPB) region includes registers implemented in the CPU core. These include registers for NVIC, SysTick timer, and SCB. For more information, see the **"Cortex®-M0+ CPU"** on page 32.

## 8.2 How it works

The EZ-PD™ PMG1-S3 MCU memory map is detailed in the following tables. For additional information, refer to the EZ-PD™ PMG1-S3 MCU registers TRM.

The Arm® Cortex®-M0+ has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in **Table 8-1**. Note that code can be executed from the code and SRAM regions.

**Table 8-1. Cortex®-M0+ address map**

| Address range | Name | Use |
|---|---|---|
| 0x00000000-0x1FFFFFFF | Code | Executable region for program code. You can also put data here. Includes the exception vector table, which starts at address 0. |
| 0x20000000-0x3FFFFFFF | SRAM | Executable region for data. You can also put code here. |
| 0x40000000-0x5FFFFFFF | Peripheral | All peripheral registers. Code cannot be executed from this region. |
| 0x60000000-0xDFFFFFFF | – | Not used |
| 0xE0000000-0xE00FFFFF | PPB | Peripheral registers within the CPU core. |
| 0xE0100000-0xFFFFFFFF | Device | EZ-PD™ PMG1-S3 MCU implementation-specific. |

### Memory map

Table 8-2 shows the EZ-PD™ PMG1-S3 MCU address map.

**Table 8-2. EZ-PD™ PMG1-S3 MCU address map**

| Address range | Use |
|---|---|
| 0x00000000–0x0003FFFF | 256 KB flash |
| 0x0FFFF000–0x0FFFF3FF | 1 KB supervisory flash |
| 0x10000000-0x10001FFF | 8 KB Boot SROM (no direct read/execute access) |
| 0x10002000-0x10017FFF | 88 KB User SROM (read/execute access allowed) |
| 0x20000000–0x20007FFF | 32 KB SRAM |
| 0x40010000–0x4001FFFF | Peripheral Interconnect (PERI) registers |
| 0x40020000–0x40023FFF | I/O port control (high-speed I/O matrix) registers |
| 0x40030000–0x4003FFFF | Power, clock, reset control registers |
| 0x40040000–0x40043FFF | I/O port registers |
| 0x40050000–0x4008FFFF | SCB registers |
| 0x40090000–0x4009FFFF | TCPWM registers |
| 0x400A0000–0x400BFFFF | USB Power Delivery Controller registers |
| 0x400C0000-0x400CFFFF | Crypto (TRNG) component registers |
| 0x40100000–0x4011FFFF | CPU subsystem registers |
| 0xE0000000–0xE00FFFFF | Cortex®-M0+ PPB registers |
| 0xF0000000–0xF0000FFF | CoreSight ROM |

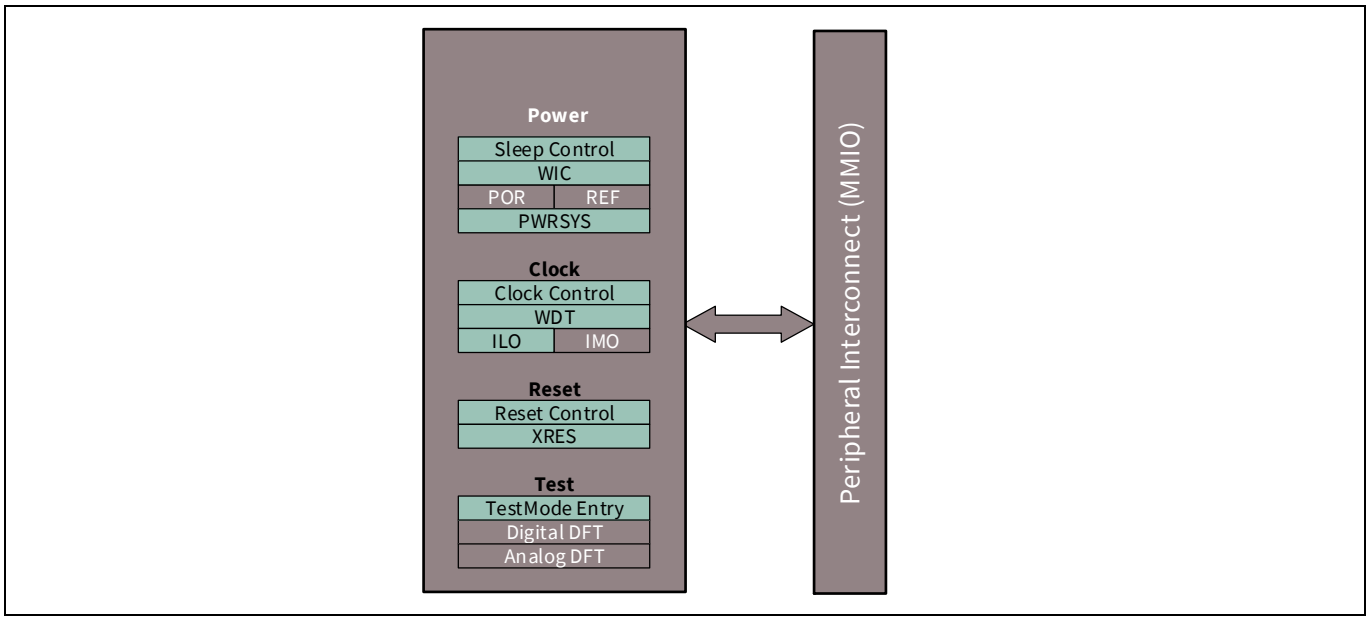
# Section D: System-wide resources

This section encompasses the following chapters:


- **"I/O system"** on page 77
- **"Clocking system"** on page 85
- **"Power supply and monitoring"** on page 91
- **"Chip operational modes"** on page 94
- **"Power modes"** on page 95
- **"Watchdog timer"** on page 99
- **"Reset system"** on page 102
- **"Device security"** on page 104
- **"Trigger multiplexer block"** on page 105


## Top level architecture

### System-wide resources block diagram



# 9 I/O system

This chapter explains the EZ-PD™ PMG1-S3 MCU I/O system, its features, architecture, operating modes, and interrupts. The general- purpose I/O (GPIOs) pins in EZ-PD™ PMG1-S3 MCU are grouped into ports; a port can have a maximum of eight GPIOs. The EZ-PD™ PMG1-S3 MCU device has a maximum of 50 GPIOs arranged in 8 ports.

## 9.1 Features

The EZ-PD™ PMG1-S3 MCU GPIOs have these features:

- Analog and digital input and output capabilities
- 10-mA sink and 4-mA source current in digital mode
- Separate port read (PS) and write (DR) data registers to avoid read-modify-write errors
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Two over-voltage tolerant GPIOs ($I^2C$ pins from SCB0)

## 9.2 Block diagram

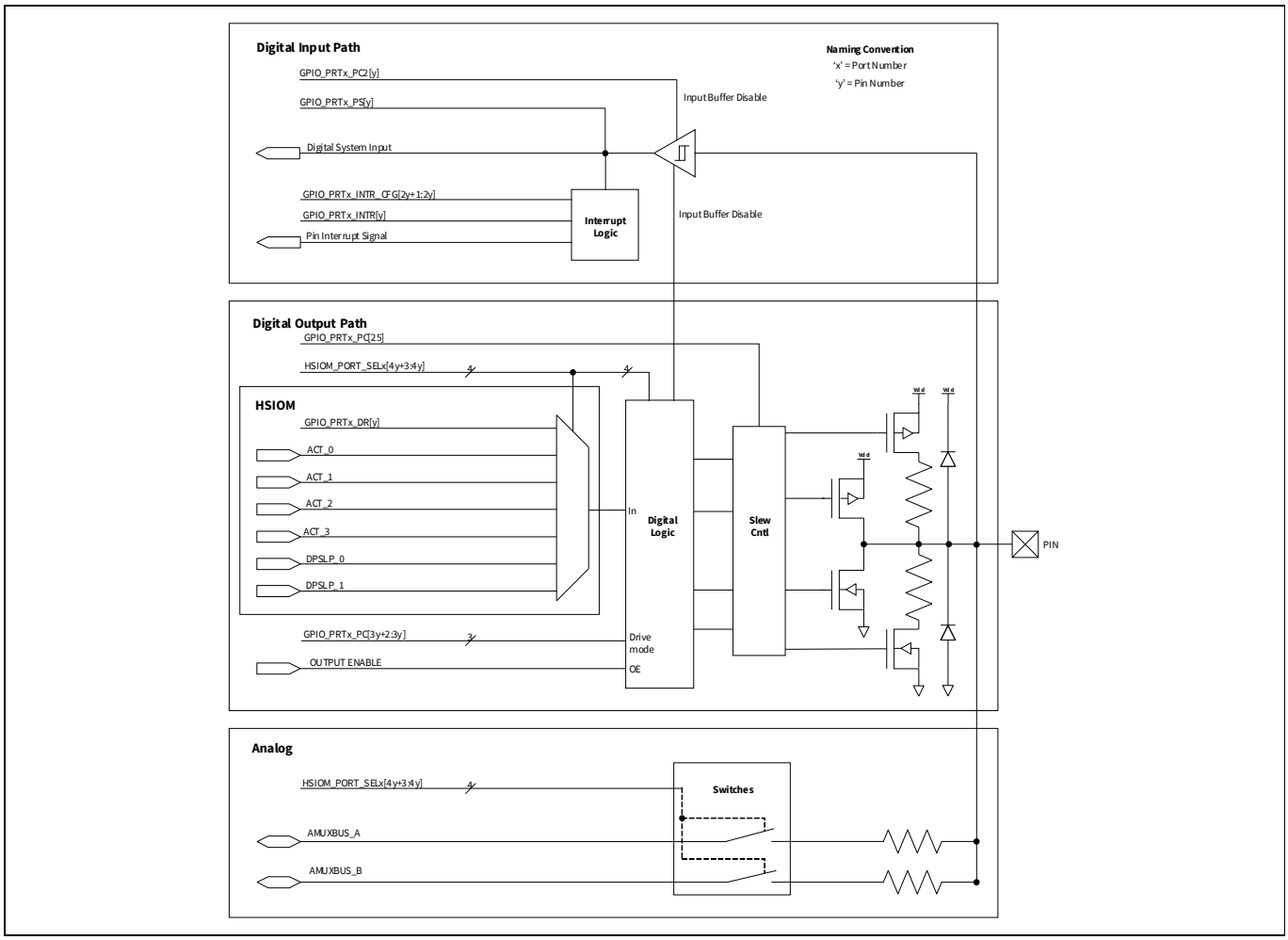



**Figure 9-1. GPIO block diagram**

*Note: The GPIO features shown in this image may not be available on all the pins. Check EZ-PD™ PMG1-S3 MCU datasheet for pin capabilities.*

## 9.3 GPIO drive modes

Each I/O is individually configurable into one of the eight drive modes listed in **Table 9-1**. **Figure 9-2** is a simplified pin diagram that shows the pin view based on each of the eight drive modes.

Two port configuration registers are used to configure GPIOs in EZ-PD™ PMG1-S3 MCU:

Port Configuration Register (GPIO_PRTx_PC) and Port Secondary Configuration Register (GPIO_PRTx_PC2). All EZ-PD™ PMG1-S3 MCU devices have ports with dedicated GPIO_PRTx_PC and GPIO_PRTx_PC2 registers.

GPIO_PRTx_PC is used to configure the following properties of a port:

- Output drive mode of each pin (three bits select a particular drive mode for a pin)
- Slew rate of the whole port (see **"Slew rate control"** on page 80)
- Input threshold selection of the whole port (see **"CMOS LVTTL level control"** on page 80)

GPIO_PRTx_PC2 is used to enable/disable the input buffer of each pin on the port, irrespective of the drive mode configured in GPIO_PRTx_PC. When analog signals are present on the pin, input buffer should be disabled by setting the bit to '1'.

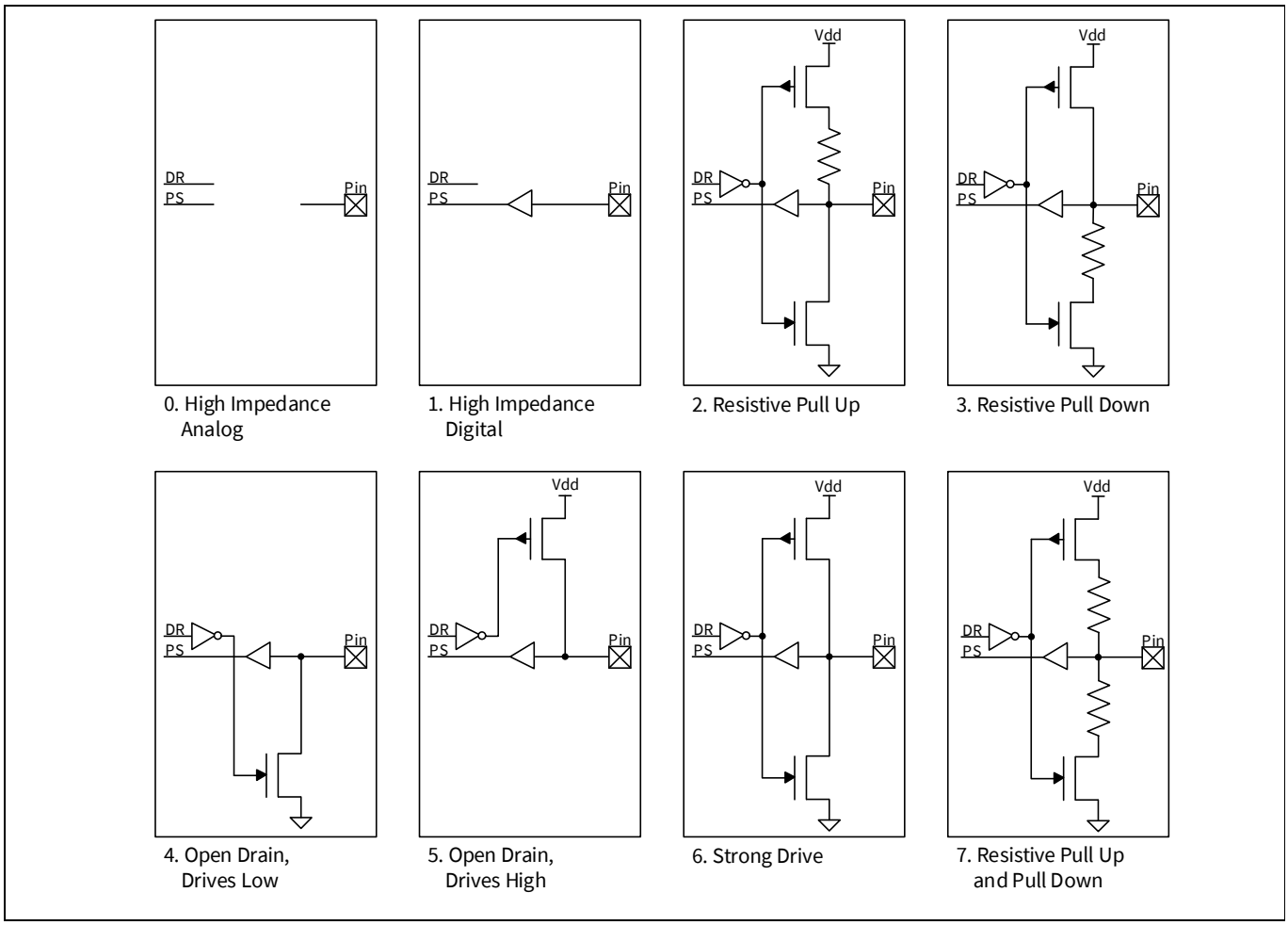



**Figure 9-2. I/O drive mode block diagram**

**Table 9-1. Drive mode settings**

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

**Table 9-2. Input buffer disable (Port configuration 2)**

| GPIO_PRTx_PC2 ('x' denotes port number and 'y' denotes pin number) | | |
|---|---|---|
| **Bits** | **Name** | **Description** |
| 7:0 | INP_DIS | Disables the input buffer independent of the port control drive mode. This bit should be set when analog signals are present on the pin. |

### 9.3.1 High-impedance analog

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

To achieve the lowest device current in low-power modes, unused GPIOs must be configured to the high-impedance analog mode.

### 9.3.2 High-impedance digital

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

### 9.3.3 Resistive pull-up or resistive pull-down

Resistive modes provide a series resistance in one of the data states and strong drive in the other. Pins can be used for either digital input or digital output in these modes. If resistive pull-up is required, a '1' must be written to that pin's Data Register bit. If resistive pull-down is required, a '0' must be written to that pin's Data Register. Interfacing mechanical switches is a common application of these drive modes. The resistive modes are also used to interface EZ-PD™ PMG1-S3 MCU with open drain drive lines. Resistive pull-up is used when input is open drain low and resistive pull-down is used when input is open drain high.

### 9.3.4 Open drain drives high and open drain drives low

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

### 9.3.5 Strong drive

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

### 9.3.6 Resistive pull-up and resistive pull-down

This mode is similar to the drive modes explained in **"Resistive pull-up or resistive pull-down"** on page 79. In the resistive pull-up and resistive pull-down mode, the GPIO will have a series resistance in both logic 1 and logic 0 output states. The high data state is pulled up while the low data state is pulled down. This mode is used when the bus is driven by other signals that may cause shorts.

## 9.4 Slew rate control

GPIO pins have fast and slow output slew rate options in strong drive mode; this can be configured using the GPIO_PRTx_PC[25] bit. Slew rate is individually configurable for each port. This bit is cleared by default and the port works in fast slew mode. This bit can be set if a slow slew rate is required. The fast slew rate is recommended for signals higher than 1 MHz. Slower slew rate results in reduced EMI and crosstalk; hence, the slow option is recommended for signals that are not speed critical – generally less than 1 MHz.

## 9.5 CMOS LVTTL level control

I/O pins can work at two voltage levels. These levels can be selected by writing to the GPIO_PRTx_PC[24] bit.

Input level is individually configurable for each port. This bit is cleared by default and the port works in CMOS mode. This bit can be set to reconfigure the port to LVTTL mode.

CMOS mode can be used in most cases, whereas LVTTL can be used for custom interface requirements, which works at lower voltage levels. See the **device datasheet** for the input voltage thresholds (VIH and VIL) for the modes.

## 9.6 GPIO-OVT

EZ-PD™ PMG1-S3 MCU device has two over-voltage tolerant (OVT) pins – $I^2C$ pins from SCB0. These are similar to regular GPIOs with the following additional features:

- Over-voltage tolerant
- Provides better pull-down drive strength
- SCB when configured as $I^2C$ and its lines routed to GPIO-OVT pins; it meets the following $I^2C$ specifications:
  - Fast Mode hot-swap
  - Fast Mode Plus IOL Specification
  - Fast Mode and Fast Mode Plus Hysteresis and minimum fall time specifications

See the **device datasheet** for specifications.

## 9.7 High-speed I/O matrix

High-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the resources inside EZ-PD™ PMG1-S3 MCU. These resources include TCPWMs, SCB, and the USB PD. The HSIOM selects Active and Deep-Sleep power domain sources for a pin. HSIOM_PORT_SELx are 32-bit wide registers that control the routing of GPIOs. Each register controls one port; four dedicated bits are assigned to each GPIO in the port. This provides up to 16 different options for GPIO routing. This selection provides different pin functions, as listed in **Table 9-3**.

**Table 9-3. HSIOM port settings**

| HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number) | | | |
|---|---|---|---|
| **Bits** | **Name (SEL 'y')** | **Value** | **Description (Selects pin 'y' source (0 ≤ y ≤ 7))** |
| 4y+3: 4y | DR | 0 | Pin is regular firmware-controlled GPIO. |
| | AMUXA | 6 | Pin is connected to AMUXBUS-A. |
| | AMUXB | 7 | Pin is connected to AMUXBUS-B. |
| | ACT_0 | 8 | Pin-specific Active source # 0 (TCPWM, EXT CLOCK). |
| | ACT_1 | 9 | Pin-specific Active source #1 (TCPWM). |
| | ACT_2 | 10 | Pin-specific Active source #2 (SCB–UART). |
| | ACT_3 | 11 | Pin-specific Active source #3 (TCPWM). |
| | DPSLP_0 | 12 | Pin-specific Deep-Sleep source #0 (SWD, USBPD) |
| | DPSLP_1 | 13 | Pin-specific Deep-Sleep source #1 (SCB–SPI) |
| | DPSLP_2 | 14 | Pin-specific Deep-Sleep source #2 (SCB–SPI) |
| | DPSLP_3 | 15 | Pin-specific Deep-Sleep source #3 (SCB–I2C) |

*Note: The active and deep-sleep sources are pin dependent. See the "Pinouts" section of the **device datasheet** for more details on the features supported by each pin.*

## 9.8 Firmware controlled GPIO

See **Table 9-3** to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

In addition to the data register, two other registers – GPIO_PRTx_DR_SET and GPIO_PRTx_DR_CLR – are provided to set or clear the output data of specific GPIOs in port. Additionally, the GPIO_PRTx_DR_INV register can be used to invert the output data of a specific GPIO. GPIO_PRTx_PS is the port I/O pad register that provides the state of the GPIOs when read. Writes to this register have no effect.

See the EZ-PD™ PMG1-S3 MCU registers TRM for details of these registers.

## 9.9 I/O port reconfiguration

Drive mode and GPIO can be reconfigured in runtime by changing the value of the GPIO_PRTx_PC and HSIOM_PORT_SELx registers. Take care to retain the pin state during reconfiguration of pins when they are connected directly to a digital peripheral. If the ports are driven by the data registers, state maintenance is automatic. During port configuration, the current configuration should be saved as follows:

1. Read the GPIO pin state – GPIO_PRTx_PS in software.
2. Write the GPIO_PRTx_PS value into the data registers – GPIO_PRTx_DR.
3. Change the corresponding field in HSIOM_PORT_SELx to drive the pin by the data register – GPIO_PRTx_DR.

## 9.10 I/O state on power up

By default, during power up all GPIOs are in high-impedance analog state and input buffers are disabled. When the device is powered, its GPIOs can be configured according to the required application, by writing to the associated registers.

## 9.11 Behavior in low-power modes

The GPIOs maintain the current pin state during Sleep mode. In Sleep mode, all the GPIOs are active and can be driven by active peripherals, such as USB PD, TCPWM, and SCB.

In Deep-Sleep mode, all the pin states are latched and the pin signals are retained, except the SCB pins, which remain functional and can wake up the processor on $I^2C$ address matching event. GPIO interrupts are also available in Deep-Sleep mode with wakeup ability.

For more details, see the **"Power modes"** on page 95, **"Interrupts"** on page 38, and **"Serial Communications Block (SCB)"** on page 112.

To achieve the lowest device current in low-power modes, unused I/Os must be configured in the high-impedance analog mode.

## 9.12 GPIO interrupt

This section describes the interrupt functionality of the EZ-PD™ PMG1-S3 MCU GPIOs.

### 9.12.1 Features

The features of the GPIO interrupt are:

- All eight pins in each port interface have an interrupt and an associated interrupt vector
- Pin status bits provide easy determination of interrupt source down to the pin level
- Rising, falling, or both edge-triggered interrupts are handled
- Pin interrupts can be individually enabled or disabled
- AHB interfaces for read and write into its registers
- Sends out a single port interrupt request (PIRQ) signal, derived from all GPIOs in a port, to the interrupt controller

### 9.12.2 Interrupt controller block diagram

Each port has its own individual interrupt request and associated interrupt request (IRQ) vector and interrupt service routine (ISR). Additionally, one pin can be selected on each port that is routed through a 50-ns glitch filter to form a glitch-tolerant interrupt for the port. The details are shown in **Figure 9-3**.

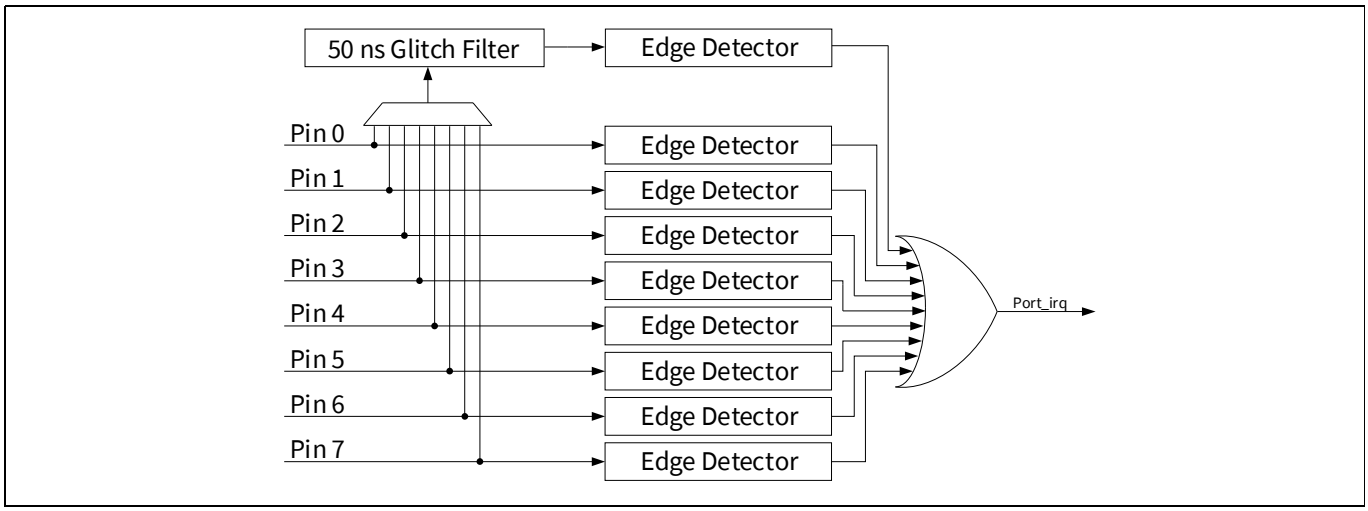



**Figure 9-3. Interrupt generator**

### 9.12.3 Function and configuration

Each pin of the port can be configured independently to generate an interrupt on the rising edge, falling edge, or on both edges by writing to the GPIO_PRTx_INTR_CFG register. Level-sensitive interrupts are not supported. GPIO_PRTx_INTR_CFG is also used to route a specific channel to the glitch filter and generate a ninth glitch-tolerant interrupt.

When a GPIO interrupt is triggered by a signal on an interrupt-enabled port pin, the GPIO_PRTx_INTR register (Port Interrupt Status Register) is updated. The firmware can read this register to determine which GPIO triggered the interrupt. Firmware can then clear the IRQ bit by writing a '1' to its corresponding bit.

Additionally, when the Port Interrupt Control Status Register is read at the same time an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code.

See GPIO_PRTx_INTR_CFG and GPIO_PRTx_INTR in the EZ-PD™ PMG1-S3 MCU registers TRM for details.

## 9.13 Registers

**Table 9-4. I/O registers**

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

*Note: The 'x' in the register name denotes the port number. For example, GPIO_PRT1_DR is the port 1 output data register.*

# 10 Clocking system

The EZ-PD™ PMG1-S3 MCU clock system includes these clock resources:

- Two internal clock sources:
  - 24–48 MHz internal main oscillator (IMO) ±2 percent across all frequencies with trim
  - 32-kHz internal low-speed oscillator (ILO)
- High-frequency clock (HFCLK) of up to 48 MHz selected from IMO or external clock
  - Dedicated prescaler for HFCLK
- Low-frequency clock (LFCLK) sourced by ILO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- EZ-PD™ PMG1-S3 MCU device has 21 divider clocks, 12 clocks with an 8-bit divider, four clocks with a 16-bit divider, and 5 clocks with a 16.5-bit divider, used for 21 outputs in total

## 10.1 Block diagram

**Figure 10-1** gives a generic view of the clocking system in EZ-PD™ PMG1-S3 MCU devices.

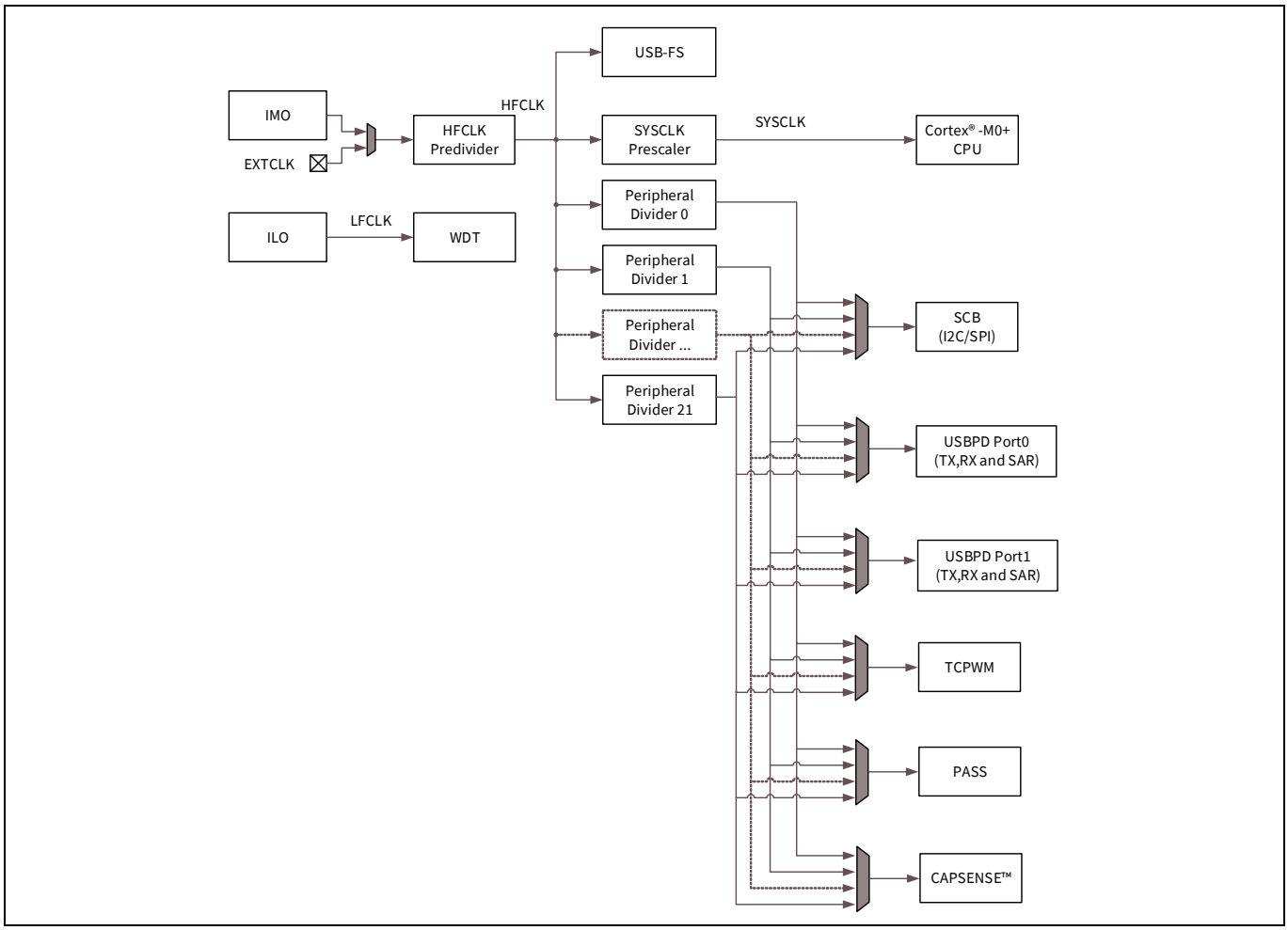



**Figure 10-1. Clocking system block diagram**

The three clock sources in the device are shown in **Figure 10-1**. The HFCLK mux selects the HFCLK source from an external clock source or the IMO. The HFCLK predivider divides the HFCLK input. The SYSCLK prescaler generates the SYSCLK and the peripheral dividers generate the individual peripheral clocks. The ILO sources the LFCLK.

Each of the SCB and TCPWM blocks used in the application need a clock generated by one of the peripheral dividers. In addition, each of the USB PD blocks require multiple clocks as shown in **Table 10-7**.

## 10.2 Clock sources

### 10.2.1 Internal main oscillator

The internal main oscillator operates with no external components and outputs a stable clock at frequencies spanning 24-48 MHz in 4-MHz increments. Frequencies are selected by setting the frequency range in the CLK_IMO_SELECT register, and setting the IMO trim in the CLK_IMO_TRIM1 register. Each device has an IMO trim measured during manufacturing to meet datasheet specifications; the trim is stored in the manufacturing configuration data in SFLASH.

These values may be loaded at startup to achieve the desired configuration. Firmware can retrieve these trim values and reconfigure the device to change the frequency at runtime.

#### 10.2.1.1 Startup behavior

After reset, the IMO is configured for 24-MHz operation. During the “boot” portion of startup, trim values are read from flash and the IMO is configured to achieve datasheet specified accuracy. The HFCLK predivider is initially set to a divide value of ‘4’ to reduce current consumption at startup.

### 10.2.2 Internal low-speed oscillator

The internal low-speed oscillator operates with no external components and outputs a stable clock at 32-kHz nominal. The ILO is relatively low power and low accuracy. It is available in all power modes. The ILO is always used as the system low-frequency clock, LFCLK, in EZ-PD™ PMG1-S3 MCU. The ILO is recommended to be always ON, because it is the source of the WDT, which is required for reliable system operation. The ILO can be disabled by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDT reset will automatically enable the ILO.

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

- WDT protection is required against firmware crashes.
- WDT protection is required against the power supply events that produce sudden brownout events, which may in turn compromise the CPU functionality.

See the **“Watchdog timer”** on page 99 for details.

### 10.2.3 External clock

The external clock is a MHz range clock that can be generated from a signal on a designated EZ-PD™ PMG1-S3 MCU pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is 0–48 MHz. EZ-PD™ PMG1-S3 MCU always starts up using the IMO, and the external clock must be enabled in user mode, so the device cannot be started from a reset clocked by the external clock.

## 10.3 Clock distribution

EZ-PD™ PMG1-S3 MCU clocks are developed and distributed throughout the device, as shown in **Figure 10-1**. The distribution configuration options are as follows:

- HFCLK input selection
- HFCLK predivider configuration
- SYSCLK prescaler configuration
- Peripheral divider configuration

### 10.3.1 HFCLK input selection

HFCLK in EZ-PD™ PMG1-S3 MCU has two input options: IMO and EXTCLK. The HFCLK input is selected using the CLK_SELECT register's HFCLK_SEL bits, as described in **Table 10-1**.

**Table 10-1. HFCLK input selection bits HFCLK_SEL**

| Name | Description |
|---|---|
| HFCLK_SEL[2:0] | HFCLK input clock selection<br>0: IMO. Uses the IMO as the source of the HFCLK<br>1: EXTCLK. Uses the EXTCLK as the source of the HFCLK<br>2–7: Reserved. Do not use |

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the **"I/O system"** on page 77 for more details.

### 10.3.2 HFCLK predivider configuration

The HFCLK predivider allows the device to divide the HFCLK selection mux input before use as HFCLK. The predivider is capable of dividing the HFCLK by powers of two between 1 and 8. The predivider value is set using register CLK_SELECT bits HFCLK_DIV, as described in **Table 10-2**. The HFCLK predivider is set to a divide value of '4' during startup to reduce current consumption.

**Table 10-2. HFCLK predivider value bits HFCLK_DIV**

| Name | Description |
|---|---|
| HFCLK_DIV[1:0] | HFCLK predivider value<br>0: 1 (no divider on HFCLK)<br>1: 2 (divides HFCLK by 2)<br>2: 4 (divides HFCLK by 4)<br>4: 8 (divides HFCLK by 8) |

### 10.3.3 SYSCLK prescaler configuration

The SYSCLK prescaler allows the device to divide the predivided HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by powers of two between 1 and 8. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in **Table 10-3**. The prescaler is initially configured to divide by 1.

**Table 10-3. SYSCLK prescaler divide value bits SYSCLK_DIV**

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

### 10.3.4 Peripheral clock divider configuration

The sixteen peripheral clocks are derived from the HFCLK using the 8-bit, 16-bit, or 16.5-bit peripheral clock dividers. Each of the sixteen dividers (eight 8-bit, four 16-bit and four 16.5 bit) is controlled by a PERI_DIV_8_CTL, PERI_DIV_16_CTL, or PERI_DIV_16_5_CTL register, whose mapping is listed in **Table 10-4**, **Table 10-5**, and **Table 10-6** respectively.

**Table 10-4. Peripheral clock divider control register PERI_DIV_8_CTLx**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enables or disables the divider<br>0: Divider disabled<br>1: Divider enabled |
| 15:8 | INT8_DIV | Divide value of the divider. Output = Input/(INT8_DIV + 1) |

**Table 10-5. Peripheral clock divider control register PERI_DIV_16_CTLx**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enables or disables the divider<br>0: Divider disabled<br>1: Divider enabled |
| 23:8 | INT16_DIV | Divide value for the divider. Output = Input/(INT16_DIV + 1). Acceptable divide values range from 0 to 65,536. |

**Table 10-6. Peripheral clock divider control register PERI_DIV_16_5_CTLx**

| Bits | Name | Description |
|---|---|---|
| 0 | EN | Enables or disables the divider<br>0: Divider disabled<br>1: Divider enabled |
| 7:3 | FRAC5_DIV | Fractional division by (FRAC5_DIV/32) |
| 23:8 | INT16_DIV | Integer division by (1+INT16_DIV). Acceptable divide values range from 0 to 65536. |

The PERI_DIV_CMD register can be used to enable, disable, and select the type of clock dividers for all peripheral clock dividers. See the PERI_DIV_CMD in the EZ-PD™ PMG1-S3 MCU registers TRM for more details.

### Clocking system

Input clocks to the peripherals are selected by PERI_PCLK_CTLx registers. **Table 10-7** shows the peripheral clocks and their respective registers. See the EZ-PD™ PMG1-S3 MCU registers TRM for more details.

**Table 10-7. Selecting peripheral clocks**

| Clock | Register |
|---|---|
| SCB0 | PERI_PCLK_CTL0 |
| SCB1 | PERI_PCLK_CTL1 |
| SCB2 | PERI_PCLK_CTL2 |
| SCB3 | PERI_PCLK_CTL3 |
| SCB4 | PERI_PCLK_CTL4 |
| SCB5 | PERI_PCLK_CTL5 |
| SCB6 | PERI_PCLK_CTL6 |
| SCB7 | PERI_PCLK_CTL7 |
| TCPWM0 | PERI_PCLK_CTL8 |
| TCPWM1 | PERI_PCLK_CTL9 |
| TCPWM2 | PERI_PCLK_CTL10 |
| TCPWM3 | PERI_PCLK_CTL11 |
| TCPWM4 | PERI_PCLK_CTL12 |
| TCPWM5 | PERI_PCLK_CTL13 |
| TCPWM6 | PERI_PCLK_CTL14 |
| TCPWM7 | PERI_PCLK_CTL15 |
| CSD | PERI_PCLK_CTL16 |
| SAR ADC | PERI_PCLK_CTL17 |
| USBPD0 - RX | PERI_PCLK_CTL18 |
| USBPD0 - TX | PERI_PCLK_CTL19 |
| USBPD0 - SAR ADC | PERI_PCLK_CTL20 |
| USBPD0 - SWAP | PERI_PCLK_CTL21 |
| USBPD0 - Filter1 | PERI_PCLK_CTL22 |
| USBPD0 - Filter2 | PERI_PCLK_CTL23 |
| USBPD0 - RefGen | PERI_PCLK_CTL24 |
| USBPD0 - BchDet | PERI_PCLK_CTL25 |
| USBPD1 - RX | PERI_PCLK_CTL26 |
| USBPD1 - TX | PERI_PCLK_CTL27 |
| USBPD1 - SAR ADC | PERI_PCLK_CTL28 |
| USBPD1 - SWAP | PERI_PCLK_CTL29 |
| USBPD1 - Filter1 | PERI_PCLK_CTL30 |
| USBPD1 - Filter2 | PERI_PCLK_CTL31 |

## 10.4 Low-power mode operation

The EZ-PD™ PMG1-S3 MCU clock behavior is different in different power modes. The MHz frequency clocks including the IMO, EXTCLK, HFCLK, SYSCLK, and peripheral clocks operate only in Active and Sleep modes. The ILO and LFCLK operate in all power modes.

## 10.5 Registers

**Table 10-8. Clocking system registers**

| Register name | Description |
|---|---|
| CLK_IMO_TRIM1 | IMO Trim Register – This register contains IMO trim, allowing fine manipulation of its frequency. |
| CLK_IMO_SELECT | IMO Frequency Selection Register – This register controls the frequency range of the IMO, allowing gross manipulation of its frequency. |
| CLK_ILO_CONFIG | ILO Configuration Register – This register controls the ILO configuration. |
| CLK_SELECT | Clock Select – This register controls clock tree configuration, selecting different sources for the system clocks. |
| PERI_DIV_8_CTLx/ PERI_DIV_16_CTLx /PERI_DIV_16_5_CTLx | Peripheral Clock Divider Control Registers – These registers configure each of the peripheral clock dividers, enabling or disabling the divider and setting the integer divide value. |
| PERI_PCLK_CTLx | Programmable clock control registers – These registers are used to select the input clocks to peripherals. |
| PERI_DIV_CMD | Peripheral Divider Command --This register enables the PCLK divider. |

# 11 Power supply and monitoring

EZ-PD™ PMG1-S3 MCU is capable of operating from two different power sources – VBUS and VSYS.

## 11.1 Block diagram

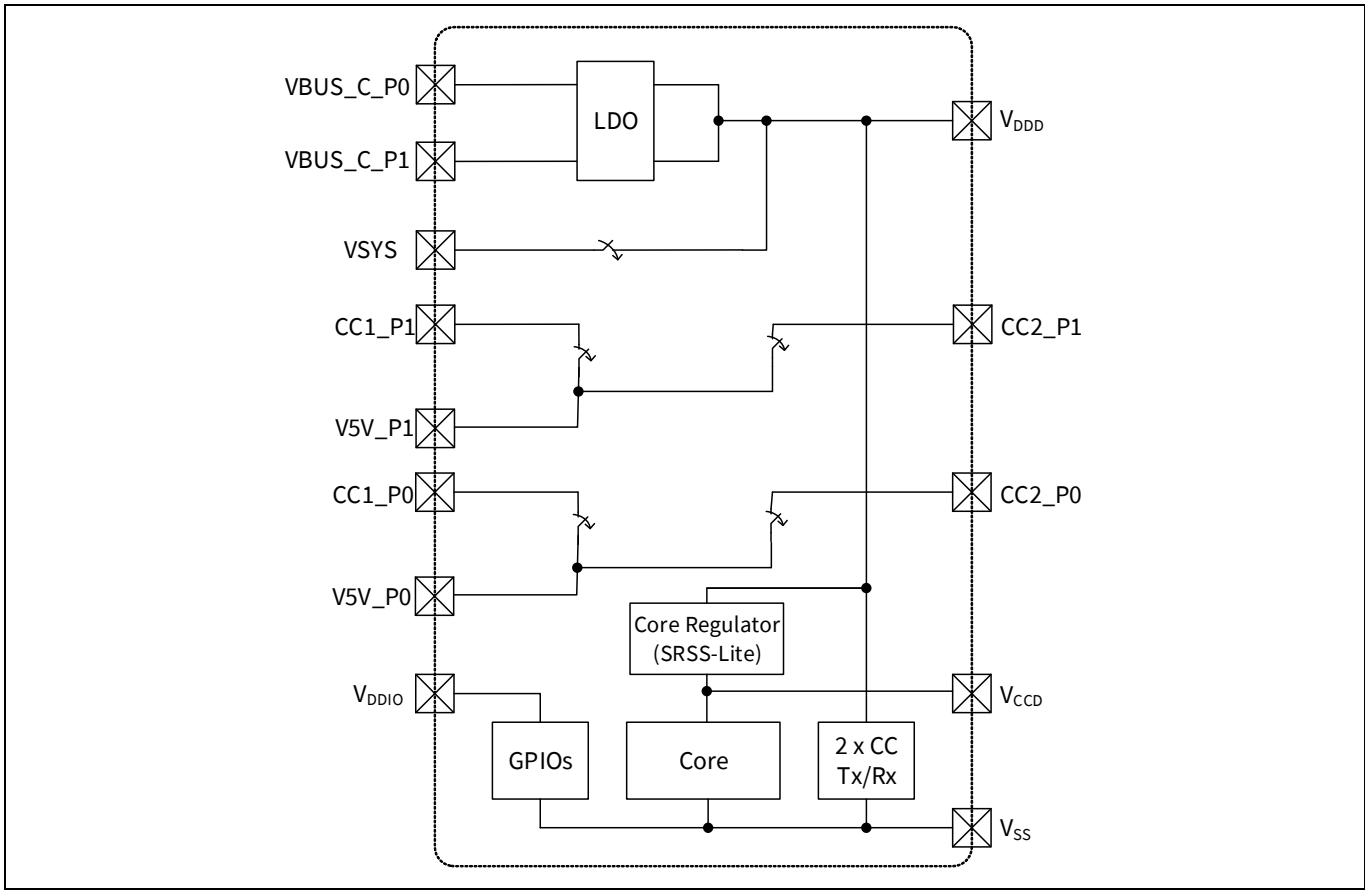



**Figure 11-1. EZ-PD™ PMG1-S3 MCU power system block diagram**

**Figure 11-1** shows an overview of the EZ-PD™ PMG1-S3 MCU power system. EZ-PD™ PMG1-S3 MCU can operate from three possible external supply sources, VSYS and VBUS_C_P0/1. The VSYS input supports operation over 2.75 – 5.5 V. EZ-PD™ PMG1-S3 MCU family of devices have three different power modes: Active, Sleep, and Deep Sleep, transitions between which are managed by the Power System. A separate power domain VDDIO is provided for the GPIOs and VDDD which generates 3.3 V from internal regulator. VDDD can be shorted to VDDIO. The VCCD pin, the output of the core (1.8 V) regulator, is brought out for connecting a 0.1-µF capacitor for the regulator stability only. This pin is not supported as a power supply.

The VCONN_Source supply is used to power Type-C cable markers through the VCONN switches and supports operation in the range 4.85 V to 5.5 V. EZ-PD™ PMG1-S3 MCU cannot be powered through this supply.

## 11.2 How it works

The regulators in **Figure 11-1** power the various domains of the device. All the core regulators and digital I/Os draw their input power from the VDDD pin supply. Digital I/Os are supplied from VDDD. The VDDIO pin, provides a separate voltage domain for the GPIOs. See the **device datasheet** for details.

### 11.2.1 Regulator summary

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

**Table 11-1. Regulator status in different power modes**

| Mode | Active regulator | Quiet regulator |
|---|---|---|
| Deep Sleep | Off | Off |
| Sleep | On | On |
| Active | On | On |

#### 11.2.1.1 Active digital regulator

For external supplies from 2.75 V to 5.5 V, the Active digital regulator provides the main digital logic in Active and Sleep modes. This regulator has its output connected to a pin (VCCD) and requires an external decoupling capacitor (0.1 µF X5R).

#### 11.2.1.2 Quiet regulator

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

#### 11.2.1.3 Deep-Sleep regulator

This regulator supplies the circuits that remain powered in Deep-Sleep mode, such as the ILO and SCB. The Deep-Sleep regulator is available in all power modes. In Active and Sleep power modes, the main output of this regulator is connected to the output of the digital regulator (VCCD). This regulator also has a separate replica output that provides a stable voltage for the ILO. This output is not connected to VCCD in Active and Sleep modes. In deep sleep mode, the VDDD regulator is always ON. The VDDD regulator either takes VSYS or VBUS supply (with priority to VSYS).

## 11.3 Voltage monitoring

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

### 11.3.1 Power-on-reset

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor VCCD voltage. Typically, the POR circuits are not very accurate with respect to trip-point. POR circuits are used during initial device power-up and then disabled.

#### 11.3.1.1 Brownout-detect

The BOD circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. BOD circuit monitors the VCCD voltage. The BOD circuit generates a reset if a voltage excursion dips below the minimum VCCD voltage required for safe operation (see the **device datasheet** for details). The system will not come out of RESET until the supply is detected to be valid again. To ensure the reliable operation of the device, the watchdog timer should be used in all designs. Watchdog timer provides protection against abnormal brownout conditions that may compromise the CPU functionality. See the **“Watchdog timer”** on page 99 for more details.

## 11.4 Registers

**Table 11-2. Power supply and monitoring registers**

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

# 12 Chip operational modes

EZ-PD™ PMG1-S3 MCU can execute firmware in four different modes. These modes dictate execution from different locations in Flash and ROM, with different levels of hardware privileges. Only three of these modes are used in end-applications; debug mode is used exclusively to debug designs during firmware development.

EZ-PD™ PMG1-S3 MCU's operational modes are:

- Boot
- User
- Privileged
- Debug

## 12.1 Boot

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

## 12.2 User

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

## 12.3 Privileged

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

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see **"Performing a system call"** on page 268. Exit from this mode returns the device to user mode.

## 12.4 Debug

Debug mode is an operational mode that allows observation of the EZ-PD™ PMG1-S3 MCU operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as ModusToolbox™ software and Arm® MDK to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the **"Program and debug Interface"** on page 257.

For more details on protection modes, see the **"Device security"** on page 104.

# 13 Power modes

The EZ-PD™ PMG1-S3 MCU provides a number of power modes, intended at minimizing the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep-Sleep

Active, Sleep, and Deep-Sleep are standard Arm®-defined power modes, supported by the Arm® CPUs and instruction set architecture (ISA).

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

- Enabling/disabling clocks to peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the EZ-PD™ PMG1-S3 MCU

**Figure 13-1** illustrates the various power modes and the possible transitions between them.

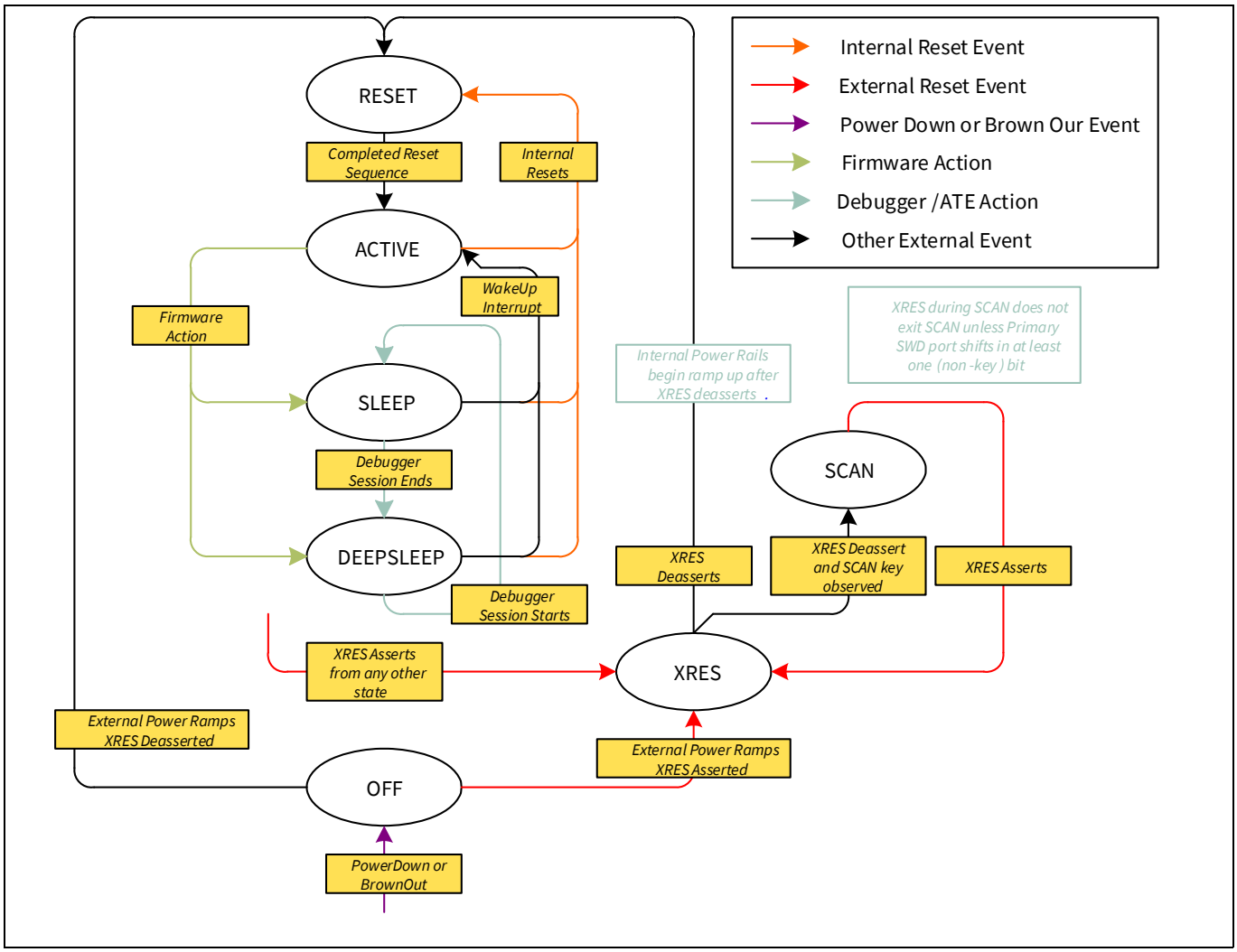



**Figure 13-1. Power mode transitions state diagram**

### Power modes

Table 13-1 illustrates the power modes offered by EZ-PD™ PMG1-S3 MCU

**Table 13-1. EZ-PD™ PMG1-S3 MCU power modes**

| Power mode | Description | Entry condition | Wakeup sources | Active clocks | Wakeup action | Available regulators |
|---|---|---|---|---|---|---|
| Active | Primary mode of operation; all peripherals are available (programmable). | Wakeup from other power modes, internal and external resets, brownout, power on reset | Not applicable | Any (programmable) | Not applicable | All regulators are available. The active digital regulator can be disabled if external regulation is used. |
| Sleep | CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable). | Manual register write | Any interrupt | Any (programmable) | Interrupt | All regulators are available. The active digital regulator can be disabled if external regulation is used. |
| Deep-Sleep | All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency (~32 kHz) clock is available. | Manual register write | GPIO interrupt, SCB, LPCOMP, watchdog timer and USB PD | ILO (~32 kHz) | Interrupt | Deep-Sleep regulator |

In addition to the wakeup sources mentioned in Table 13-1, external reset (XRES) and brownout reset bring the device to Active mode from any power mode.

## 13.1 Active mode

Active mode is the primary power mode of the EZ-PD™ PMG1-S3 MCU device. This mode provides the option to use every possible subsystem/peripheral in the device. In this mode, the CPU is running and all the peripherals are powered. The firmware may be configured to disable specific peripherals that are not in use to reduce power consumption.

## 13.2 Sleep mode

This is a CPU-centric power mode. In this mode, the Cortex®-M0+ CPU enters Sleep mode and its clock is disabled. It is a mode that the EZ-PD™ PMG1-S3 MCU can use to accomplish low power consumption in states where Deep-Sleep mode cannot be used. It is identical to Active mode from a peripheral point of view. Any enabled interrupt can cause wakeup from Sleep mode.

## 13.3 Deep-Sleep mode

In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The MHz range clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (32 kHz) oscillator remains on and low-frequency peripherals continue to operate. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, $I^2C$ slave) continue to operate. Interrupts from low-speed, asynchronous or USB PD can cause a wakeup from Deep-Sleep mode.

The available wakeup sources are listed in **Table 13-3**.

## 13.4 Power mode summary

**Table 13-2** illustrates the peripherals available in each low-power mode; **Table 13-3** illustrates the wakeup sources available in each power mode.

**Table 13-2. Available peripherals**

| Peripheral | Active | Sleep | Deep-Sleep |
|---|---|---|---|
| CPU | On | Retention[a)] | Retention |
| SRAM | On | Retention | Retention |
| High-speed peripherals | On | On | Retention |
| Low-speed peripherals | On | On | On (optional) |
| Internal main oscillator (IMO) | On | On | Off |
| Internal low-speed oscillator (ILO, 32 kHz) | On | On | On (optional) |
| Asynchronous peripherals | On | On | On |
| Power-on-reset, brownout detection | On | On | On |
| Regular analog peripherals | On | On | Off |
| GPIO output state | On | On | On |

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

**Table 13-3. Wakeup sources**

| Power mode | Wakeup source | Wakeup action |
|---|---|---|
| Sleep | Any interrupt source | Interrupt |
| | Any reset source | Reset |
| Deep-Sleep | GPIO interrupt | Interrupt |
| | I2C address match | Interrupt |
| | Watchdog timer | Interrupt/Reset |
| | USB PD | Interrupt |
| | XRES (external reset pin)[a)], Brownout | Reset |

a) XRES triggers a full system restart. All the states including frozen GPIOs are lost. In this case, the cause of wakeup is not readable after the device restarts.

## 13.5 Low-power mode entry and exit

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

The transition to Sleep and Deep-Sleep modes are controlled by the flags SLEEPDEEP in the CM0+ System Control Register (CM0P_SCR):

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

The LPM READY bit in the PWR_CONTROL register shows the status of Deep-Sleep regulator. If the firmware tries to enter Deep-Sleep mode before the regulators are ready, then EZ-PD™ PMG1-S3 MCU goes to Sleep mode first; when the regulators are ready, the device enters Deep-Sleep mode. This operation is automatically done in hardware.

In Sleep and Deep-Sleep modes, a selection of peripherals are available (see **Table 13-3**), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode.

## 13.6 Registers

**Table 13-4. Power mode registers**

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

# 14 Watchdog timer

The watchdog timer (WDT) is used to automatically reset the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality. The WDT runs from the LFCLK (32-kHz clock), generated by the ILO. The timer must be serviced periodically in firmware to avoid a reset. Otherwise, the timer will elapse and generate a device reset. The WDT can be used as an interrupt source or a wakeup source in low-power modes.

## 14.1 Features

The WDT has these features:

- Configurable timer period
- Can generate an interrupt in Sleep and Deep-Sleep power modes to wake up the device
- Can generate an interrupt in Active mode after a specified interval

## 14.2 Block diagram

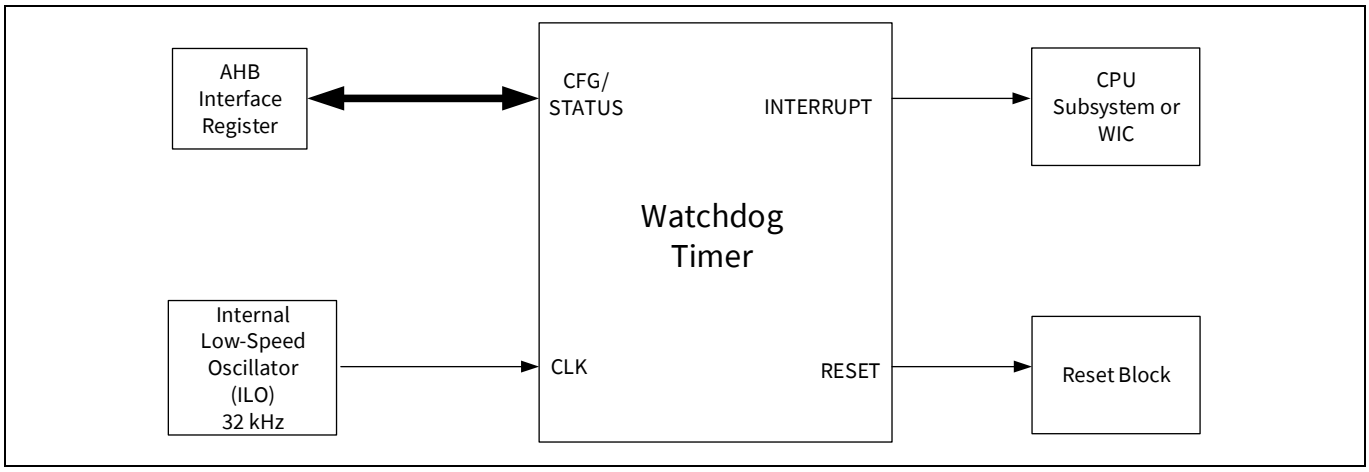



**Figure 14-1. Watchdog timer block diagram**

## 14.3 How it works

The WDT asserts a hardware reset to the device after three WDT interrupts each of programmable interval of up to 2048 ms, unless it is periodically serviced in firmware. The WDT is a 16-bit free-running wraparound up-counter.

The WDT_COUNTER register provides the count value of the WDT. The WDT generates an interrupt when the count value in WDT_COUNTER equals the match value stored in the WDT_MATCH register, but it does not reset the count to '0'. Instead, the WDT keeps counting until it overflows and rolls back to 0. When the count value reaches the match value again, another interrupt is generated.

A bit named WDT_MATCH in the SRSS_INTR register is set whenever the WDT interrupt occurs. This interrupt must be cleared by writing a '1' to the WDT_MATCH bit in SRSS_INTR to feed the watchdog timer. If the firmware does not feed the WDT for two consecutive interrupts, the third match event will generate a hardware reset. For details, see the WDT_COUNTER, WDT_MATCH, and SRSS_INTR registers in the EZ-PD™ PMG1-S3 MCU registers TRM.

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

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

- Feed the watchdog by clearing the interrupt bit regularly in the main body of the firmware code.
- Guarantee that the interrupt is cleared at least once every WDT period.
- Use the WDT ISR only as a timer to trigger certain actions and to change the next WDT_MATCH value. Do not feed WDT in this ISR.

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

1. Write the desired match value to the WDT_MATCH register.
2. Clear the WDT_MATCH bit in SRSS_INTR to clear any pending WDT interrupt.
3. Enable the WDT interrupt by setting the WDT_MATCH bit in SRSS_INTR_MASK.
4. In the ISR, clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.
5. The IGNORE_BITS in the WDT_MATCH register can be used to reduce the entire WDT counter period. The ignore bits can specify the number of MSbs that needs to be discarded. For example, if the IGNORE_BITS value is 3, then WDT counter becomes a 13-bit counter.

### 14.3.1 Enabling and disabling WDT

The WDT is a free-running counter that cannot be disabled. However, it is possible to disable the WDT reset by writing a key '0xACED8865' to the WDT_DISABLE_KEY register. Writing any other value to this register will enable the WDT reset. If WDT reset is disabled, the firmware does not need to periodically feed the WDT to avoid a reset. The WDT can still be used as an interrupt source or wakeup source. The only way to stop WDT from generating interrupts and wakeup events is to disable the ILO by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDT reset will automatically enable the ILO.

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

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

### 14.3.2 WDT interrupts and low-power modes

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

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

For more details, see the **"Power modes"** on page 95.

### 14.3.3 WDT reset mode

The RESET_WDT bit in the RES_CAUSE register indicates the reset generated by the WDT. This bit remains set until cleared or until a POR, BOD, or XRES occurs. All other resets leave this bit untouched.

For more details, see the **"Reset system"** on page 102.

## 14.4 Registers

**Table 14-1. WDT registers**

| Register name | Description |
|---|---|
| WDT_DISABLE_KEY | Disables the WDT when 0XACED8865 is written, for any other value WDT works normally |
| WDT_COUNTER | Provides the count value of the WDT |
| WDT_MATCH | Stores the match value of the WDT |
| SRSS_INTR | Feeds the WDT to avoid reset |

# 15 Reset system

EZ-PD™ PMG1-S3 MCU supports several types of resets that guarantee error-free operation during power up and allow the device to reset based on user-supplied external hardware or internal software reset signals. EZ-PD™ PMG1-S3 MCU also contains hardware to enable the detection of certain resets.

The reset system has these sources:

- Power-on reset (POR) to hold the device in reset while the power supply ramps up
- Brownout reset (BOD) to reset the device if the power supply falls below specifications during operation
- Watchdog reset (WRES) to reset the device if firmware execution fails to service the watchdog timer
- Software initiated reset (SRES) to reset the device on demand using firmware
- External reset (XRES) to reset the device using an electrical signal external to the EZ-PD™ PMG1-S3 MCU
- Protection fault reset (PROT_FAULT) to reset the device if unauthorized operating conditions occur

## 15.1 Reset sources

The following sections provide a description of the reset sources available in EZ-PD™ PMG1-S3 MCU.

### 15.1.1 Power-on reset

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

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

### 15.1.2 Brownout reset

Brownout reset monitors the digital voltage supply VCCD and generates a reset if VCCD is below the minimum logic operating voltage specified in the **device datasheet**. BOD is available in all power modes and cannot be distinguished from a POR or XRES reset.

### 15.1.3 Watchdog reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. Watchdog reset is enabled by default. To disable it, refer to **“Enabling and disabling WDT”** on page 100.

The RESET_WDT status bit of the RES_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched. For more details, see the **“Watchdog timer”** on page 99.

### 15.1.4 Software initiated reset

Software initiated reset (SRES) is a mechanism that allows a software-driven reset. The Cortex®-M0+ application interrupt and reset control register (CM0P_AIRCR) forces a device reset when a ‘1’ is written into the SYSRESETREQ bit. CM0P_AIRCR requires a value of 0x05FA written to the top two bytes for writes. Therefore, write 0x05FA0004 to trigger the reset.

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

### 15.1.5 External reset

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

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

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

### 15.1.6 Protection fault reset

Protection fault reset (PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code. For details about privilege code, see **"Privileged"** on page 94.

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

## 15.2 Identifying reset sources

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

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

## 15.3 Registers

**Table 15-1. Reset system registers**

| Register name | Description |
|---|---|
| CM0P_AIRCR | Cortex®-M0+ Application Interrupt and Reset Control Register – This register allows initiation of software resets, among other Cortex®-M0+ functions. |
| RES_CAUSE | Reset Cause Register – This register captures the cause of recent resets. |

# 16 Device security

EZ-PD™ PMG1-S3 MCU offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and robust flash protection provide a high level of security.

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

*Note: Because all programming, debug, and test interfaces are disabled when maximum device security is enabled, EZ-PD™ PMG1-S3 MCU devices with full device security enabled may not be returned for failure analysis.*

## 16.1 Features

The EZ-PD™ PMG1-S3 MCU device security system has the following features:

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

## 16.2 How it works

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

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

# 17 Trigger multiplexer block

Select peripherals in the EZ-PD™ PMG1-S3 MCU are interconnected using trigger signals. Trigger signals are means by which peripherals denote an occurrence of an event or a state. These triggers are used as means to affect or initiate some action in other peripherals. The trigger multiplexer block helps to route triggers from a source peripheral block to a destination.

## 17.1 Features

The trigger multiplex block has these features:

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

## 17.2 Architecture

The trigger signals in the EZ-PD™ PMG1-S3 MCU are digital signals generated by peripheral blocks to denote a state such as TCPWM overflow, or an event such as the completion of an action. These trigger signals typically serve as initiator of other actions in other peripheral blocks. An example is chaining two TCPWMs together in order to make a 32-bit counter instead of a 16-bit counter. This can be done by using a counter overflow trigger to then trigger a second TCPWM's count input. This can be seen in **Figure 17-1**.

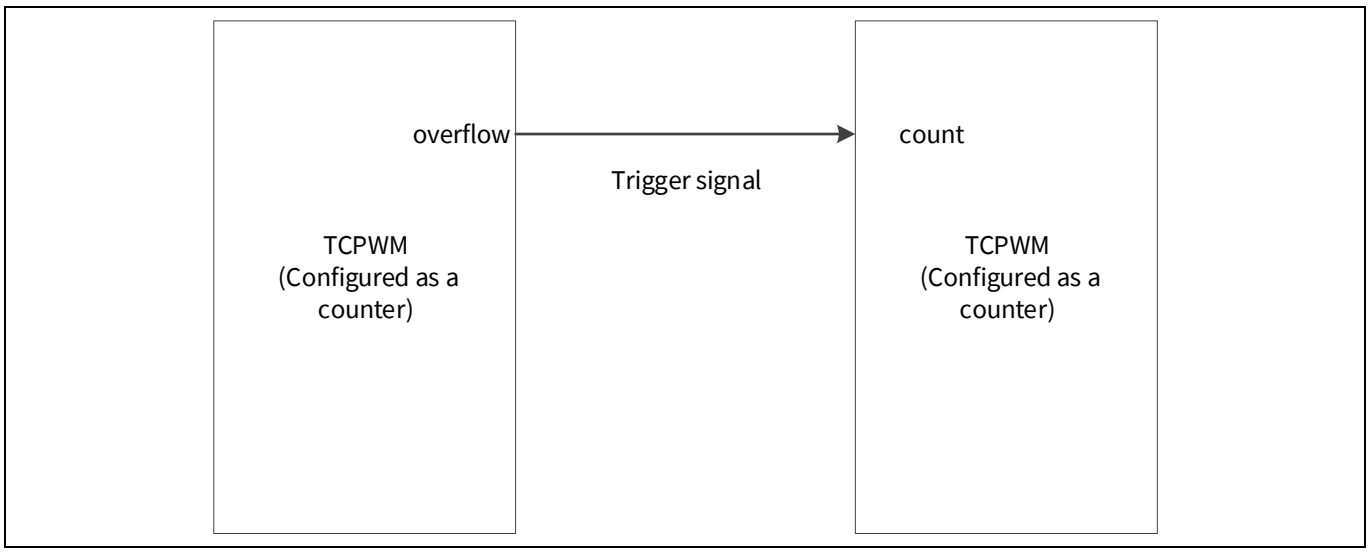



**Figure 17-1. Trigger signal example**

To support trigger routing the EZ-PD™ PMG1-S3 MCU has hardware, which is a series of multiplexers used to route the trigger signals from potential sources to destinations. This hardware is called the trigger multiplexer block. The trigger multiplexer can connect to a trigger signal emanating out of a peripheral block in the EZ-PD™ PMG1-S3 MCU and route it to a different peripheral to initiate or affect an operation at the destination peripheral block. There are two types of triggers, level sensitive triggers and rising edge triggers. Rising edge triggers should remain '1' for at least 2 "clk_sys" cycles.

### 17.2.1 Trigger multiplexer group

The trigger multiplexer block is implemented using several trigger multiplexers. A trigger multiplexer selects a signal from a set of trigger output signals from different peripheral blocks to route it to a specific trigger input of another peripheral block. The multiplexers are grouped into a trigger group. All the trigger multiplexers in a trigger group have similar input options and are designed to feed similar destination signals. Hence the trigger group can be considered as a block that multiplexes multiple inputs to multiple outputs. This concept is illustrated in **Figure 17-2**.

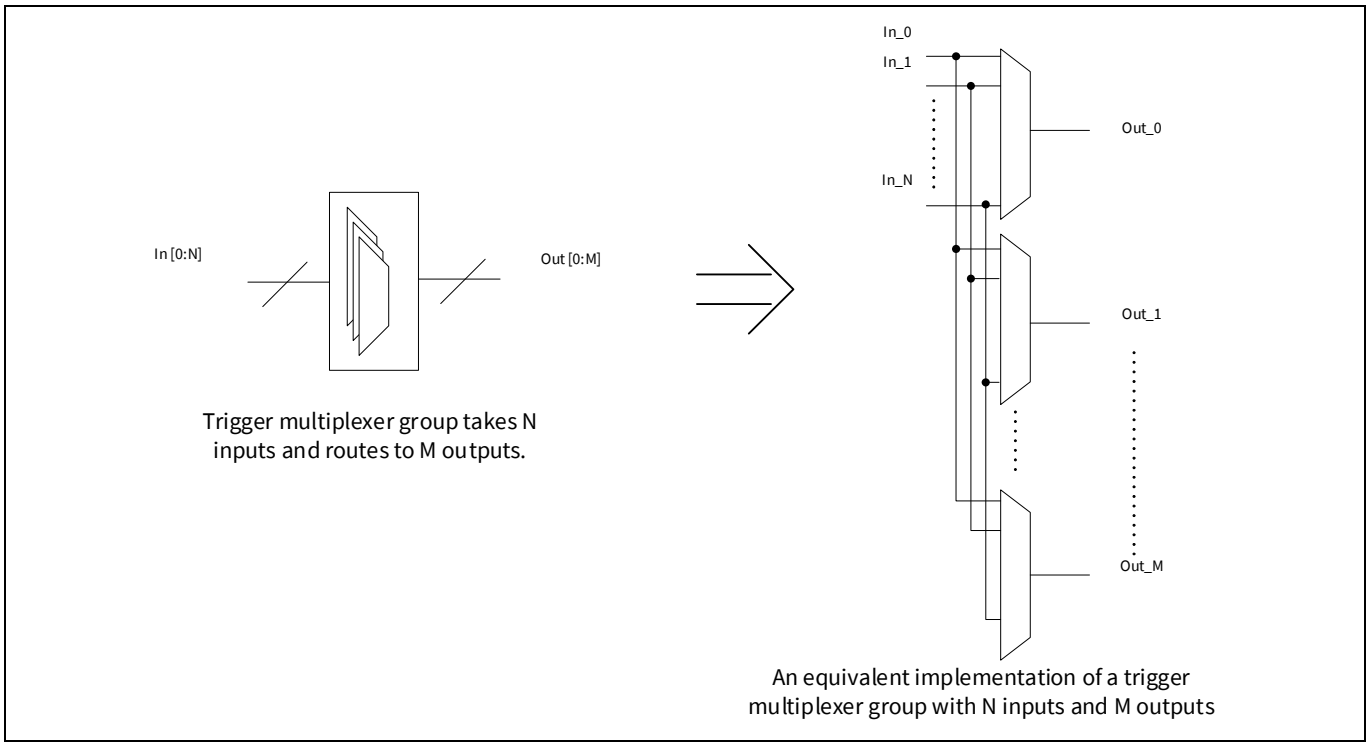



**Figure 17-2. Trigger multiplexer groups**

### 17.2.2 Software triggers

All input and output signals to a trigger multiplexer can be triggered from software. This is accomplished by writing into the PERI_TR_CTL register. This register allows you to trigger the corresponding signal for a number of peripheral clock cycles. The PERI_TR_CTL[TR_GROUP] bitfield selects the trigger group of the signal being activated. The PERI_TR_CTL[TR_OUT] bitfield determines whether the trigger signal is in output or input of the multiplexer. PERI_TR_CTL[TR_SEL] selects the specific line in the trigger group. The PERI_TR_CTL[TR_COUNT] bitfield sets up the number of peripheral clocks the trigger will be activated. The PERI_TR_CTL[TR_ACT] bitfield is set to '1' to activate the trigger line specified. Hardware resets this bit after the trigger is deactivated after the number of cycles set by the PERI_TR_CTL[TR_COUNT].

## 17.3 Trigger sources and destinations

The triggers are grouped into five different trigger groups. Each group has its own set of trigger sources and destinations. Each trigger source can be routed to trigger destination available in same group. The setting in PERI_TR_GROUPxTR_OUT_CTLy register, where 'x' is the group number and 'y' is the trigger destination. **Table 17-1** and **Table 17-2** contains the list of trigger sources and destinations for available trigger groups.

**Table 17-1. EZ-PD™ PMG1-S3 MCU trigger group and sources**

| PERI_TR_GROUPx_TR_OUT_CTLy[5:0] | Trigger group 0 | Trigger group 1 | Trigger group 2 | Trigger group 3 | Trigger group 4 |
|---|---|---|---|---|---|
| 0 | Software Trigger | Software Trigger | Software Trigger | Software Trigger | Software Trigger |
| 1 | TCPWM 0 Overflow | USB FS Trigger 0 | DMA channel 8 trigger out | TCPWM 0 Overflow | TCPWM 0 Overflow |
| 2 | TCPWM 1 Overflow | USB FS Trigger 1 | DMA channel 9 trigger out | TCPWM 1 Overflow | TCPWM 1 Overflow |
| 3 | TCPWM 2 Overflow | USB FS Trigger 2 | DMA channel 10 trigger out | TCPWM 2 Overflow | TCPWM 2 Overflow |
| 4 | TCPWM 3 Overflow | USB FS Trigger 3 | DMA channel 11 trigger out | TCPWM 3 Overflow | TCPWM 3 Overflow |
| 5 | TCPWM 4 Overflow | USB FS Trigger 4 | DMA channel 12 rigger out | TCPWM 4 Overflow | TCPWM 4 Overflow |
| 6 | TCPWM 5 Overflow | USB FS Trigger 5 | DMA channel 13 trigger out | TCPWM 5 Overflow | TCPWM 5 Overflow |
| 7 | TCPWM 6 Overflow | USB FS Trigger 6 | DMA channel 14 trigger out | TCPWM 6 Overflow | TCPWM 6 Overflow |
| 8 | TCPWM 7 Overflow | USB FS Trigger 7 | DMA channel 15 trigger out | TCPWM 7 Overflow | TCPWM 7 Overflow |
| 9 | TCPWM 0 Compare | DMA channel 8 trigger out | ¾ | TCPWM 0 Compare | TCPWM 0 Compare |
| 10 | TCPWM 1 Compare | DMA channel 9 trigger out | ¾ | TCPWM 1 Compare | TCPWM 1 Compare |
| 11 | TCPWM 2 Compare | DMA channel 10 trigger out | ¾ | TCPWM 2 Compare | TCPWM 2 Compare |
| 12 | TCPWM 3 Compare | DMA channel 11 trigger out | ¾ | TCPWM 3 Compare | TCPWM 3 Compare |
| 13 | TCPWM 4 Compare | DMA channel 12 trigger out | ¾ | TCPWM 4 Compare | TCPWM 4 Compare |
| 14 | TCPWM 5 Compare | DMA channel 13 trigger out | ¾ | TCPWM 5 Compare | TCPWM 5 Compare |
| 15 | TCPWM 6 Compare | DMA channel 14 trigger out | ¾ | TCPWM 6 Compare | TCPWM 6 Compare |
| 16 | TCPWM 7 Compare | DMA channel 15 trigger out | ¾ | TCPWM 7 Compare | TCPWM 7 Compare |

**Table 17-1. EZ-PD™ PMG1-S3 MCU trigger group and sources** (continued)

| PERI_TR_GROUPx_TR_OUT_CTLy[5:0] | Trigger group 0 | Trigger group 1 | Trigger group 2 | Trigger group 3 | Trigger group 4 |
|---|---|---|---|---|---|
| 17 | TCPWM 0 Underflow | ¾ | ¾ | TCPWM 0 Underflow | TCPWM 0 Underflow |
| 18 | TCPWM 1 Underflow | ¾ | ¾ | TCPWM 1 Underflow | TCPWM 1 Underflow |
| 19 | TCPWM 2 Underflow | ¾ | ¾ | TCPWM 2 Underflow | TCPWM 2 Underflow |
| 20 | TCPWM 3 Underflow | ¾ | ¾ | TCPWM 3 Underflow | TCPWM 3 Underflow |
| 21 | TCPWM 4 Underflow | ¾ | ¾ | TCPWM 4 Underflow | TCPWM 4 Underflow |
| 22 | TCPWM 5 Underflow | ¾ | ¾ | TCPWM 5 Underflow | TCPWM 5 Underflow |
| 23 | TCPWM 6 Underflow | ¾ | ¾ | TCPWM 6 Underflow | TCPWM 6 Underflow |
| 24 | TCPWM 7 Underflow | ¾ | ¾ | TCPWM 7 Underflow | TCPWM 7 Underflow |
| 25 | SCB 0 TX Request | ¾ | ¾ | 12-bit SAR ADC Sample Done | TCPWM 0 line out |
| 26 | SCB 0 RX Request | ¾ | ¾ | 12-bit SAR ADC EOC | TCPWM 1 line out |
| 27 | SCB 1 TX Request | ¾ | ¾ | CTBm cmp0 | TCPWM 2 line out |
| 28 | SCB 1 RX Request | ¾ | ¾ | CTBm cmp1 | TCPWM 3 line out |
| 29 | SCB 2 TX Request | ¾ | ¾ | LPCOMP 0 output | TCPWM 4 line out |
| 30 | SCB 2 RX Request | ¾ | ¾ | LPCOMP 1 output | TCPWM 5 line out |
| 31 | SCB 3 TX Request | ¾ | ¾ | USBPD0 Trigger 0 | TCPWM 6 line out |
| 32 | SCB 3 RX Request | ¾ | ¾ | USBPD0 Trigger 1 | TCPWM 7 line out |
| 33 | SCB 4 TX Request | ¾ | ¾ | USBPD0 Trigger 2 | 12-bit SAR ADC Sample Done |
| 34 | SCB 4 RX Request | ¾ | ¾ | USBPD0 Trigger 3 | 12-bit SAR ADC EOC |
| 35 | SCB 2 TX Request | | | USBPD0 Trigger 4 | CTBm cmp0 |
| 36 | SCB 2 RX Request | ¾ | ¾ | USBPD0 Trigger 5 | CTBm cmp1 |
| 37 | SCB 3 TX Request | ¾ | ¾ | USBPD0 Trigger 6 | ¾ |
| 38 | SCB 3 RX Request | ¾ | ¾ | USBPD0 Trigger 0 | ¾ |
| 39 | SCB 4 TX Request | ¾ | ¾ | USBPD1 Trigger 1 | ¾ |
| 40 | SCB 4 RX Request | ¾ | ¾ | USBPD2 Trigger 2 | ¾ |
| 41 | DMA channel 0 trigger out | ¾ | ¾ | USBPD3 Trigger 3 | ¾ |

**Table 17-1. EZ-PD™ PMG1-S3 MCU trigger group and sources** (continued)

| PERI_TR_GROUPx_TR_OUT_CTLy[5:0] | Trigger group 0 | Trigger group 1 | Trigger group 2 | Trigger group 3 | Trigger group 4 |
|---|---|---|---|---|---|
| 42 | DMA channel 1 trigger out | ¾ | ¾ | USBPD4 Trigger 4 | ¾ |
| 43 | DMA channel 2 trigger out | ¾ | ¾ | USBPD5 Trigger 5 | ¾ |
| 44 | DMA channel 3 trigger out | ¾ | ¾ | USBPD6 Trigger 6 | ¾ |
| 45 | DMA channel 4 trigger out | ¾ | ¾ | ¾ | ¾ |
| 46 | DMA channel 5 trigger out | ¾ | ¾ | ¾ | ¾ |
| 47 | DMA channel 6 trigger out | ¾ | ¾ | ¾ | ¾ |
| 48 | DMA channel 7 trigger out | ¾ | ¾ | ¾ | ¾ |
| 49 | 12-bit SAR ADC Sample Done | ¾ | ¾ | ¾ | ¾ |
| 50 | 12-bit SAR ADC EOC | ¾ | ¾ | ¾ | ¾ |
| 51 | CTBm cmp0 | ¾ | ¾ | ¾ | ¾ |
| 52 | CTBm cmp1 | ¾ | ¾ | ¾ | ¾ |
| 53 | LPCOMP 0 output | ¾ | ¾ | ¾ | ¾ |
| 54 | LPCOMP 1 output | ¾ | ¾ | ¾ | ¾ |

**Table 17-2. EZ-PD™ PMG1-S3 MCU trigger group and destinations**

| Trigger destination | Trigger group 0 | Trigger group 1 | Trigger group 2 | Trigger group 3 | Trigger group 4 |
|---|---|---|---|---|---|
| 0 | DMA channel 0 trigger | DMA channel 8 trigger | USB FS Burst End 0 | TCPWM 8 trigger | 12-bit SAR ADC trigger |
| 1 | DMA channel 1 trigger | DMA channel 9 trigger | USB FS Burst End 1 | TCPWM 9 trigger | ¾ |
| 2 | DMA channel 2 trigger | DMA channel 10 trigger | USB FS Burst End 2 | TCPWM 10 trigger | ¾ |
| 3 | DMA channel 3 trigger | DMA channel 11 trigger | USB FS Burst End 3 | TCPWM 11 trigger | ¾ |
| 4 | DMA channel 4 trigger | DMA channel 12 trigger | USB FS Burst End 4 | TCPWM 12 trigger | ¾ |
| 5 | DMA channel 5 trigger | DMA channel 13 trigger | USB FS Burst End 5 | TCPWM 13 trigger | ¾ |

**Table 17-2. EZ-PD™ PMG1-S3 MCU trigger group and destinations** (continued)

| Trigger destination | Trigger group 0 | Trigger group 1 | Trigger group 2 | Trigger group 3 | Trigger group 4 |
|---|---|---|---|---|---|
| 6 | DMA channel 6 trigger | DMA channel 14 trigger | USB FS Burst End 6 | ¾ | ¾ |
| 7 | DMA channel 7 trigger | DMA channel 15 trigger | USB FS Burst End 7 | ¾ | ¾ |

## 17.4 Registers

**Table 17-3. Trigger multiplexer registers**

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

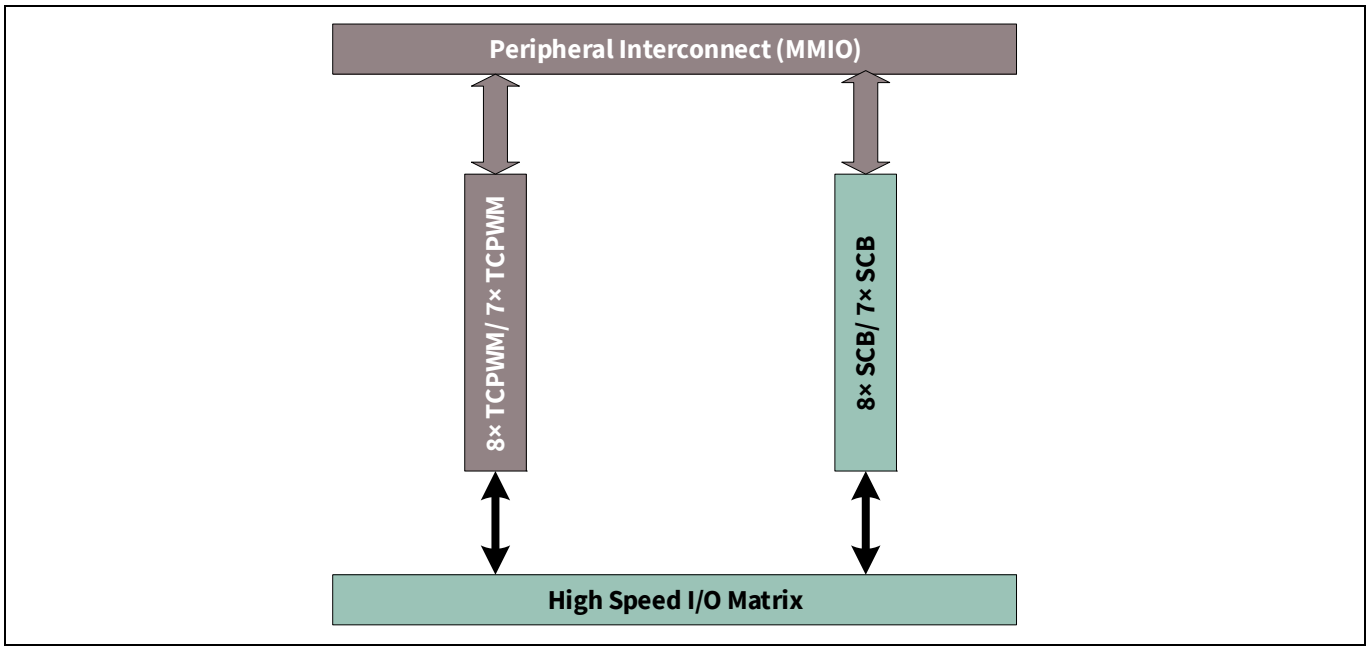
# Section E: Digital system

This section encompasses the following chapters:

- **"Serial Communications Block (SCB)"** on page 112
- **"Timer, Counter, and PWM"** on page 155

## Top level architecture

### Digital system block diagram



# 18 Serial Communications Block (SCB)

The Serial Communications Block (SCB) of EZ-PD™ PMG1-S3 MCU supports three serial interface protocols: SPI, UART, and I²C. Only one of the protocols is supported by an SCB at any given time. EZ-PD™ PMG1-S3 MCU 97-BGA has eight SCBs and 48-QFN has seven SCBs out of which five can be configured as SPI and UART. See the **device datasheet** for more details.

## 18.1 Features

This block supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, local interconnect network (LIN), and IrDA protocols
- Standard I²C master and slave functionality
- SPI and I²C EZ mode, which allows operating without CPU intervention
- Low-power (Deep-Sleep) power mode for SPI and I²C protocols (using external clocking)

Each of the three protocols is explained in the following sections.

## 18.2 Serial Peripheral Interface (SPI)

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

You can use the SPI master mode when the EZ-PD™ PMG1-S3 MCU has to communicate with one or more SPI slave devices. The SPI slave mode can be used when the EZ-PD™ PMG1-S3 MCU has to communicate with an SPI master device.

### 18.2.1 Features

- Supports master and slave functionality
- Supports three types of SPI protocols:
  - Motorola SPI – modes 0, 1, 2, and 3
  - TI SPI, with coinciding and preceding data frame indicator for mode 1
  - National (MicroWire) SPI for mode 0
- Data frame size programmable from 4 bits to 16 bits
- Interrupts or polling CPU interface
- Programmable oversampling
- Supports EZ mode of operation (**"Easy SPI (EZSPI) protocol"** on page 120 (applicable only for SPI slave functionality))
- Supports externally clocked slave operation:
  - In this mode, the slave operates in Active, Sleep, and Deep-Sleep system power modes
  - EZSPI mode allows for operation without CPU intervention

### 18.2.2 General description

**Figure 18-1** illustrates an example of SPI master with four slaves.

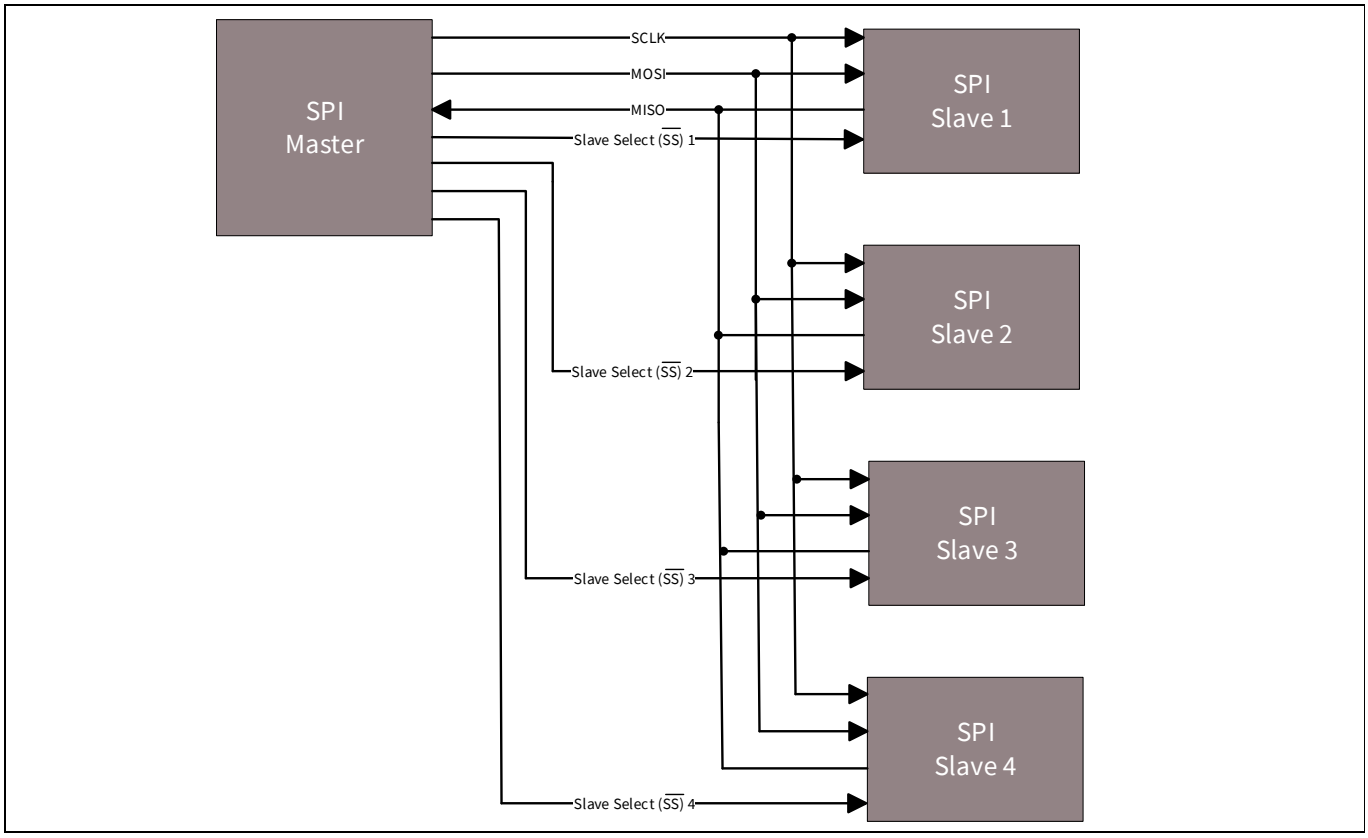



**Figure 18-1. SPI example**

A standard SPI interface consists of four signals as follows.

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

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

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

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

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

## 18.2.3 SPI modes of operation

### 18.2.3.1 Motorola SPI

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

#### Modes of Motorola SPI

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

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

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

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

**Figure 18-2** illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

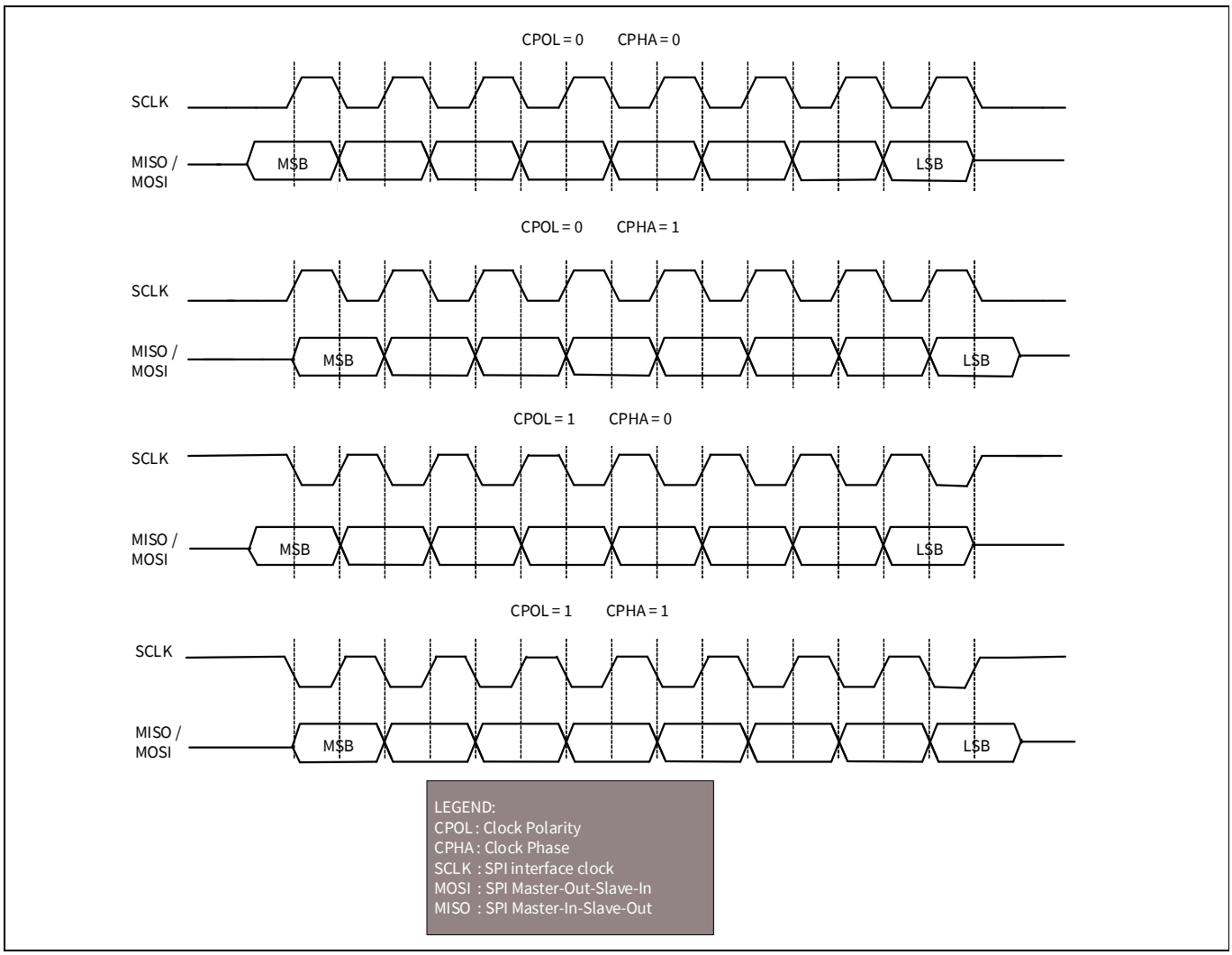



**Figure 18-2. SPI Motorola, 4 modes**

**Figure 18-3** illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

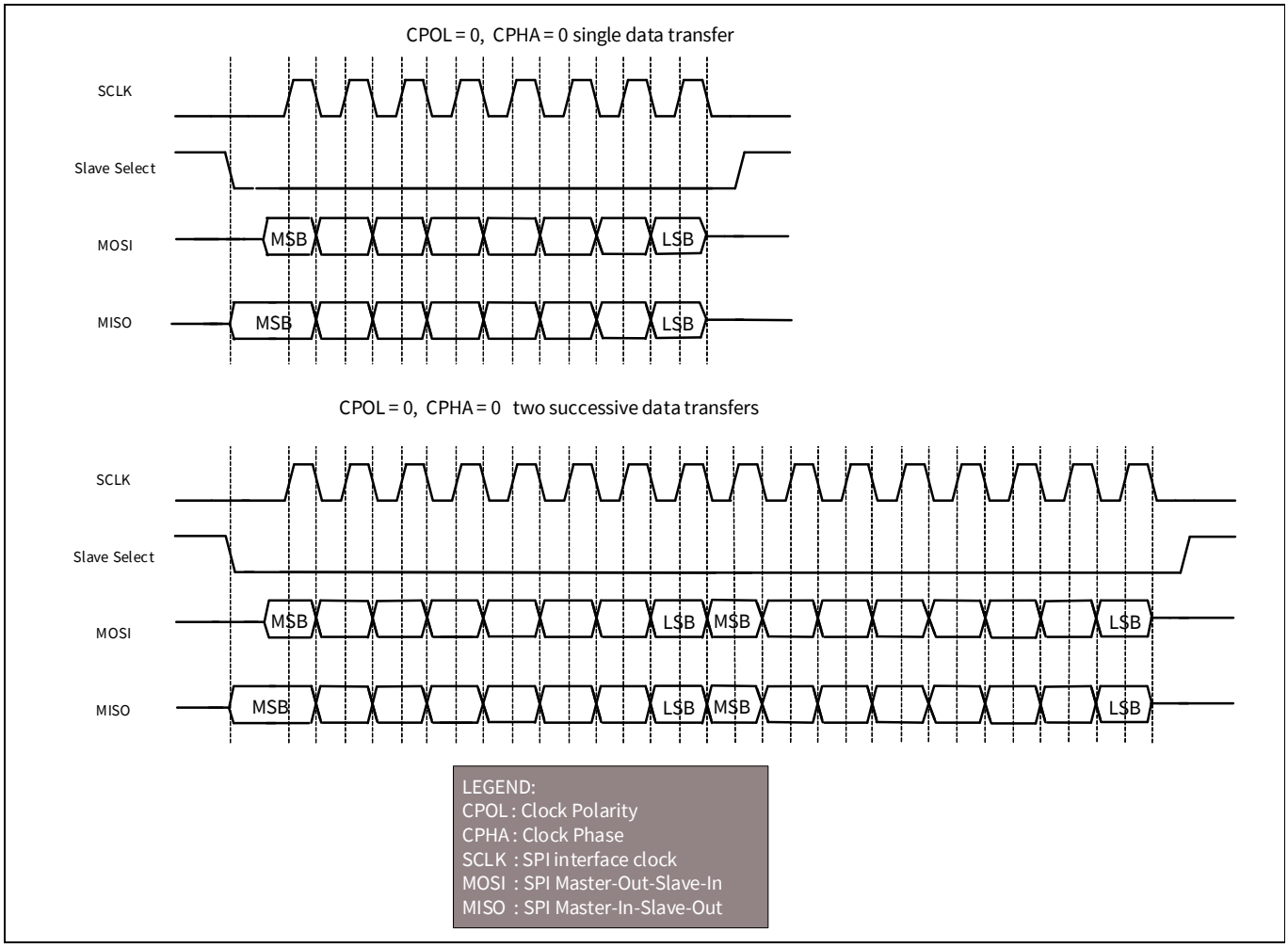



**Figure 18-3. SPI Motorola data transfer example**

### Configuring SCB for SPI Motorola mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI Motorola mode by writing '00' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the SCB_CPHA and SCB_CPOL fields (bits 2 and 3 respectively) of the SCBx_SPI_CTRL register.
4. Follow steps 2 to 4 from **“Enabling and initializing SPI”** on page 123.

For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

### 18.2.3.2 Texas Instruments SPI

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

**Figure 18-4** illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse precedes the first data bit. Note how the SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

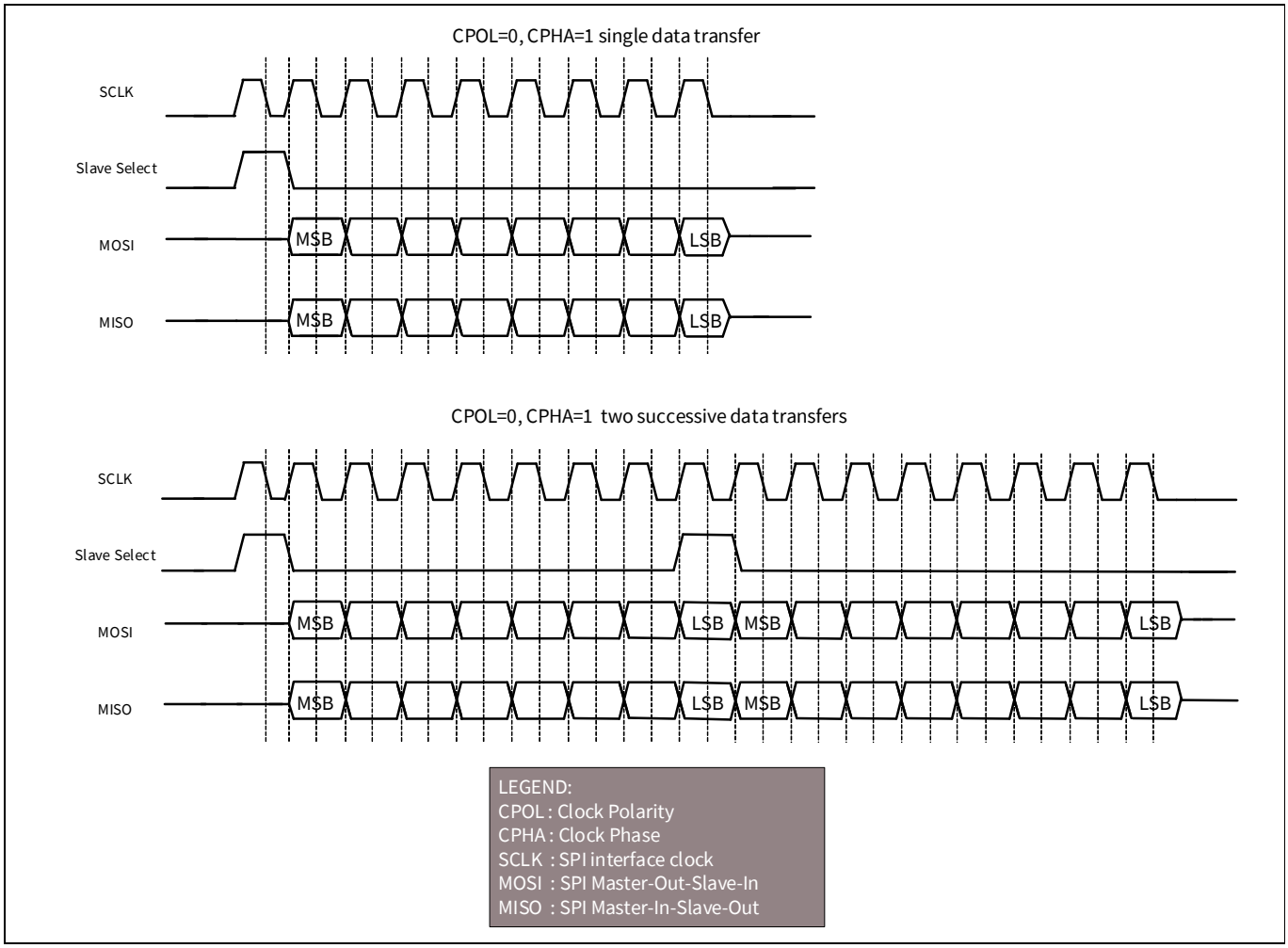



**Figure 18-4. SPI TI data transfer example**

**Figure 18-5** illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

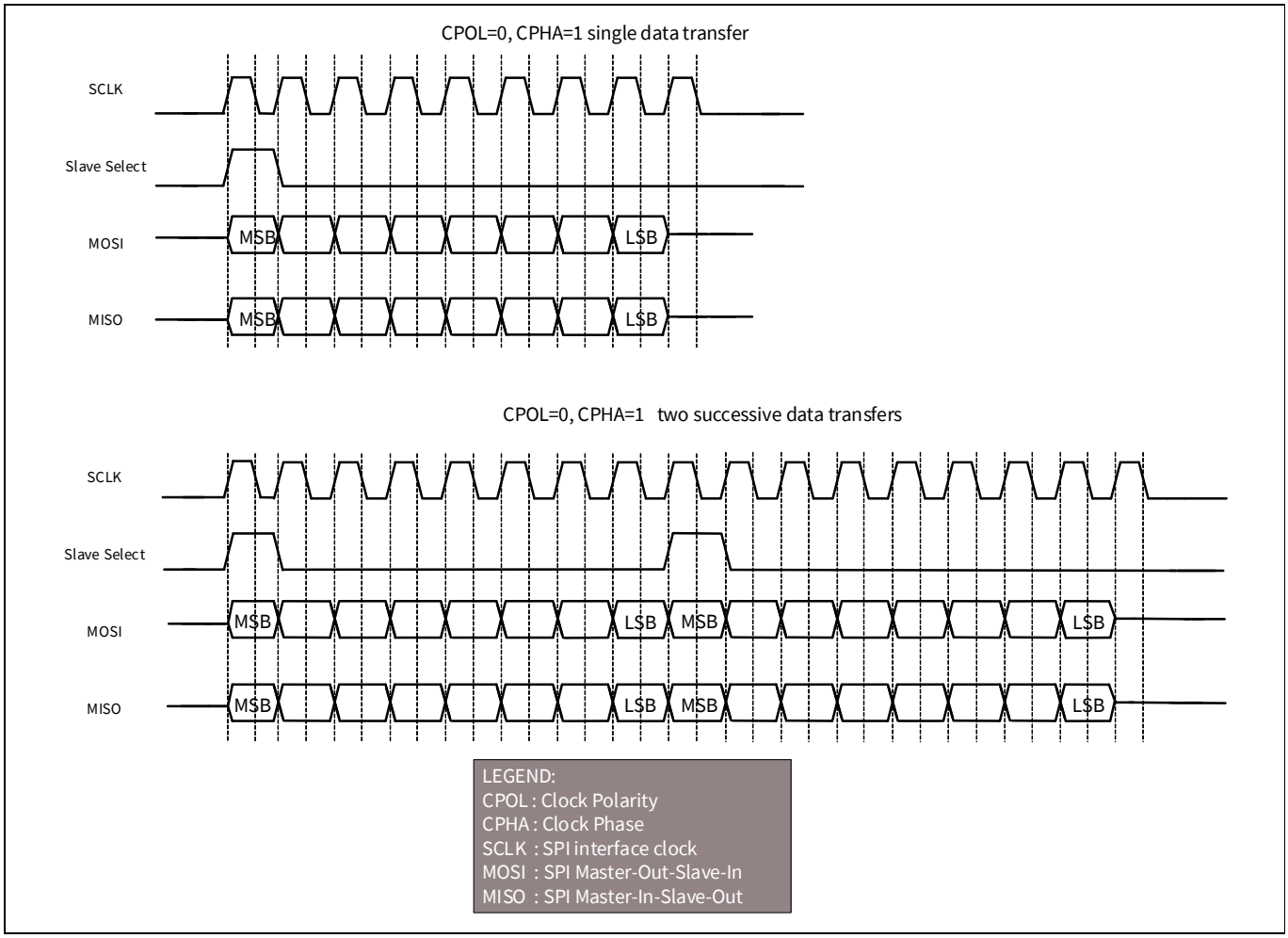



**Figure 18-5. SPI TI data transfer example**

### Configuring the SCB for SPI TI mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI TI mode by writing '01' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Select the mode of operation in TI by writing to the SELECT_PRECEDE field (bit 1) of the SCBx_SPI_CTRL register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 4 from **“Enabling and initializing SPI”** on page 123.

For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

### 18.2.3.3 National Semiconductors SPI

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

The National Semiconductors SPI protocol only supports mode 0 (CPOL is ‘0’ and CPHA is ‘0’): data is driven on a falling edge of SCLK and data is captured on a rising edge of SCLK.

**Figure 18-6** illustrates a single data transfer and two successive data transfers. In both cases, the transmission data transfer size is eight bits and the reception data transfer size is four bits.

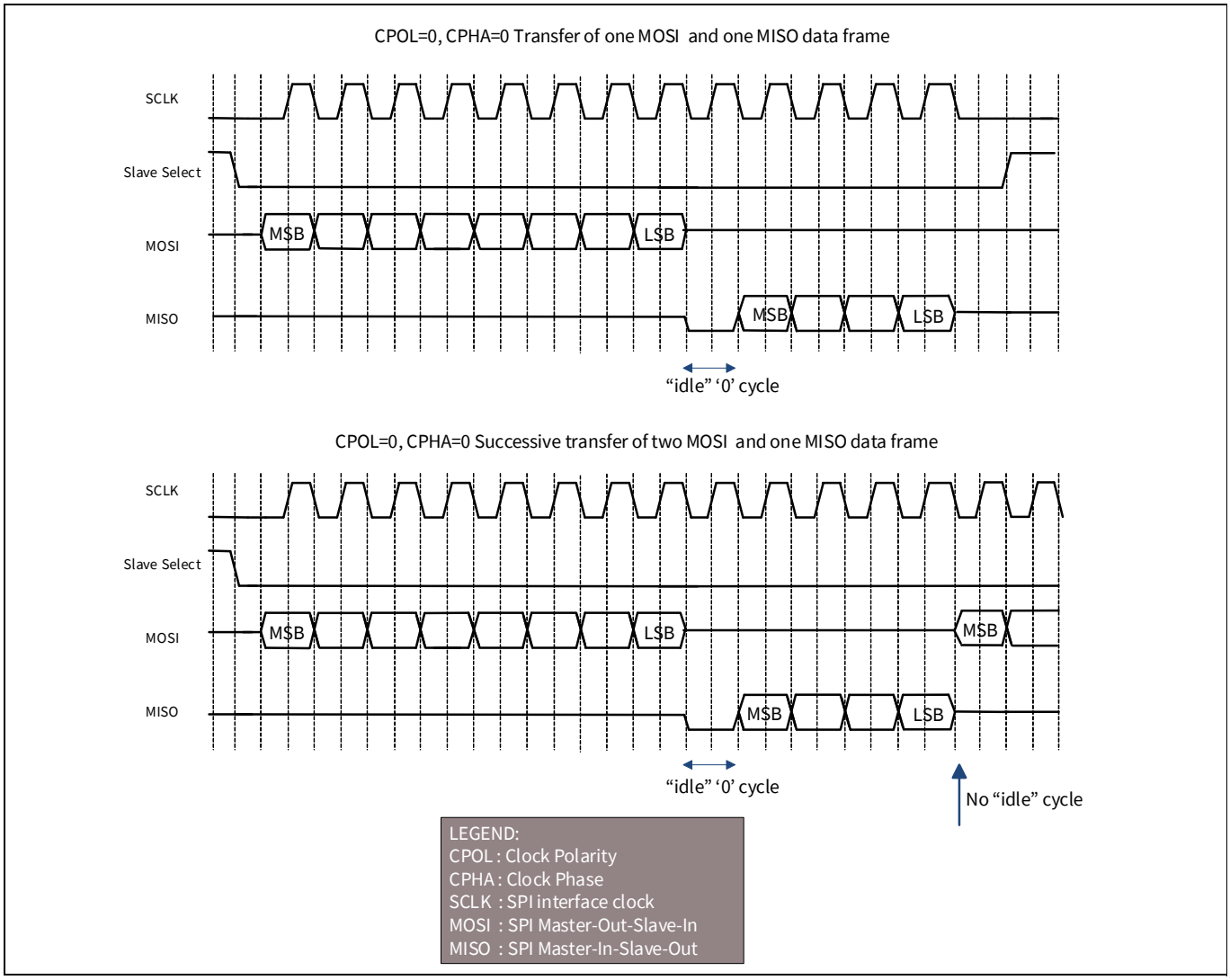



**Figure 18-6. SPI NS data transfer example**

#### Configuring the SCB for SPI NS mode

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

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Follow steps 2 to 4 from **“Enabling and initializing SPI”** on page 123.

For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

### 18.2.4 Easy SPI (EZSPI) protocol

The easy SPI (EZSPI) protocol is based on the Motorola SPI operating in mode 0 (CPOL is '0' and CPHA is '0'). EZSPI mode is applicable only for slave functionality in a single slave topology. It allows communication between a master and a single slave without the need for CPU intervention at the level of individual frames.

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

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

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

#### 18.2.4.1 EZ address write

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

#### 18.2.4.2 Memory array write

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

### 18.2.4.3 Memory array read

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

**Figure 18-7** illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.

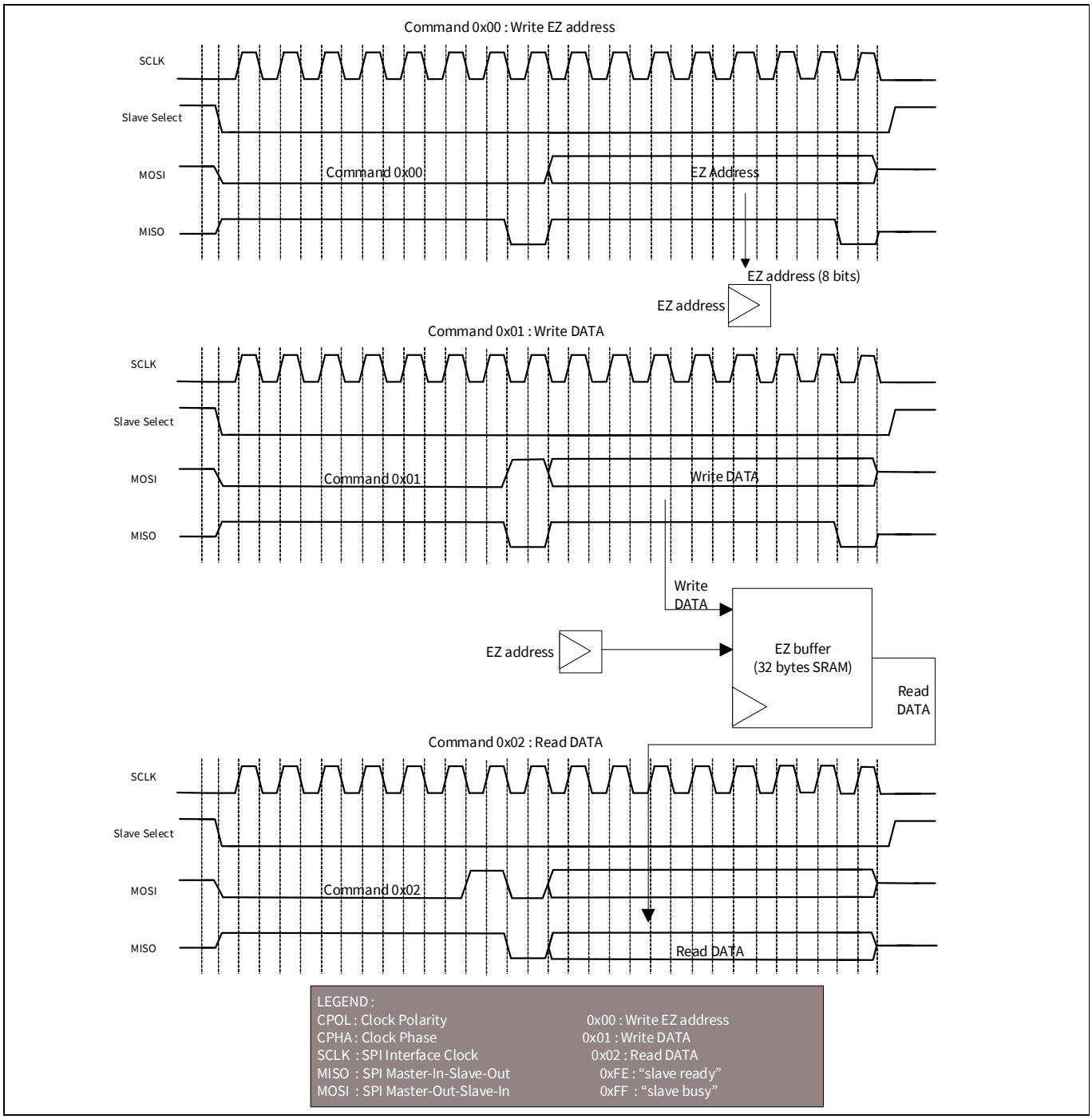



**Figure 18-7. EZSPI example**

#### 18.2.4.4 Configuring SCB for EZSPI mode

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

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCBx_CTRL register.
2. Follow steps 2 to 4 from **"Enabling and initializing SPI"** on page 123.
3. Use continuous transmission mode for transmitter by writing '1' to the CONTINUOUS bit of SCBx_SPI_CTRL register.
4. EZSPI mode is applicable only for slave functionality (write '0' to the MASTER_MODE field, bit 31 of SCBx_SPI_CTRL register).
5. Set the data frame width eight bits long (write '0111' to the DATA_WIDTH field, bits [3:0] of SCBx_TX_CTRL and SCBx_RX_CTRL registers).
6. Set the shift direction as MSB first (write '1' to the MSB_FIRST field, bit 8 of SCBx_TX_CTRL and SCBx_RX_CTRL registers).

For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

### 18.2.5 SPI registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in **Table 18-1**. For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

**Table 18-1. SPI registers**

| Register name | Operation |
|---|---|
| SCBx_CTRL | Enables the SCB, selects the type of serial interface (SPI, UART, I2C), and selects internally and externally clocked operation, EZ and non-EZ modes of operation. |
| SCBx_STATUS | In EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory. |
| SCBx_SPI_CTRL | Configures the SPI as either a master or a slave, selects SPI protocols (Motorola, TI, National) and clock-based submodes in Motorola SPI (modes 0,1,2,3), selects the type of SELECT signal in TI SPI. |
| SCBx_SPI_STATUS | Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode. |
| SCBx_TX_CTRL | Enables the transmitter, specifies the data frame width, and specifies whether MSB or LSB is the first bit in transmission. |
| SCBx_RX_CTRL | Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver. |
| SCBx_TX_FIFO_CTRL | Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO. |
| SCBx_RX_FIFO_CTRL | Performs the same function as that of the SCBx_TX_FIFO_CTRL register, but for the receiver. |
| SCBx_TX_FIFO_WR | Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation. |
| SCBx_RX_FIFO_RD | Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO – behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO. |
| SCBx_RX_FIFO_RD_SILENT | Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation. |

**Table 18-1. SPI registers** (continued)

| Register name | Operation |
|---|---|
| SCBx_TX_FIFO_STATUS | Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data. |
| SCBx_RX_FIFO_STATUS | Performs the same function as that of the SCBx_TX_FIFO_STATUS register, but for the receiver. |
| SCBx_EZ_DATAn | Holds the data in EZ memory location (n: 0–31) |

### 18.2.6 SPI interrupts

The SPI supports both internal and external interrupt requests. The internal interrupt events are listed here.

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

The SPI supports interrupts on the following events:

- SPI transfer done
- SPI is Idle
- TX FIFO is not full
- TX FIFO is empty
- SPI byte/word transfer is complete
- RX FIFO is empty
- RX FIFO is not empty
- Attempt to write to a full RX FIFO.
- RX FIFO is full

### 18.2.7 Enabling and initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCBx_SPI_CTRL register, according to **Table 18-2**. This includes selecting the submodes of the protocol and selecting master-slave functionality.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in **Table 18-3**.
   a) Specify the data frame width.
   b) Specify whether MSB or LSB is the first bit to be transmitted/received.
   c) Enable the transmitter and receiver.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers respectively, as shown in **Table 18-4**.
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
   c) Freeze the TX and RX FIFO.
4. Program the SCBx_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in **Table 18-5**.

**Table 18-2. SCBx_SPI_CTRL register**

| Bits | Name | Value | Description |
|---|---|---|---|
| [25:24] | MODE | 00 | SPI Motorola submode |
| | | 01 | SPI Texas Instruments submode |
| | | 10 | SPI National Semiconductors submode |
| | | 11 | Reserved |
| 31 | MASTER_MODE | 0 | Master mode |
| | | 1 | Slave mode |

**Table 18-3. SCBx_TX_CTRL/SCBx_RX_CTRL registers**

| Bits | Name | Description |
|---|---|---|
| [3:0] | DATA_ WIDTH | 'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits. |
| 8 | MSB_FIRST | 1= MSB first<br>0= LSB first |

**Table 18-4. SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL registers**

| Bits | Name | Description |
|---|---|---|
| [2:0] | TRIGGER_LEVEL | Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case. |
| 16 | CLEAR | When '1', the transmitter or receiver FIFO and the shift registers are cleared. |
| 17 | FREEZE | When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer. |

**Table 18-5. SCBx_CTRL register**

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

After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally to internally clocked operation (explained in **“Internally and externally clocked SPI operations”** on page 125). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCBx_CTRL register).

### 18.2.8 Internally and externally clocked SPI operations

The SCB supports both internally and externally clocked operations for SPI and $I^2C$ functions. An internally clocked operation uses a clock provided by the device. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation enables operation in the Deep-Sleep system power mode, in which an on-chip internal clock is provided to the block.

Internally clocked operation uses the high-frequency clock of the system. For more information on system clocking, see the **"Clocking system"** on page 85. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCBx_CTRL register specify the oversampling.

In SPI master mode, the valid range for oversampling is 4 to 16. Hence, the maximum bit rate is 12 Mbps. However, if you consider the I/O cell and routing delays, the effective oversampling range becomes 6 to 16. So, the maximum bit rate is 8 Mbps.

*Note: LATE_MISO_SAMPLE must be set to '1' in SPIM mode.*

In SPI slave mode, the oversampling field (bits [3:0]) of SCBx_CTRL register is not used. However, there is a frequency requirement for the SCB clock with respect to the interface clock (SCLK). This requirement is expressed in terms of the ratio (SCB clock/SCLK). This ratio is dependent on two fields: MEDIAN of SCBx_RX_CTRL register and LATE_MISO_SAMPLE of SCBx_CTRL register. With the MEDIAN bit set to '0' and LATE_MISO_SAMPLE bit set to '1', the SCB can achieve a maximum bit rate of 16 Mbps. However, if you consider the I/O cell and routing delays, the maximum data rate that can be achieved becomes 8 Mbps. Based on these bits, the maximum bit rates are given in **Table 18-6**.

**Table 18-6. SPI slave maximum data rates**

| Median of SCBx_RX_CTRL | LATE_MISO_SAMPLE of SCBx_CTRL | Ratio requirement | Maximum bit rate at peripheral clock of 48 MHz |
|---|---|---|---|
| 0 | 0 | ≥12 | 4 Mbps |
| 0 | 1 | ≥6 | 8 Mbps |
| 1 | 0 | ≥16 | 3 Mbps |
| 1 | 1 | ≥8 | 6 Mbps |

Externally clocked operation is limited to:

- Slave functionality.
- EZ functionality. EZ functionality uses the block's SRAM as a memory structure. Non-EZ functionality uses the block's SRAM as TX and RX FIFOs; FIFO support is not available in externally clocked operation.
- Motorola mode 0 (in the case of SPI slave functionality).

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

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

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

These two register fields determine the functional behavior of SPI. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. **Table 18-7** and **Table 18-8** describe the settings for SPI (in EZ and non-EZ mode).

### 18.2.8.1 Non-EZ mode of operation

In non-EZ mode there are two possible settings. As externally clocked operation is not supported for non-EZ functionality (no FIFO support), EC_OP_MODE should always be set to '0'. However, EC_AM_MODE can be set to '0' or '1'. **Table 18-7** gives an overview of the possibilities. The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

**Table 18-7. SPI Non-EZ mode**

| SPI, standard (non-EZ) mode | | | | |
|---|---|---|---|---|
| **System power mode** | **EC_OP_MODE = 0** | | **EC_OP_MODE = 1** | |
| | **EC_AM_MODE = 0** | **EC_AM_MODE = 1** | **EC_AM_MODE = 1** | **EC_AM_MODE=0** |
| Active and Sleep | Selection using internal clock. Operation using internal clock. | Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in the Sleep mode, the MASK bit can be configured by the user. After that, selection using internal clock. Operation using internal clock. | Not supported | Invalid configuration |
| Deep-Sleep | Not supported | Selection using external clock: Wakeup interrupt cause in enabled (MASK = 1). Generate 0xff bytes. | Not supported | |

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power modes and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active mode both are active and in Deep-Sleep mode only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active mode, the CPU and the block's internally clocked slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in the Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic takes care of the ongoing SPI transfer.
- In Deep-Sleep mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged (‘1’ bits or “0xFF” bytes are send out on the MISO line) and the internally clocked logic takes care of the next SPI transfer when it is woken up.

### 18.2.8.2 EZ Mode of operation

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. **Table 18-8** gives an overview of the possibilities. The grey cells indicate a possible, yet not recommended, setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

**Table 18-8. SPI EZ mode**

| SPI, EZ mode | | | | |
|---|---|---|---|---|
| **System power mode** | **EC_OP_MODE = 0** | | **EC_OP_MODE = 1** | |
| | **EC_AM_MODE = 0** | **EC_AM_MODE = 1** | **EC_AM_MODE = 1** | **EC_AM_MODE=0** |
| Active and Sleep | Selection using internal clock. Operation using internal clock. | Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in Sleep mode, the mask bit can be configured by the user. After that, selection using internal clock. Operation using internal clock. | Selection using external clock. Operation using external clock. | Invalid configuration |
| Deep-Sleep | Not supported | Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Generate 0xff bytes. | Selection using external clock. Operation using external clock. | |

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active system power mode. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active system power mode and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active mode both are active and in Deep-Sleep mode only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active mode, the CPU and the block's internally clocked slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic takes care of the ongoing SPI transfer.
- In Deep-Sleep mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged ('1' bits or “0xFF” bytes are send out on the MISO line) and the internally clocked logic takes care of the next SPI transfer when it is woken up.

EC_OP_MODE is '1' and EC_AM_MODE is '1': This setting works in Active and Deep-Sleep system power modes. The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK of the SCBx_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

## 18.3 UART

The Universal Asynchronous Receiver/Transmitter (UART) protocol is an asynchronous serial interface protocol. UART communication is typically point-to-point. The UART interface consists of two signals:

- TX: Transmitter output
- RX: Receiver input

### 18.3.1 Features

- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 1 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO 7816) reader.
  - IrDA
- Supports local interconnect network (LIN)
- Break detection
- Baud rate detection
- Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus).
- Multi-processor mode
- Data frame size programmable from 4 bits to 16 bits.
- Programmable number of STOP bits, which can be set to 1, 1.5, or 2 data bits
- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

### 18.3.2 General description

**Figure 18-8** illustrates a standard UART TX and RX.

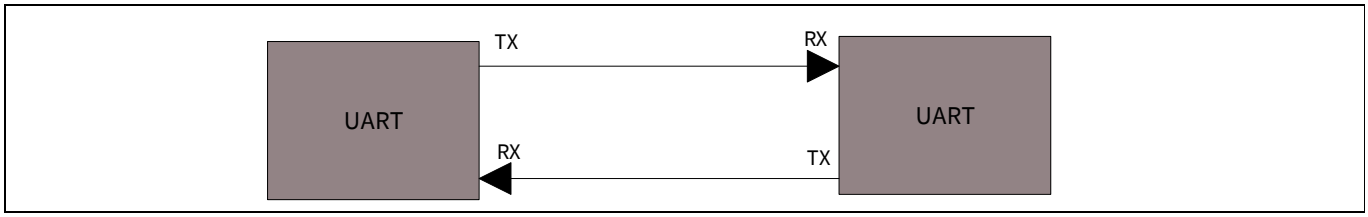



**Figure 18-8. UART example**

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

Three different serial interface protocols are supported:

- Standard UART protocol
- Multi-Processor mode
- LIN
- SmartCard, similar to UART, but with a possibility to send a negative acknowledgement
- IrDA, modification to the UART with a modulation scheme

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 3. The parity bit can be either enabled or disabled. If enabled, the type of parity can be set to either even parity or odd parity. The option of using the parity bit is available only in the Standard UART and SmartCard UART modes. For IrDA UART mode, the parity bit is automatically disabled. **Figure 18-9** depicts the default configuration of the UART interface of the SCB.

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

## 18.3.3 UART modes of operation

### 18.3.3.1 Standard protocol

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

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

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

The stop period or the amount of stop bits between successive data transfers is typically agreed upon between transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods. **Figure 18-9** illustrates the UART protocol.

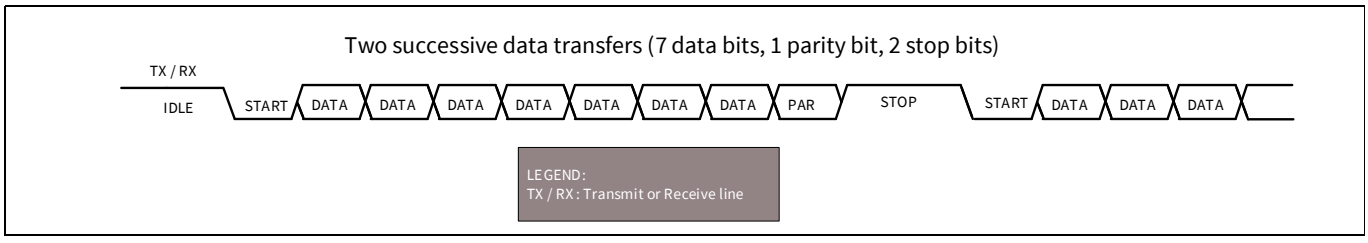



**Figure 18-9. UART, standard protocol example**

The receiver over samples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. **Figure 18-10** illustrates this.

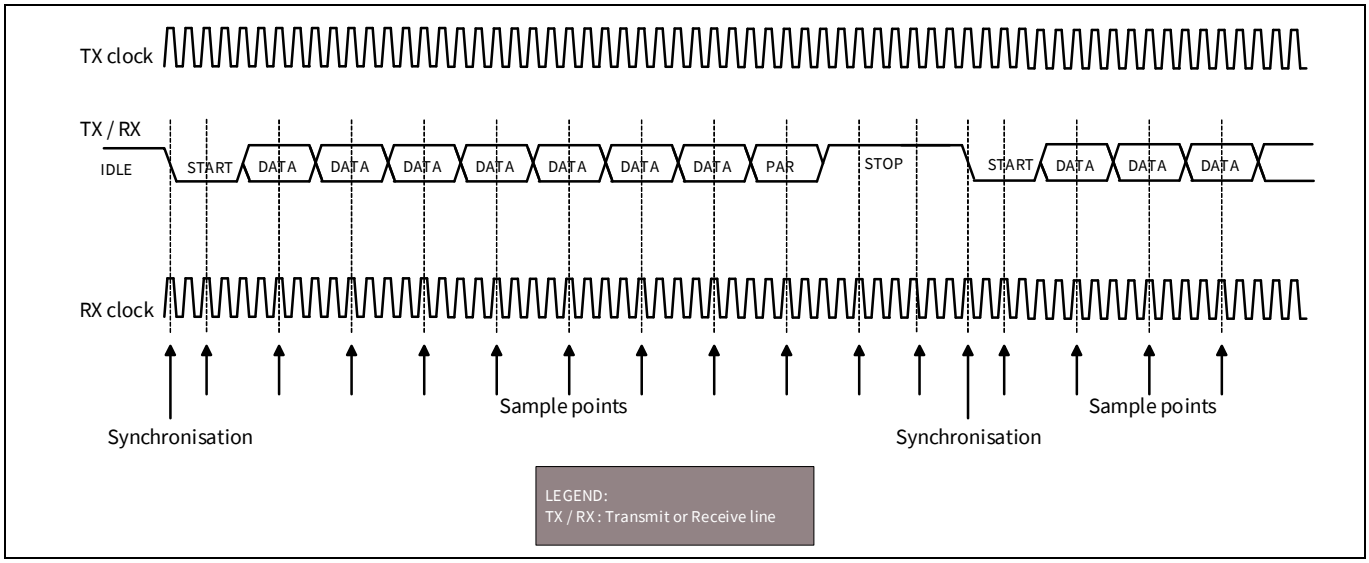



**Figure 18-10. UART, standard protocol example (Single sample)**

Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to increase accuracy. **Figure 18-11** illustrates this.

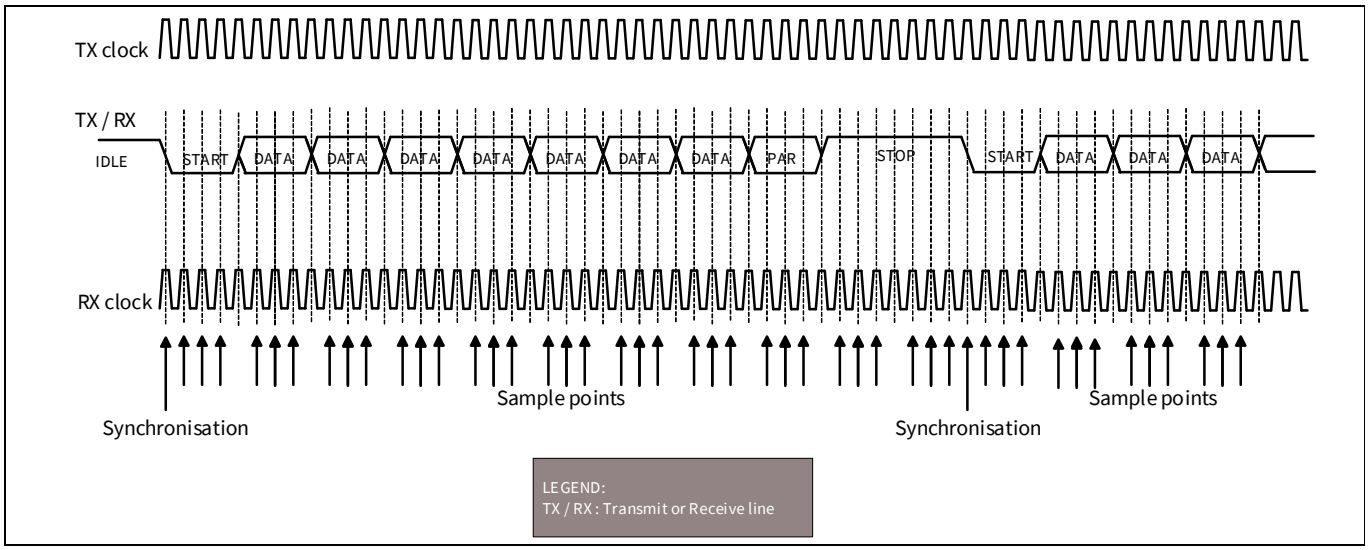



**Figure 18-11. UART, standard protocol (Multiple samples)**

### UART multi-processor mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as shown in **Figure 18-12**. This mode is also known as UART 9-bit protocol because the data field is nine bits wide. UART_MP is part of Standard UART mode.

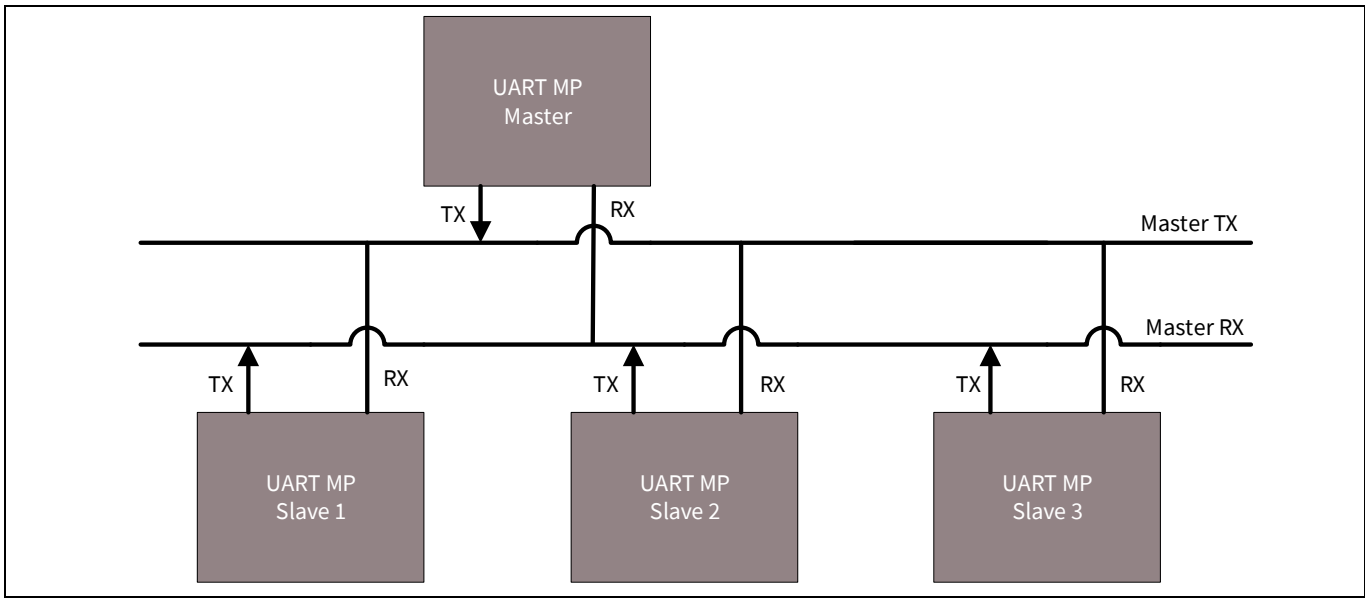



**Figure 18-12. UART MP mode bus connections**

The main properties of UART_MP mode are:

- Single master with multiple slave concept (multi-drop network)
- Each slave is identified by a unique address
- Using 9-bit data field, with the ninth bit as address/data flag (MP bit). When set high, it indicates an address byte; when set low it indicates a data byte. A data frame is illustrated in **Figure 18-13**
- Parity bit is disabled

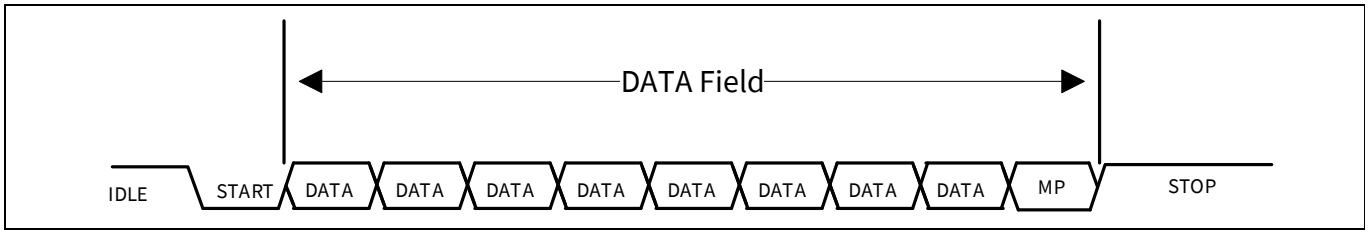



**Figure 18-13. UART MP data frame**

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

### UART LIN mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with single-master-multi-slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports both LIN master and slave functionality. **Figure 18-14** illustrates the UART_LIN and LIN transceiver.

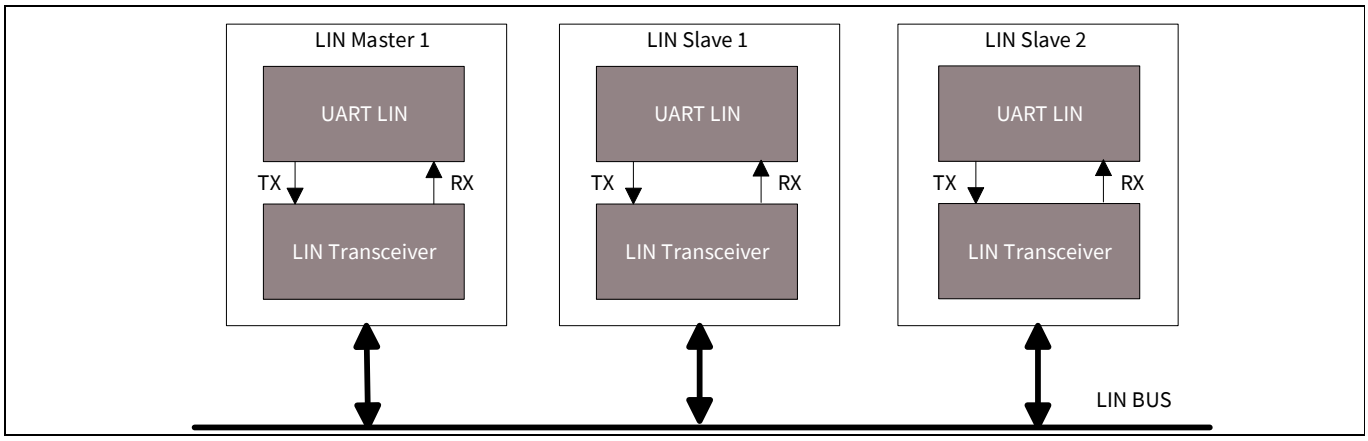



**Figure 18-14. UART_LIN and LIN transceiver**

LIN protocol defines two tasks:

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

The master node supports master task and slave task; the slave node supports only slave task, as shown in **Figure 18-15**.

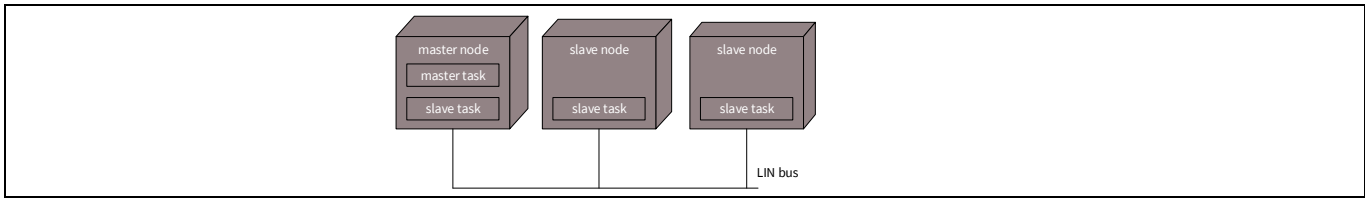



**Figure 18-15. LIN bus nodes and tasks**

LIN is based on the transmission of frames at pre-determined moments of time. A frame is divided into header and response fields.

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

The UART LIN of SCB supports slave task, receiving the header and transmitting the response. It provides baud rate detection (using sync field – 0x55) operation. Apart from the break field, a frame transmission (both header and response) consist of one or multiple byte frame transmissions, with each byte transmission consisting of a start bit, eight data bits and one or more stop bits (on both the UART TX and RX lines).

To support LIN, a dedicated (off-chip) line driver/receiver is required. Supply voltage range on the LIN bus is 7 V to 18 V. Typically, LIN line drivers will drive the LIN line with the value provided on the SCB TX line and present the value on the LIN line to the SCB RX line. By comparing TX and RX lines in the SCB, bus collisions can be detected (indicated by the UART_ARB_LOST field of the SCBx_INTR_TX register).

### Configuring the SCB as standard UART interface

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

1. Configure the SCB as UART interface by writing '10' to the MODE field (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCBx_UART_CTRL register.
3. To enable the UART MP Mode or UART LIN Mode, write '1' to the MP_MODE (bit 10) or SCB_LIN_MODE (bit 12) respectively of the SCBx_UART_RX_CTRL register.
4. Follow steps 2 to 4 described in **"Enabling and initializing UART"** on page 135.

For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

## 18.3.3.2 SmartCard (ISO 7816)

ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO 7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the **ISO 7816 specification**. Only master (reader) function is supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. UART_TX line is connected to SmartCard I/O line, by internally multiplexing between UART_TX and UART_RX control modules.

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

A SmartCard transfer has the transmitter drive the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus. Releasing results in the line being '1' (the value of a stop bit). After one bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of '0') for one bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by one bit transfer period. For this protocol to work, the stop period should be longer than one bit transfer period. Note that a data transfer with a NACK takes one bit transfer period longer, than a data transfer without a NACK. Typically, implementations use a tristate driver with a pull-up resistor, such that when the line is not transmitting data or transmitting the Stop bit, its value is '1'.

**Figure 18-16** illustrates the SmartCard protocol.

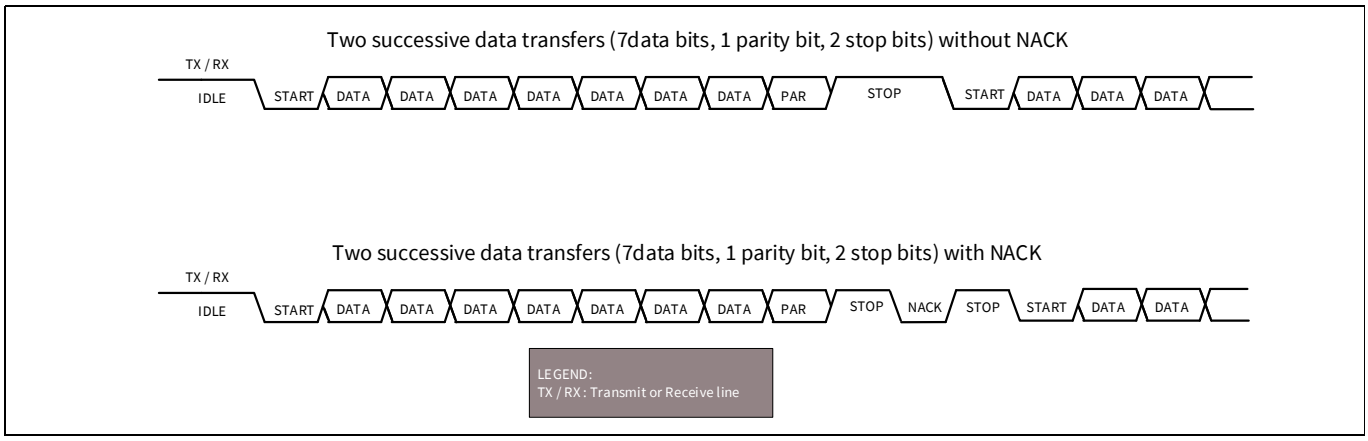



**Figure 18-16. SmartCard example**

The communication Baud rate for ISO7816 is given as:

Baud rate= f7816 × (D/F)

Where f7816 is the clock frequency, F is the clock rate conversion integer, and D is the baud rate adjustment integer.

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

#### Configuring SCB as UART SmartCard interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order. For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as a SmartCard protocol by writing '01' to the MODE (bits [25:24]) of the SCBx_UART_CTRL register.
3. Follow steps 2 to 4 described in **"Enabling and initializing UART"** on page 135.

### 18.3.3.3 IrDA

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

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

Different communication speeds under 115.2 Kbps can be achieved by configuring corresponding block clock frequency. Additional allowable rates are 2.4 Kbps, 9.6 Kbps, 19.2 Kbps, 38.4 Kbps, and 57.6 Kbps. An IrDA serial infrared interface operates at 9.6 Kbps. **Figure 18-17** shows how a UART transfer is IrDA modulated.

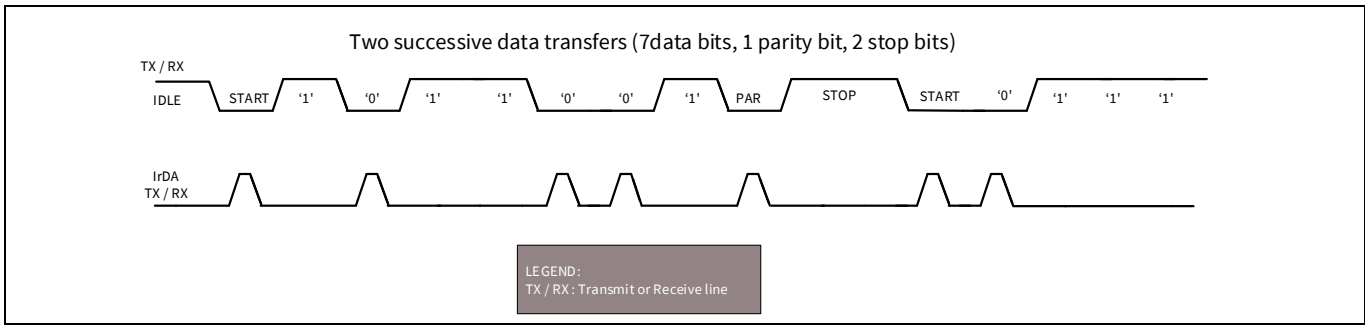



**Figure 18-17. IrDA example**

#### Configuring the SCB as UART IrDA interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order. For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the MODE (bits [25:24]) of the SCBx_UART_CTRL register.
3. Configure the SCB as described in **"Enabling and initializing UART"** on page 135.

### 18.3.4 UART registers

The UART interface is controlled using a set of 32-bit registers listed in **Table 18-9**. For more information on these registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

**Table 18-9. UART registers**

| Register name | Operation |
|---|---|
| SCBx_UART_CTRL | Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control. |
| SCBx_UART_TX_CTRL | Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK. |
| SCBx_UART_RX_CTRL | Performs same function as SCBx_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error. |
| SCBx_TX_CTRL | Used to enable the transmitter, also to specify the data frame width and to specify whether MSB or LSB is the first bit in transmission. |
| SCBx_RX_CTRL | Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver. |

### 18.3.5 UART interrupts

The UART supports both internal and external interrupt requests. The internal interrupt events are listed in this section.

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

- UART transmission is complete
- UART TX received a NACK in SmartCard mode
- UART arbitration lost (in LIN or SmartCard modes)
- Frame error in received data frame
- Parity error in received data frame
- LIN baud rate detection is complete
- LIN break detection is successful

### 18.3.6 Enabling and initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCBx_UART_CTRL register, according to **Table 18-10**. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in **Table 18-11**.
   a) Specify the data frame width.
   b) Specify whether MSB or LSB is the first bit to be transmitted or received.
   c) Enable the transmitter and receiver.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers respectively, as shown in **Table 18-12**.
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
   c) Freeze the TX and RX FIFOs.

4. Program the SCBx_CTRL register to enable the SCB block. Also select the mode of operation, as shown in **Table 18-13**.

**Table 18-10. SCBx_UART_CTRL register**

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

**Table 18-11. SCBx_TX_CTRL/SCBx_RX_CTRL registers**

| Bits | Name | Description |
|---|---|---|
| [3:0] | DATA_ WIDTH | 'DATA_WIDTH + 1' is the no. of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits. |
| 8 | MSB_FIRST | 1= MSB first<br>0= LSB first |

**Table 18-12. SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL registers**

| Bits | Name | Description |
|---|---|---|
| [2:0] | TRIGGER_LEVEL | Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case. |
| 16 | CLEAR | When '1', the transmitter or receiver FIFO and the shift registers are cleared/invalidated. |
| 17 | FREEZE | When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze will not advance the TX or RX FIFO read/write pointer. |

**Table 18-13. SCBx_CTRL register**

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

After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from SmartCard to IrDA). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example FIFO content).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCBx_CTRL register).

## 18.4 Inter integrated circuit (I²C)

EZ-PD™ PMG1-S3 MCU contains an SCB configured to operate as a I²C block. This section explains the I²C implementation in EZ-PD™ PMG1-S3 MCU. For more information on the I²C protocol specification, refer to the I²C-bus specification available on the **NXP website**.

### 18.4.1 Features

EZ-PD™ PMG1-S3 MCU supports the following I²C features:

- Master, slave, and master/slave mode
- Slow-mode (50 kbps), standard-mode (100 kbps), fast-mode (400 kbps), and fast-mode plus (1000 kbps) data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of I2C clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I²C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface
- Supports EZ mode of operation (EZI2C protocol – applicable only for I²C slave functionality)

### 18.4.2 General description

**Figure 18-18** illustrates an example of an I²C communication network.

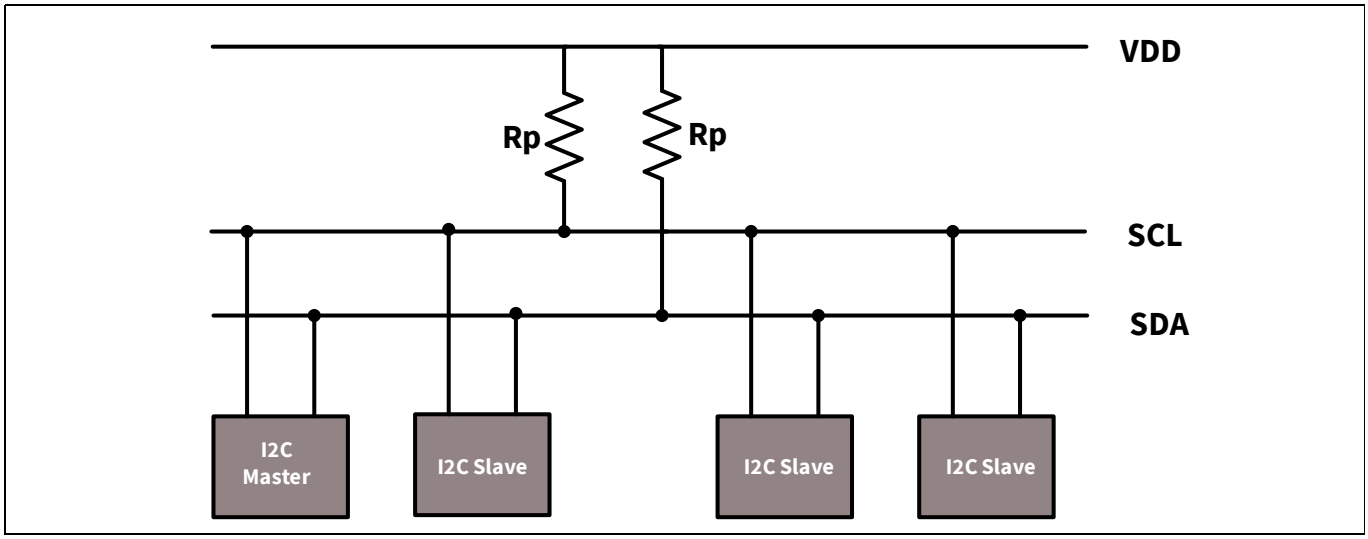



**Figure 18-18. I²C interface block diagram**

The standard I²C bus is a two wire interface with the following lines:

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

I²C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address.

### 18.4.3 Terms and definitions

Table 18-14 describes the commonly used terms in an I²C communication network.

**Table 18-14. Definition of I²C bus terminology**

| Term | Description |
|---|---|
| Transmitter | The device that sends data to the bus |
| Receiver | The device that receives data from the bus |
| Master | The device that initiates a transfer, generates clock signals, and terminates a transfer |

**Table 18-15. Illustration of I²C modes of operation**

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

**Table 18-16. I²C modes**

| Term | Description |
|---|---|
| Slave | Slave only operation (default) |
| Master | Master only operation |
| Multi-master | Supports more than one master on the bus |
| Multi-master-slave | Simultaneous slave and multi-master operation |

#### Bus stalling (Clock stretching)

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

#### Bus arbitration

The I²C protocol is a multi-master, multi-slave interface. Bus arbitration is implemented on master devices by monitoring the SDA line. Bus collisions are detected when the master observes an SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value '1' on the SDA line and master 2 is driving the value '0' on the SDA line, the actual line value will be '0' due to the implementation of the I/O signal interface. Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

### 18.4.4 I²C modes of operation

I²C is a synchronous single master, multi-master, multi-slave serial interface. Devices operate in either master mode, slave mode, or master/slave mode. In master/slave mode, the device switches from master to slave mode when it is addressed. Only a single master may be active during a data transfer. The active master is responsible for driving the clock on the SCL line.

Data transfer through the I²C bus follows a specific format. **Table 18-17** lists some common bus events that are part of an I²C data transfer. The **"Write transfer"** on page 139 and **"Read transfer"** on page 140 sections explain the format of bits on an I²C bus during data transfer.

**Table 18-17. I²C bus events terminology**

| Bus event | Description |
|---|---|
| START | A HIGH to LOW transition on the SDA line while SCL is HIGH |
| STOP | A LOW to HIGH transition on the SDA line while SCL is HIGH |
| ACK | The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse after the transmitter transmits each byte. |
| NACK | The receiver does not pull the SDA line LOW and it remains HIGH during the HIGH period of clock pulse after the transmitter transmits each byte. |
| Repeated START | START condition generated by master at the end of a transfer instead of a STOP condition |

When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see **Table 18-17**, **Figure 18-19**, and **Figure 18-20**). The master looks for a STOP signal as an indicator that it can start its data transmission.

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt.

With all of these modes, there are two types of transfer – read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in **"Master mode transfer examples"** on page 147, **"Slave mode transfer examples"** on page 149, and **"Multi-master mode transfer example"** on page 153.

#### 18.4.4.1 Write transfer

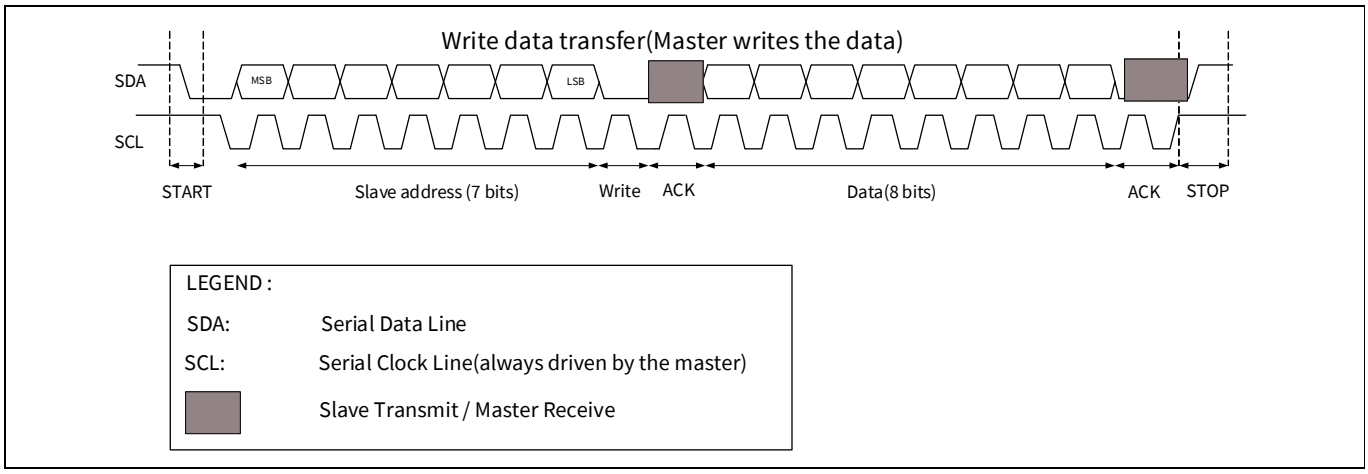



**Figure 18-19. Master write data transfer**

- A typical write transfer begins with the master generating a START condition on the I²C bus. The master then writes a 7-bit I²C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match any of the slave devices or if the addressed device does not want to acknowledge the request, it transmits a no acknowledgement (NACK). The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- The master may transmit write data to the bus if it receives an acknowledgement. The addressed slave transmits an acknowledgement to confirm the receipt of the write data. Upon receipt of this acknowledgement, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

### 18.4.4.2 Read transfer

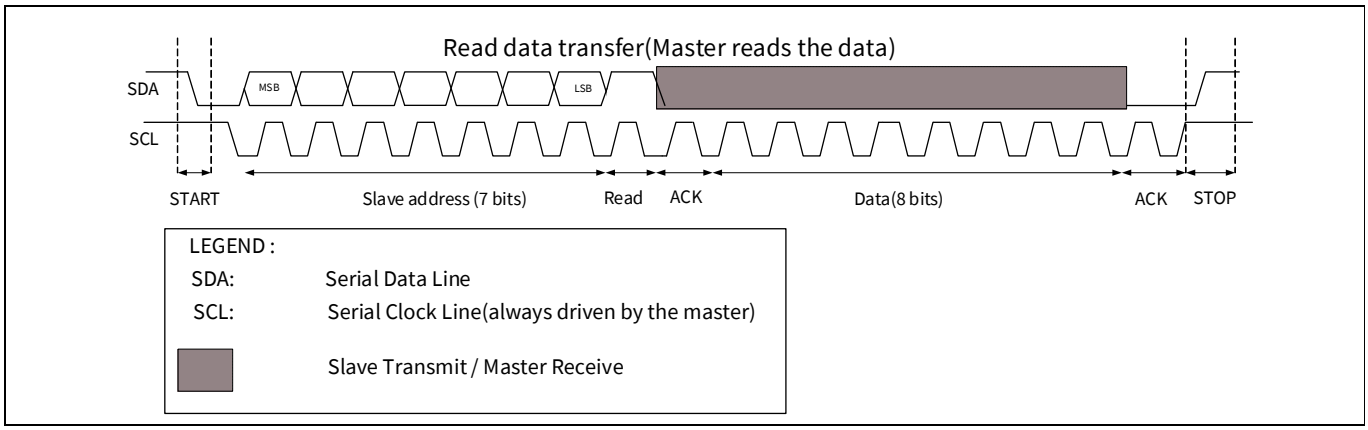



**Figure 18-20. Master read data transfer**

- A typical read transfer begins with the master generating a START condition on the I²C bus. The master then writes a 7-bit I²C slave address and a read indicator ('1') after the START condition. The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgement (NACK) is transmitted. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgement signal. The master transmits an acknowledgement to confirm the receipt of each data byte sent by the slave. Upon receipt of this acknowledgement, the addressed slave may transmit another data byte.
- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.

### 18.4.5 EZI2C protocol

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

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

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

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

### 18.4.6 Memory array write

An EZ write to a memory array index is by means of an I$^2$C write transfer. The first transmitted write data is used to send an EZ address from the master to the slave. The five lowest significant bits of the write data are used as the “new” EZ address at the slave. Additional write data elements in the write transfer are bytes that are written to the memory array. The EZ address is automatically incremented by the slave as bytes are written into the memory array. When the EZ address exceeds 32 memory entries, it wraps around to 0.

### 18.4.7 Memory array read

An EZ read from a memory array index is by means of an I$^2$C read transfer. The EZ read relies on an earlier EZ write to have set the EZ address at the slave. The first received read data is the byte from the memory array at the EZ address memory location. The EZ address is automatically incremented as bytes are read from the memory array. When the EZ address exceeds the 32 memory entries, it wraps around to 0.

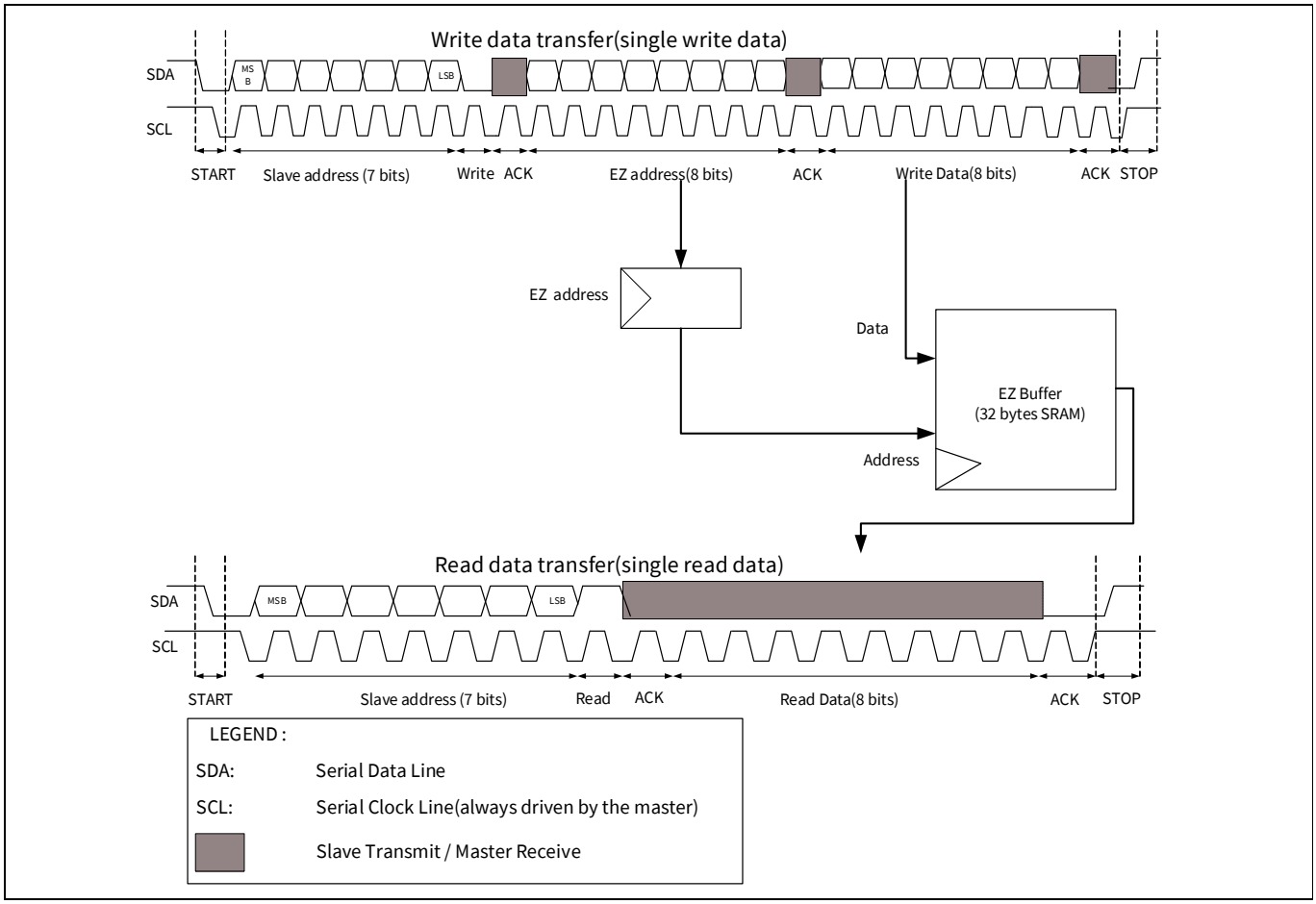



**Figure 18-21. EZI2C write and read data transfer**

See **"EZ slave mode transfer example"** on page 151 for examples.

## 18.4.8 I²C registers

The I²C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in **Table 18-18**.

**Table 18-18. I²C registers**

| Register | Function |
|---|---|
| SCBx_CTRL | Enables the I2C block and selects the type of serial interface (SPI, UART, I2C). Also used to select internally and externally clocked operation and EZ and non-EZ modes of operation. |
| SCBx_I2C_CTRL | Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status. |
| SCBx_I2C_STATUS | Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address. |
| SCBx_I2C_M_CMD | Enables the master to generate START, STOP, and ACK/NACK signals. |
| SCBx_I2C_S_CMD | Enables the slave to generate ACK/NACK signals. |
| SCBx_STATUS | Indicates whether the externally clocked logic is using the EZ memory. This bit can be used by software to determine whether it is safe to issue a software access to the EZ memory. |

**Table 18-18. I²C registers** (continued)

| Register | Function |
|---|---|
| SCBx_I2C_CFG | Configures filters, which removes glitches from the SDA and SCL lines. |
| SCBx_TX_CTRL | Enables the transmitter and specifies the data frame width; also used to specify whether MSB or LSB is the first bit in transmission. |
| SCBx_TX_FIFO_CTRL | Specifies the trigger level, clearing of the transmitter FIFO and shift registers, and FREEZE operation of the transmitter FIFO. |
| SCBx_TX_FIFO_STATUS | Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data. |
| SCBx_TX_FIFO_WR | Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation. |
| SCBx_RX_CTRL | Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver. |
| SCBx_RX_FIFO_CTRL | Performs the same function as that of the SCBx_TX_FIFO_CTRL register, but for the receiver. |
| SCBx_RX_FIFO_STATUS | Performs the same function as that of the SCBx_TX_FIFO_STATUS register, but for the receiver. |
| SCBx_RX_FIFO_RD | Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO; behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO. |
| SCBx_RX_FIFO_RD_SILENT | Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation. |
| SCBx_RX_MATCH | Stores slave device address and is also used as slave device address MASK. |
| SCBx_EZ_DATAn | Holds the data in an EZ memory location (n: 0–31). |

*Note: Detailed descriptions of the I²C register bits are available in the EZ-PD™ PMG1-S3 MCU registers TRM.*

### 18.4.9 I²C interrupts

The I²C block generates interrupts for the following conditions.

- Arbitration lost
- Slave address match
- I²C bus stop/start condition detected
- I²C bus error detected
- I²C byte/word transfer complete
- I²C TX FIFO not full
- I²C TX FIFO empty
- I²C RX FIFO empty
- I²C RX FIFO not empty
- I²C RX FIFO overrun
- I²C RX FIFO full

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

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

### 18.4.10 Enabling and initializing the I²C

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

#### Configuring for I²C standard (Non-EZ) mode

The I²C interface must be programmed in the following order.

1. Program protocol specific information using the SCBx_I2C_CTRL register according to **Table 18-19**. This includes selecting master-slave functionality.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in **Table 18-20**.
   a) Specify the data frame width.
   b) Specify whether MSB or LSB is the first bit to be transmitted/received.
   c) Enable the transmitter and receiver.
3. Program transmitter and receiver FIFO using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers, respectively, as shown in **Table 18-21**.
   a) Set the trigger level.
   b) Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCBx_CTRL register to enable the SCB block and select the I²C mode. These register bits are shown in **Table 18-22**. For a complete description of the I²C registers, see the EZ-PD™ PMG1-S3 MCU registers TRM.

**Table 18-19. SCBx_I2C_CTRL register**

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

**Table 18-20. SCBx_TX_CTRL/SCB_RX_CTRL register**

| Bits | Name | Description |
|---|---|---|
| [3:0] | DATA_ WIDTH | 'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15]. |
| 8 | MSB_FIRST | 1= MSB first |
| | | 0= LSB first |

**Table 18-21. SCBx_TX_FIFO_CTRL/ SCBx_RX_FIFO_CTRL**

| Bits | Name | Description |
|---|---|---|
| [2:0] | TRIGGER_LEVEL | Trigger level. When the transmitter FIFO has less entries or the receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case. |
| 16 | CLEAR | When '1', the transmitter or receiver FIFO and the shift registers are cleared. |
| 17 | FREEZE | When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer. |

**Table 18-22. SCBx_CTRL registers**

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

**Configuring for EZI2C mode**

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

1. Select the EZI2C mode by writing '1' to the EZ_MODE bit (bit 10) of the SCBx_CTRL register.
2. Follow steps 2 to 4 from **"Configuring for I2C standard (Non-EZ) mode"** on page 144.
3. Set the S_READY_ADDR_ACK (bit 12) and S_READY_DATA_ACK (bit 13) bits of the SCBx_I2C_CTRL register.

## 18.4.11 Internal and external clock operation in I²C

The I2C supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the EZ-PD™ PMG1-S3 MCU system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the **"Clocking system"** on page 85.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality. TX and RX FIFOs do not support externally clocked operation; therefore, it is not used for non-EZ functionality.

Internally and externally clocked operations are determined by two register fields of the SCBx_CTRL register:

- **EC_AM_MODE (Externally clocked address matching mode)**: Indicates whether I²C address match is internally ('0') or externally ('1') clocked.
- **EC_OP_MODE (Externally clocked operation mode)**: Indicates whether the remaining protocol operation (besides I²C address match) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does not support non-EZ functionality.

These two register fields determine the functional behavior of I²C. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep power modes. Improper setting may result in faulty behavior in certain power modes. **Table 18-23** and **Table 18-23** describe the settings for I²C in EZ and non-EZ mode.

### 18.4.11.1 I²C non-EZ operation mode

Externally clocked operation is not supported for non-EZ functionality because there is no FIFO support for this mode. So, the EC_OP_MODE should always be set to '0' for non-EZ mode. However, EC_AM_MODE can be set to '0' or '1'. **Table 18-9** gives an overview of the possibilities. The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

**EC_AM_MODE is '0' and EC_OP_MODE is '0'.** This setting only works in Active and Sleep power modes. All the I²C's functionality is provided in the internally clocked domain.

**EC_AM_MODE is '1' and EC_OP_MODE is '0'.** This setting works in Active and Deep-Sleep power modes. I²C address match is performed by the externally clocked logic in both these modes. When the externally clocked logic matches the address, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wakeup the CPU.

- In Active mode, the CPU is active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). The externally clocked logic takes care of the address match and the internally locked logic takes care of the rest of the I²C transfer.
- In Sleep mode, wakeup interrupt cause can be either enabled or disabled based on the application. The remaining operations are similar to the Active mode.
- In Deep-Sleep mode, the CPU is shut down and will wake up on I²C activity if the wakeup interrupt cause is enabled. CPU wakeup takes time and the ongoing I2C transfer is either negatively acknowledged (NACK) or the clock is stretched. In the case of a NACK, the internally clocked logic takes care of the first I²C transfer after it wakes up. For clock stretching, the internally clocked logic takes care of the ongoing/stretched transfer when it wakes up. The register bit S_NOT_READY_ADDR_NACK (bit 14) of the SCBx_I2C_CTRL register determines whether the externally clocked logic performs a negative acknowledge ('1') or clock stretch ('0').

### 18.4.11.2 EZ operation mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. **Table 18-23** gives an overview of the possibilities. The cells in grey indicate a possible, yet not recommended setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

**Table 18-23. I²C EZ mode**

| I2C, EZ mode | | | | |
|---|---|---|---|---|
| **System power mode** | **EC_OP_MODE= 0** | | **EC_OP_MODE = 1** | |
| | **EC_AM_MODE = 0** | **EC_AM_MODE = 1** | **EC_AM_MODE = 1** | **EC_AM_MODE=0** |
| Active and Sleep | Address match using internal clock<br>Operation using internal clock | Address match using external clock<br>Operation using internal clock | Address match using external clock<br>Operation using external clock | Invalid configuration |
| Deep-Sleep | Not supported | Address match using external clock<br>Operation using internal clock | Address match using external clock<br>Operation using external clock | |

- **EC_AM_MODE is '0' and EC_OP_MODE is '0'**. This setting only works in Active and Sleep power modes.
- **EC_AM_MODE is '1' and EC_OP_MODE is '0'**. This setting works similar to the I²C non-EZ mode.
- **EC_AM_MODE is '1' and EC_OP_MODE is '1'**. This setting works in Active and Deep-Sleep power modes.

The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The FIFO_BLOCK field (bit 17) of the SCBx_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

## 18.4.12 Wake up from Sleep

The system wakes up from Sleep or Deep-Sleep power modes when an I²C address match occurs. The I²C block performs either of two actions after address match: Address ACK or Address NACK.

**Address ACK** – The I²C slave executes clock stretching and waits until the device wakes up and ACKs the address.

**Address NACK** – The I²C slave NACKs the address immediately. The master must poll the slave again after the device wakeup time is passed. This option is only valid in the slave or multi-master-slave modes.

*Note: The interrupt bit WAKEUP (bit 0) of the INTR_I2C_EC register must be enabled for the I²C to wake up the device on slave address match while switching to the Sleep mode.*

### 18.4.13 Master mode transfer examples

Master mode transmits or receives data.

#### 18.4.13.1 Master transmit

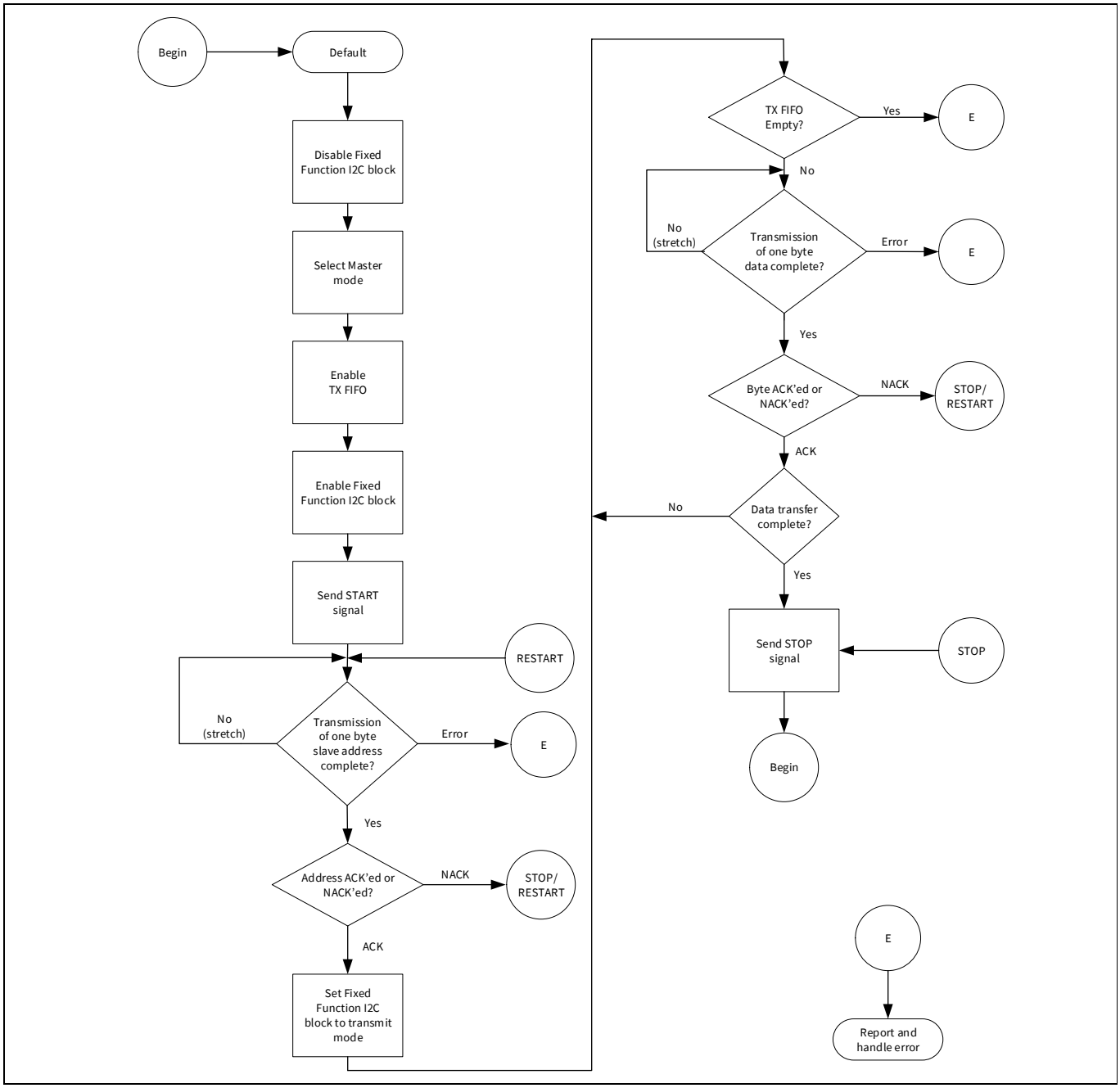



**Figure 18-22. Single master mode write operation flow chart**

### 18.4.13.2 Master receive

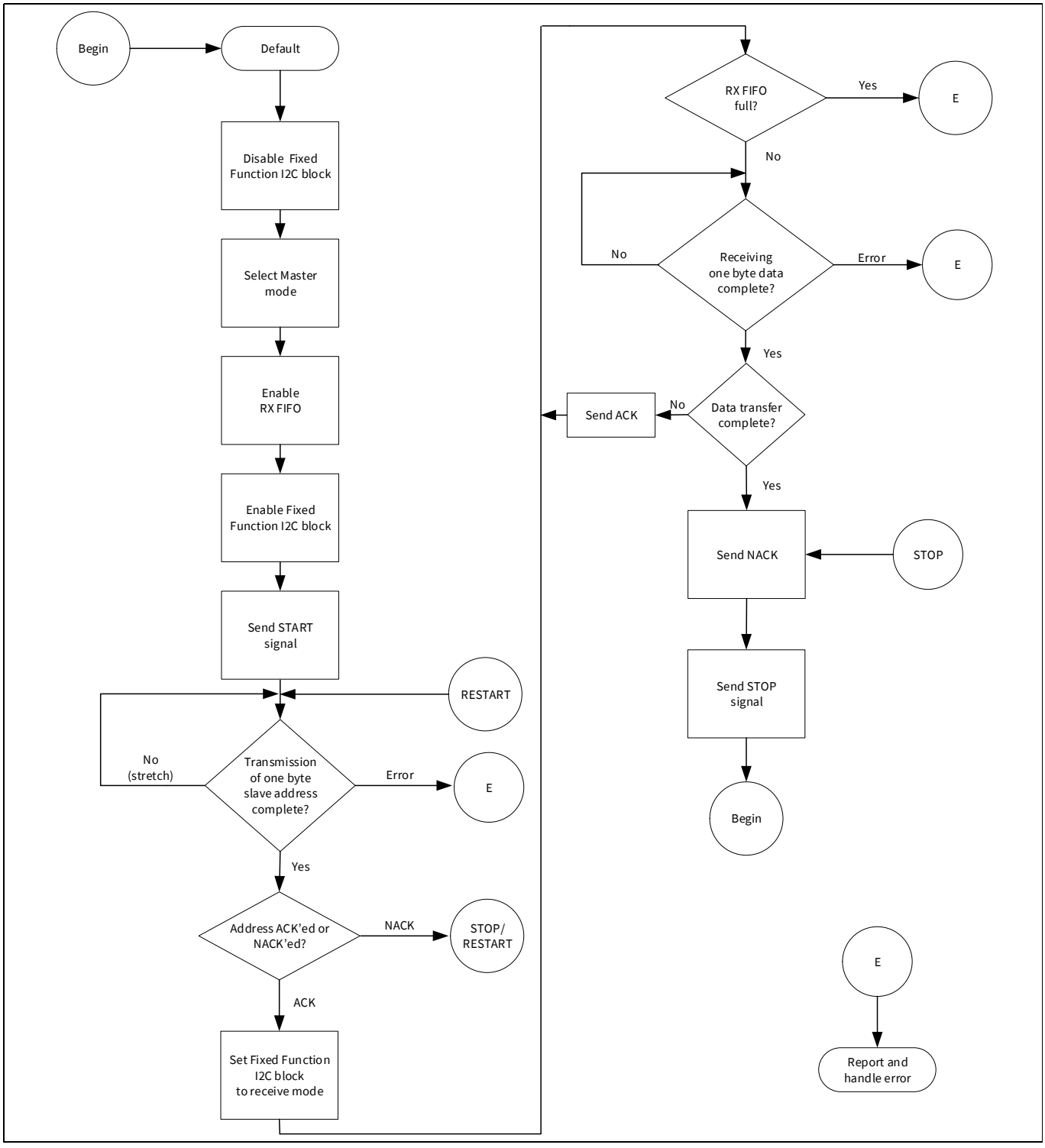



**Figure 18-23. Single master mode read operation flow chart**

### 18.4.14 Slave mode transfer examples

Slave mode transmits or receives data.

#### 18.4.14.1 Slave transmit

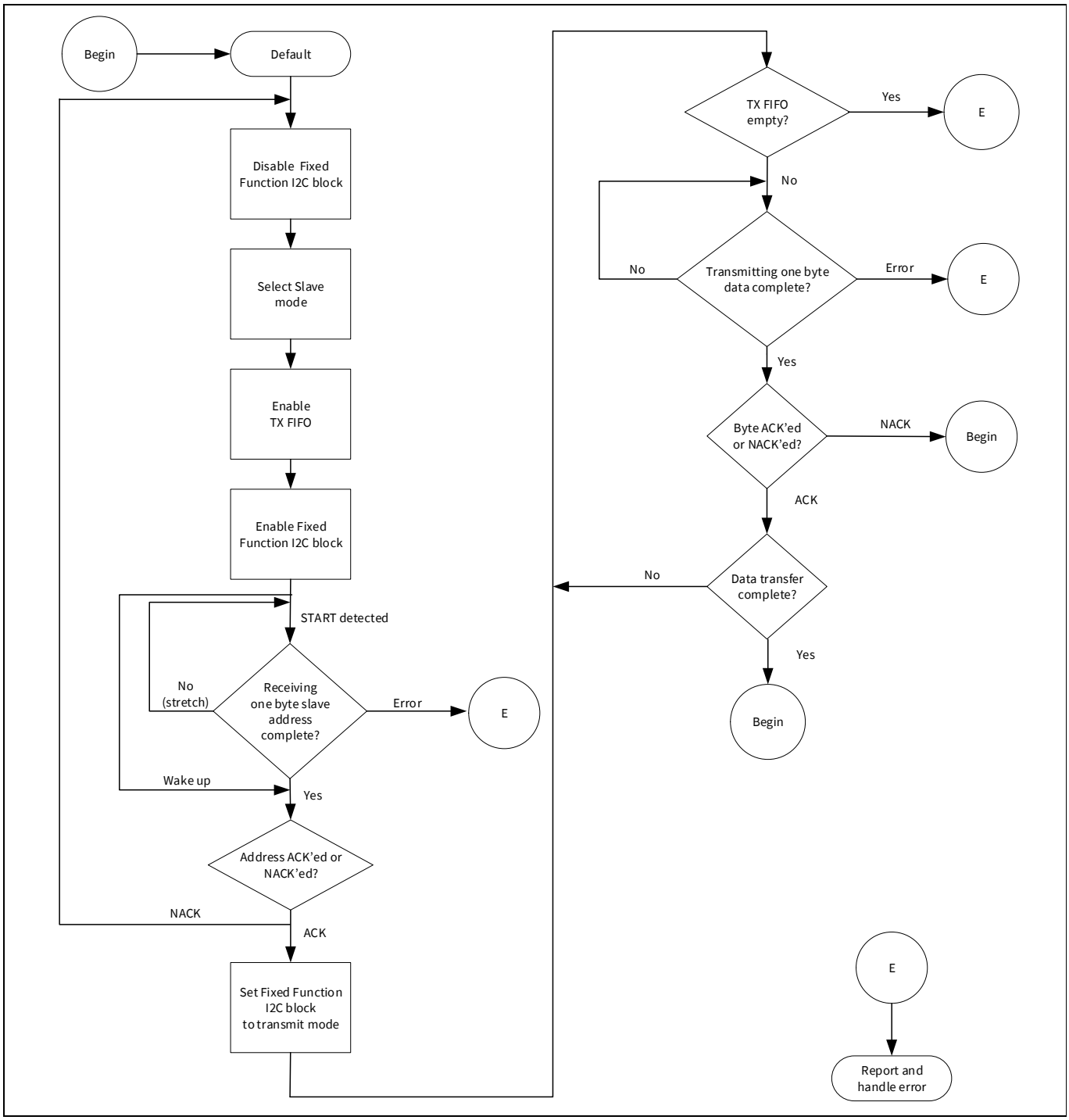



**Figure 18-24. Slave mode write operation flow chart**

### 18.4.14.2 Slave receive

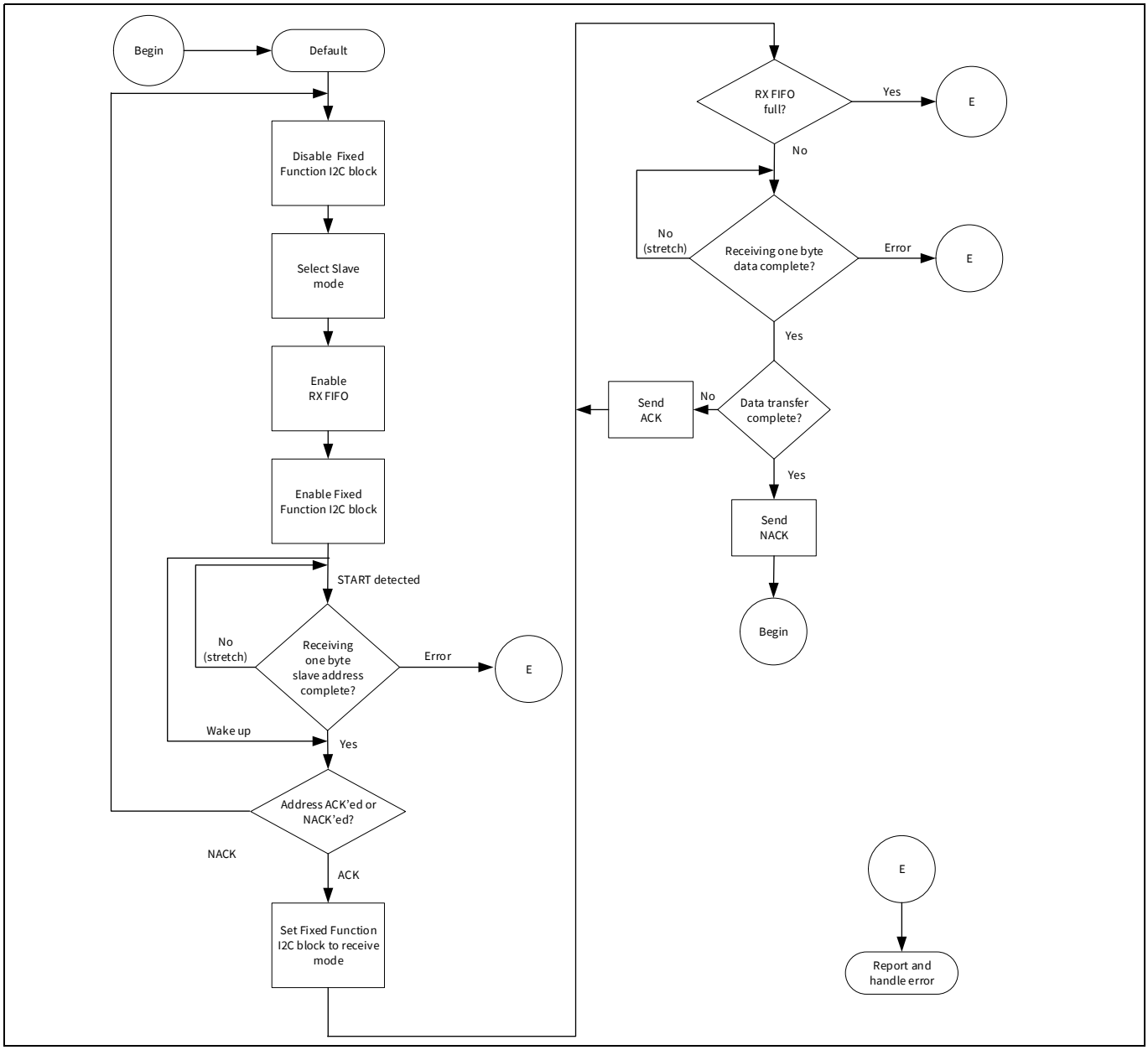



**Figure 18-25. Slave mode read operation flow chart**

### 18.4.15 EZ slave mode transfer example

The EZ Slave mode transmits or receives data.

#### 18.4.15.1 EZ slave transmit

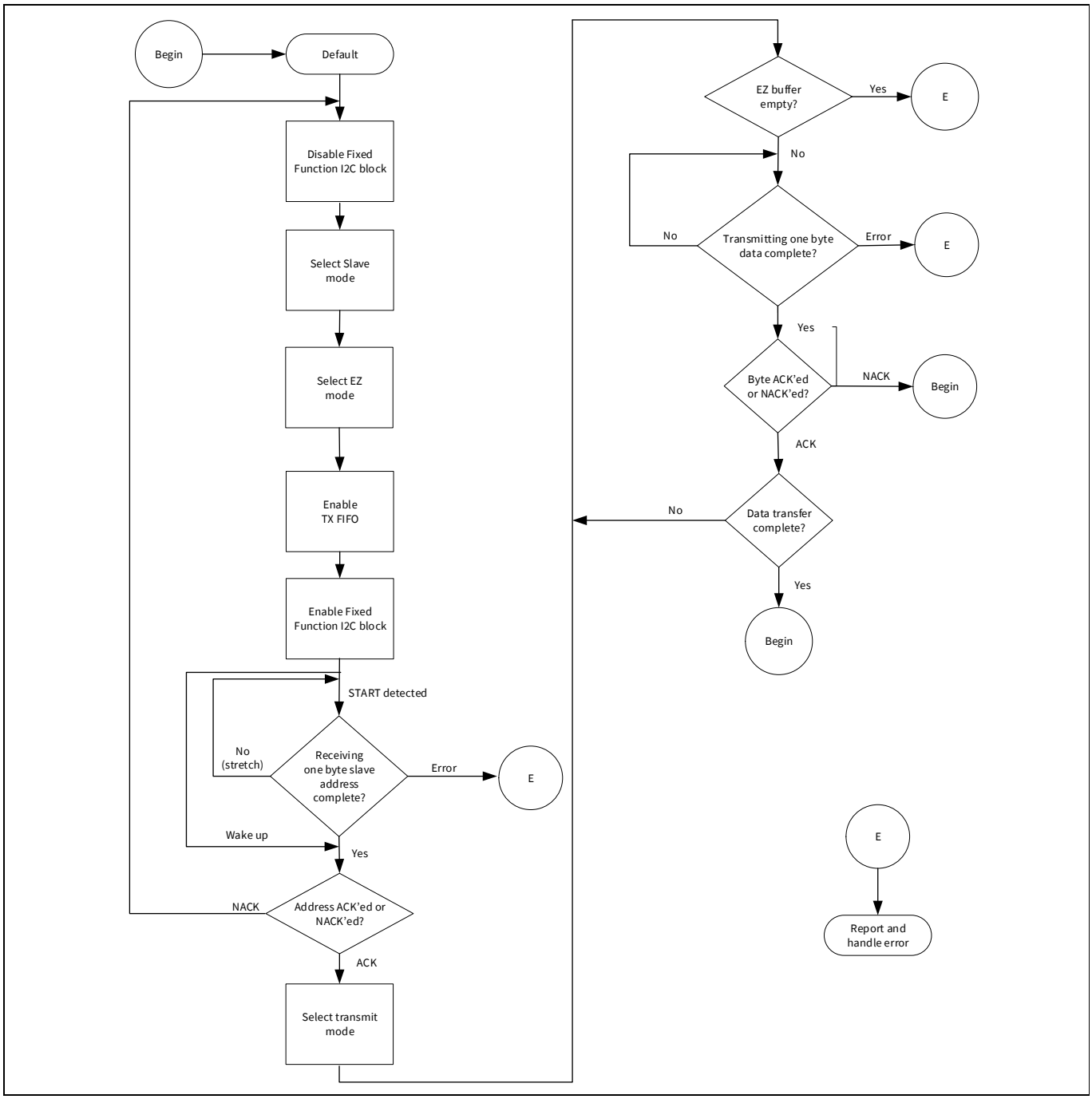



**Figure 18-26. EZI2C slave mode write operation flow chart**

### 18.4.15.2 EZ slave receive

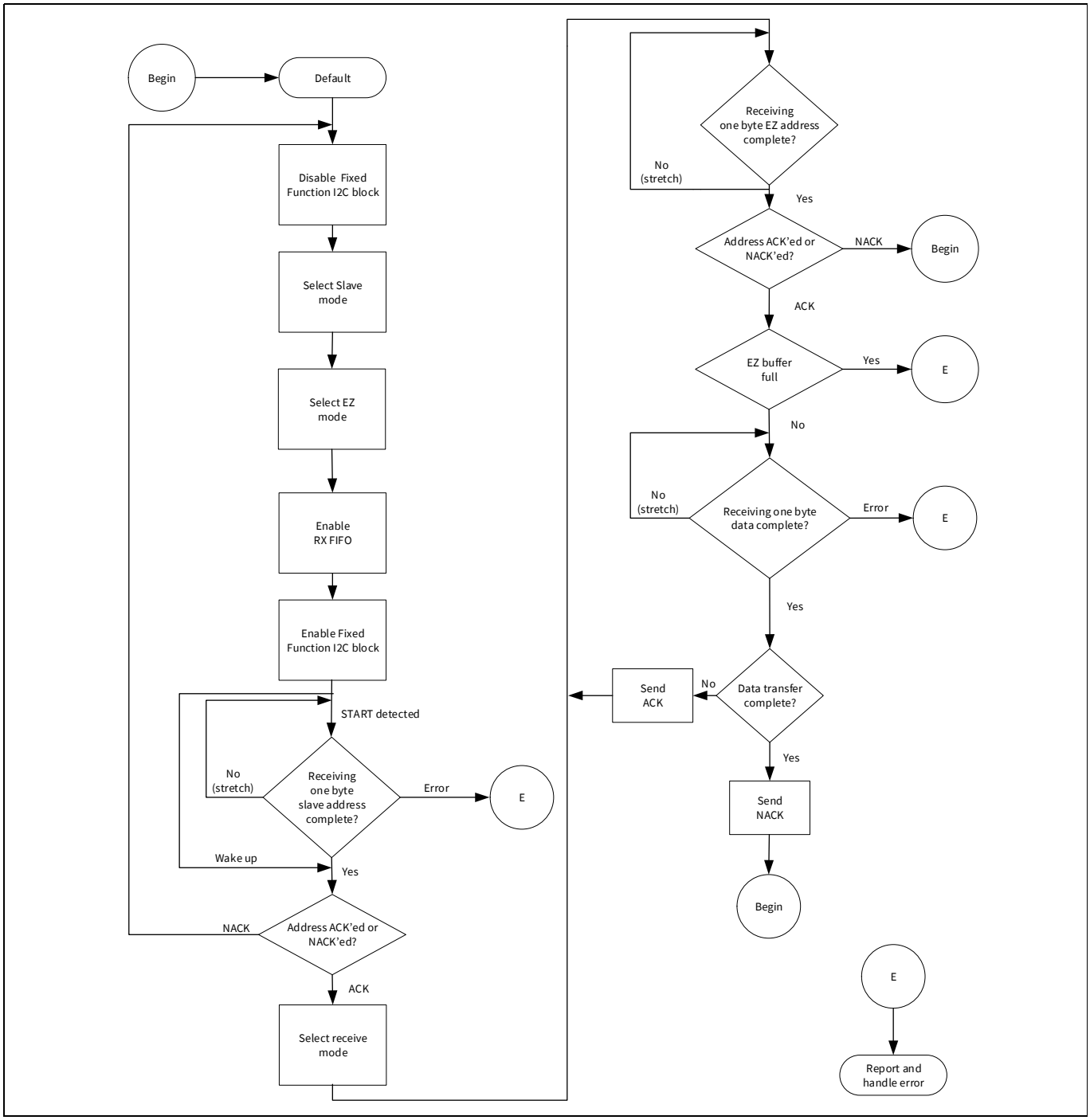



**Figure 18-27. EZI2C slave mode read operation flow chart**

### 18.4.16 Multi-master mode transfer example

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

#### 18.4.16.1 Multi-master – Slave not enabled

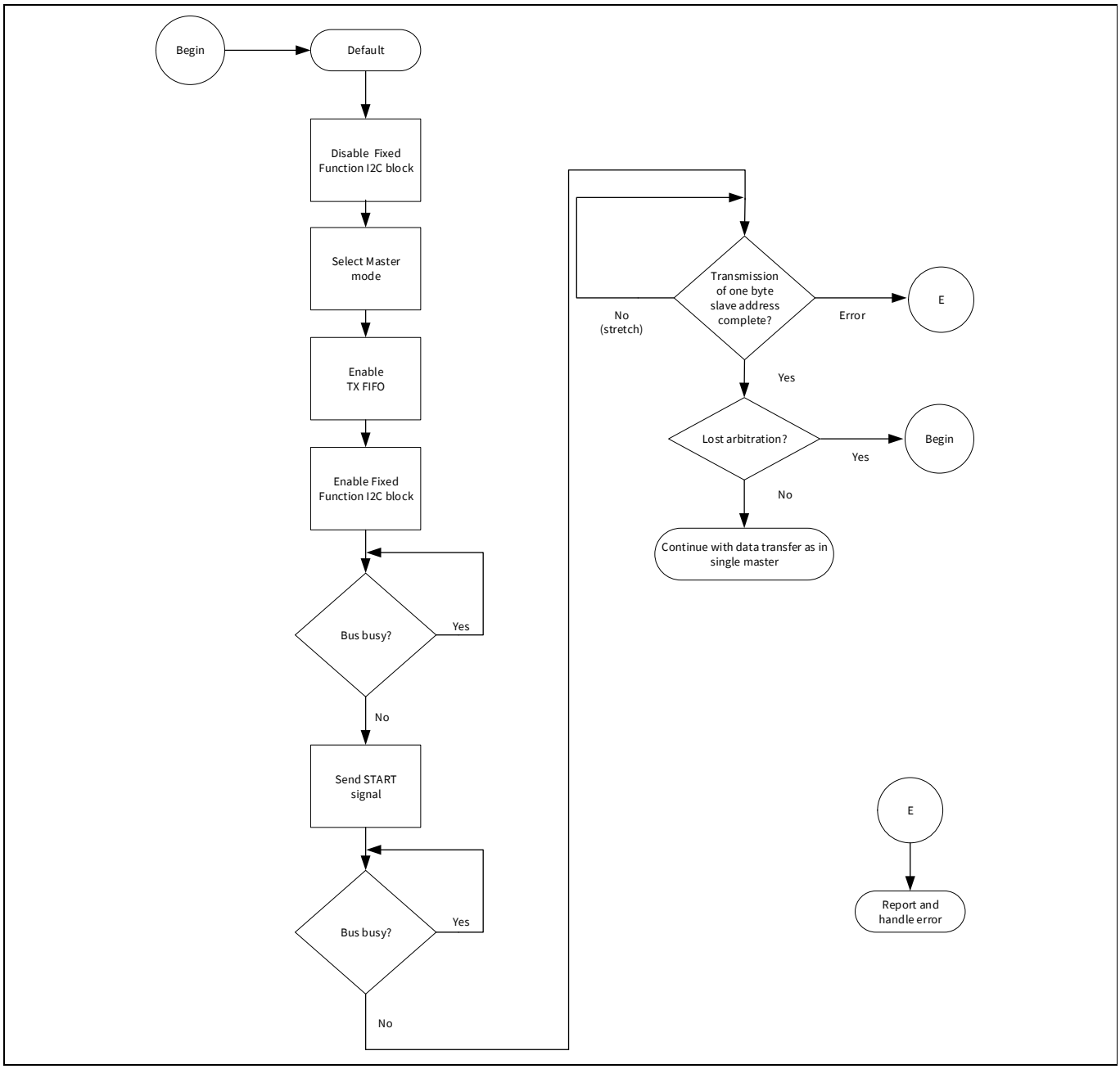



**Figure 18-28. Multi-master, slave not enabled flow chart**

### 18.4.16.2 Multi-master – Slave enabled

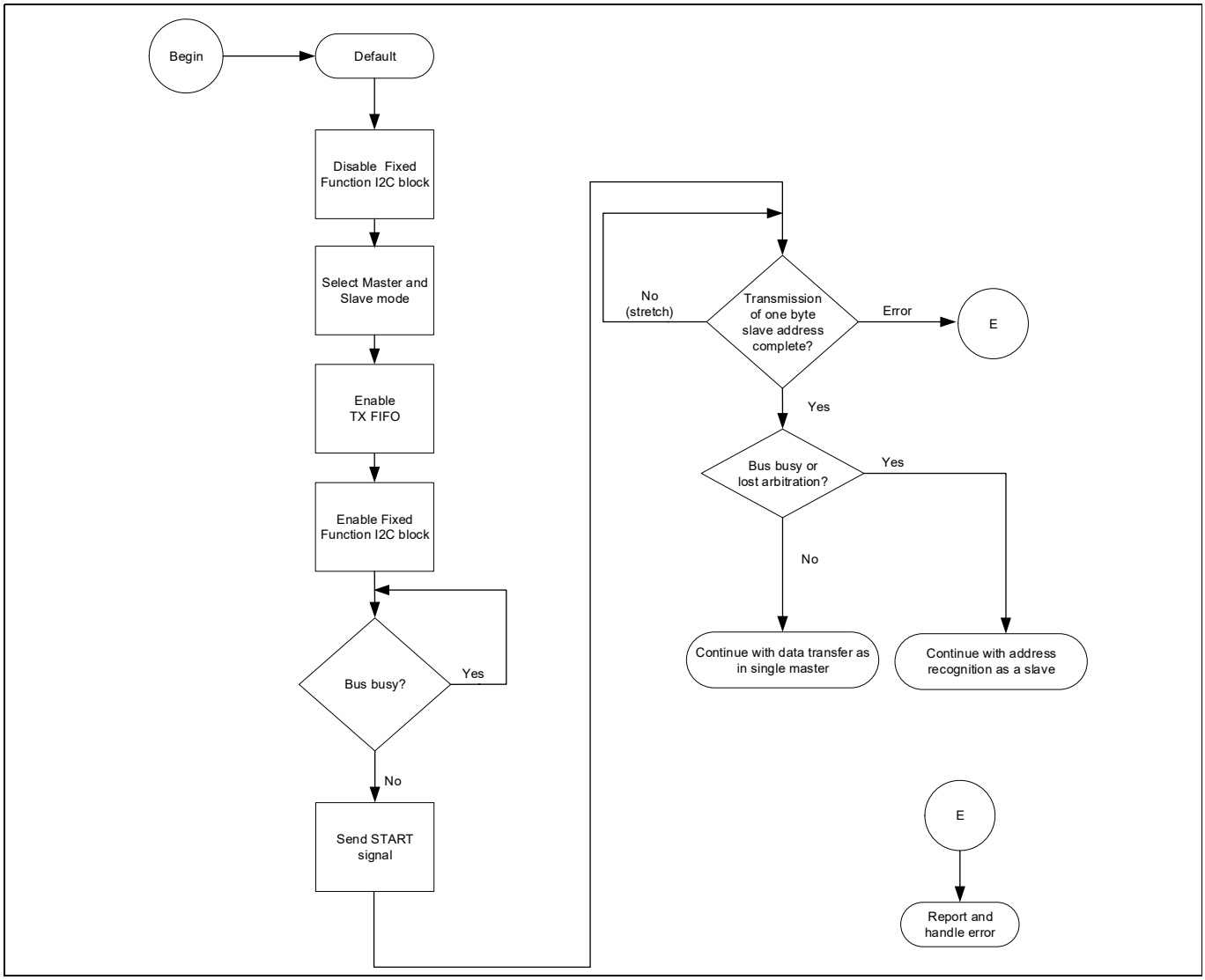



**Figure 18-29. Multi-master, slave enabled flow chart**

# 19 Timer, Counter, and PWM

The Timer, Counter, and Pulse Width Modulator (TCPWM) block in EZ-PD™ PMG1-S3 MCU implements the 16-bit timer, counter, pulse width modulator (PWM), and quadrature decoder functionality. The block can be used to measure the period and pulse width of an input signal (timer), find the number of times a particular event occurs (counter), generate PWM signals, or decode quadrature signals. This chapter explains the features, implementation, and operational modes of the TCPWM block.

## 19.1 Features

- Eight TCPWM modules in EZ-PD™ PMG1-S3 MCU 97-BGA package and seven TCPWM modules in PMG1-S3 48-QFN package.
- Supports the following operational modes:
  - Timer
  - Counter
  - Capture
  - Quadrature decoding
  - Pulse width modulation
  - Pseudo-random PWM
  - PWM with dead time
- Multiple counting modes – up, down, and up/down
- Clock pre-scaling (division by 1, 2, 4..64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count (TC) – The final value in the counter register is reached
  - Capture/Compare (CC) – The count is captured to the capture/compare register or the counter value equals the compare value
- Synchronized counters – The counters can reload, start, stop, and count at the same time
- Complementary line output for PWMs

## 19.2 Block diagram

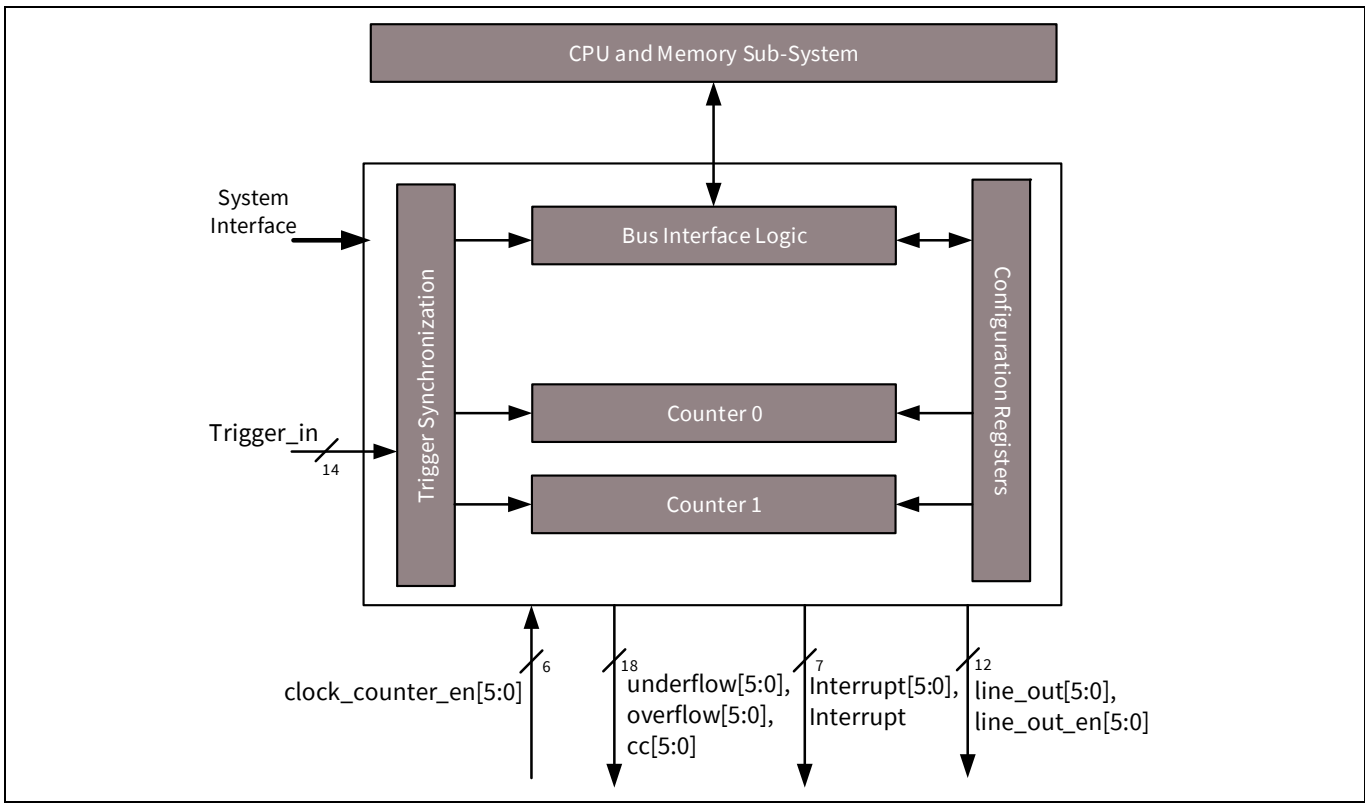



**Figure 19-1. TCPWM block diagram**

The block has these interfaces:

- Bus interface: Connects the block to the CPU subsystem.
- Interrupts: Provides interrupt request signals from each counter, based on terminal count (TC) or CC conditions, and a combined interrupt signal generated by the logical OR of all six interrupt request signals.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem.

This TCPWM block can be configured by writing to the TCPWM registers. See **"Registers"** on page 177 for more information on all registers required for this block.

The Trigger_in interfaces of the TCPWM blocks are connected to internal trigger outputs from the USBPD block. TCPWM on EZ-PD™ PMG1-S3 MCU does not support any external triggers for the counters.

### 19.2.1 Enabling and disabling counter in TCPWM block

The counter can be enabled by setting the COUNTER_ENABLED field (bit 0) of the control register, TCPWM_CTRL.

*Note: The counter must be configured before enabling it. If the counter is enabled after being configured, registers are updated with the new configuration values. Disabling the counter retains the values in the registers until it is enabled again (or reconfigured).*

### 19.2.2 Clocking

The TCPWM receives the HFCLK through the system interface to synchronize all events in the block. The counter enable signal (counter_en), which is generated when the counter is enabled, gates the HFCLK to provide a counter-specific clock (counter_clock). Output triggers (explained later in this chapter) are also synchronized with the HFCLK.

**Clock pre-scaling:** counter_clock can be pre-scaled, with divider values of 1, 2, 4… 64, 128. This is done by modifying the GENERIC field of the counter control (TCPWM_CNTx_CTRL) register, as shown in **Table 19-1**.

**Table 19-1. Bit-field setting to pre-scale counter clock**

| GENERIC[10:8] | Description |
|---|---|
| 0 | Divide by 1 |
| 1 | Divide by 2 |
| 2 | Divide by 4 |
| 3 | Divide by 8 |
| 4 | Divide by 16 |
| 5 | Divide by 32 |
| 6 | Divide by 64 |
| 7 | Divide by 128 |

*Note: Clock pre-scaling cannot be done in quadrature mode and pulse width modulation mode with dead time (PWM-DT).*

### 19.2.3 Events based on trigger inputs

These are the events triggered by hardware or software.

- Reload
- Start
- Stop
- Count
- Capture/switch

Hardware triggers can be level signal, rising edge, falling edge, or both edges.

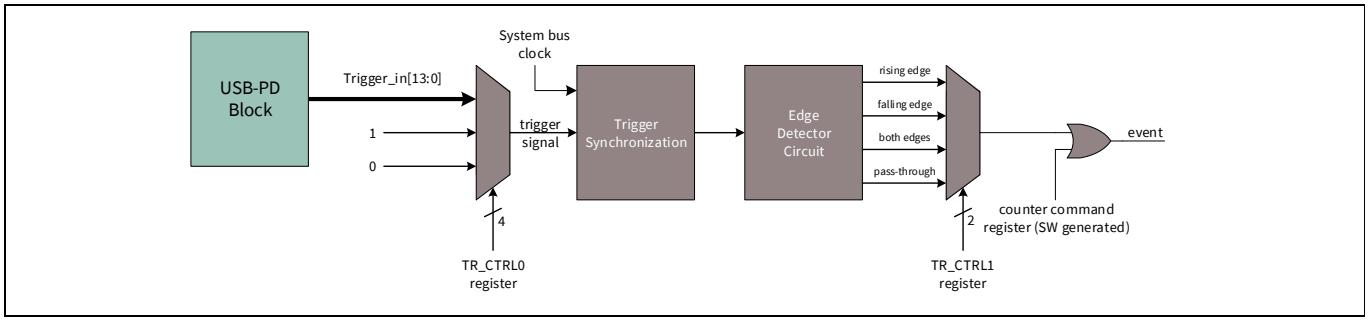



**Figure 19-2. TCPWM trigger selection and event detection**

**Figure 19-2** shows the trigger selection and event detection in the TCPWM block. The trigger control register 0 (TCPWM_CNTx_TR_CTRL0) selects one of the 16 trigger inputs as the event signal. Additionally, constant '0' and '1' signals are available to be used as the event signal.

As shown in the figure, the Trigger_in inputs on EZ-PD™ PMG1-S3 MCU have been connected internally to trigger outputs generated by the USBPD block. External (GPIO) trigger inputs are not supported.

Any edge (rising, falling, or both) or level (high or low) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNTx_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in **Figure 19-2**.

The events derived from these triggers can have different definitions in different modes of the TCPWM block.

- **Reload:** A reload event initializes and starts the counter.
  - In up counting mode, the count register (TCPWM_CNTx_COUNTER) is initialized with '0'.
  - In down counting mode, the counter is initialized with the period value stored in the TCPWM_CNTx_PERIOD register.
  - In up/down counting mode, the count register is initialized with '0'.
  - In quadrature mode, the reload event acts as a quadrature index event. An index/reload event indicates a completed rotation and can be used to synchronize quadrature decoding.
- **Start:** A start event is used to start counting; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
  - In quadrature mode, the start event acts as quadrature phase input phiB, which is explained in detail in **"Quadrature decoder mode"** on page 166.
- **Count:** A count event causes the counter to increment or decrement, depending on its configuration.
  - In quadrature mode, the count event acts as quadrature phase input phiA.
- **Stop:** A stop event stops the counter from incrementing or decrementing. A start event will start the counting again.
  - In the PWM modes, the stop event acts as a kill event. A kill event disables all the PWM output lines.
- **Capture:** A capture event copies the counter register value to the capture register and capture register value to the buffer capture register. In the PWM modes, the capture event acts as a switch event. It switches the values of the capture/compare and period registers with their buffer counterparts. This feature can be used to modulate the pulse width and frequency.

**Notes**

- All trigger inputs are synchronized to the HFCLK.
- When more than one event occurs in the same counter clock period, one or more events may be missed. This can happen for high-frequency events (frequencies close to the counter frequency) and a timer configuration in which a pre-scaled (divided) counter clock is used.

## 19.2.4 Output signals

The TCPWM block generates several output signals, as shown in **Figure 19-3**.

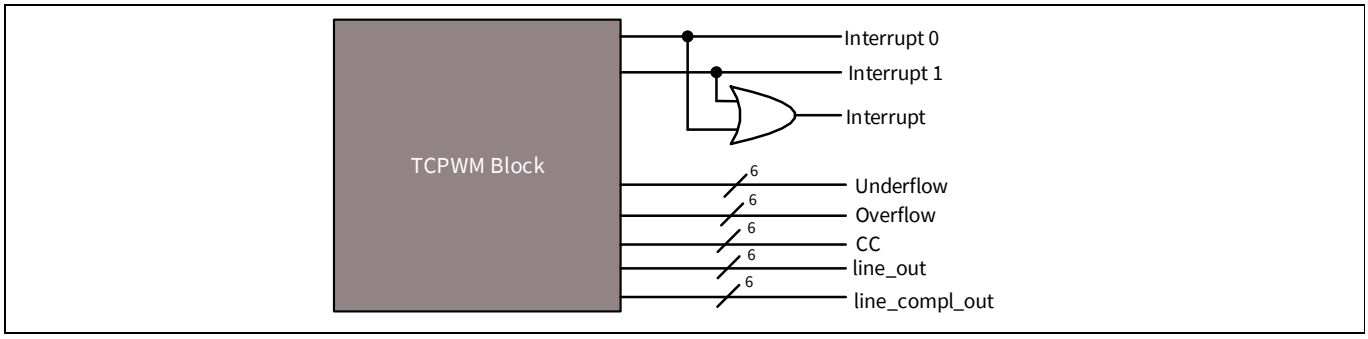



**Figure 19-3. TCPWM output signals**

### 19.2.4.1 Signals upon trigger conditions

- Counter generates an internal overflow (OV) condition when counting up and the count register reaches the period value.
- Counter generates an internal underflow (UN) condition when counting down and the count register reaches zero.
- The capture/compare (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:
  - The counter value equals the compare value.
  - A capture event occurs – When a capture event occurs, the TCPWM_CNTx_COUNTER register value is copied to the capture register and the capture register value is copied to the buffer capture register.

*Note: These signals, when they occur, remain at logic high for one cycle of the HFCLK. For reliable operation, the condition that causes this trigger should occur in a frequency less than a quarter of the HFCLK. For example, if the HFCLK is running at 24 MHz, the condition causing the trigger should occur at a frequency less than 6 MHz.*

On EZ-PD™ PMG1-S3 MCU, the Underflow output signals have not been bonded to any of the GPIOs and hence are not available as external outputs.

### 19.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific. All six interrupt output signals from the four TCPWMs are also OR'ed together to produce a single interrupt output signal.

Four registers are used for interrupt handling in this block, as shown in **Table 19-2**.

**Table 19-2. Interrupt register**

| Interrupt registers | Bits | Name | Description |
|---|---|---|---|
| TCPWM_CNTx_INTR (Interrupt request register) | 0 | TC | This bit is set to '1', when a terminal count is detected. Write '1' to clear this bit. |
| | 1 | CC_MATCH | This bit is set to '1' when the counter value matches capture/compare register value. Write with '1' to clear this bit. |
| TCPWM_CNTx_INTR_SET (Interrupt set request register) | 0 | TC | Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status. |
| | 1 | CC_MATCH | Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status. |
| TCPWM_CNTx_INTR_MASK (Interrupt mask register) | 0 | TC | Mask bit for the corresponding TC bit in the interrupt request register. |
| | 1 | CC_MATCH | Mask bit for the corresponding CC_MATCH bit in the interrupt request register. |
| TCPWM_CNTx_INTR_MASKED (Interrupt masked request register) | 0 | TC | Logical AND of the corresponding TC request and mask bits. |
| | 1 | CC_MATCH | Logical AND of the corresponding CC_MATCH request and mask bits. |

### 19.2.4.3 Outputs

The TCPWM has two outputs, line_out and line_compl_out (complementary of line_out). Note that the OV, UN, and CC conditions can be used to drive line_out and line_compl_out if needed, by configuring the TCPWM_CNTx_TR_CTRL2 register (see **Table 19-3**).

**Table 19-3. Configuring output line for OV, UN, and CC conditions**

| Field | Bit | Value | Event | Description |
|---|---|---|---|---|
| CC_MATCH_MODE Default Value = 3 | 1:0 | 0 | Set line_out to '1 | Configures output line on a compare match (CC) event |
| | | 1 | Clear line_out to '0 | |
| | | 2 | Invert line_out | |
| | | 3 | No change | |
| OVERFLOW_MODE Default Value = 3 | 3:2 | 0 | Set line_out to '1 | Configures output line on a overflow (OV) event |
| | | 1 | Clear line_out to '0 | |
| | | 2 | Invert line_out | |
| | | 3 | No change | |
| UNDERFLOW_MODE Default Value = 3 | 5:4 | 0 | Set line_out to '1 | Configures output line on a underflow (UN) event |
| | | 1 | Clear line_out to '0 | |
| | | 2 | Invert line_out | |
| | | 3 | No change | |

## 19.2.5 Power modes

The TCPWM block works in Active and Sleep modes. The TCPWM block is powered from VCCD. The configuration registers and other logic are powered in Deep-Sleep mode to keep the states of configuration registers. See **Table 19-4**.

**Table 19-4. Power modes in TCPWM block**

| Power mode | Block status |
|---|---|
| Active | This block is fully operational in this mode with clock running and power switched ON. |
| Sleep | All counter clocks are on, but bus interface cannot be accessed. |
| Deep-Sleep | In this mode, the power to this block is still on but no bus clock is provided; hence, the logic is not functional. All the configuration registers will keep their state. |

## 19.3 Modes of operation

The counter block can function in six operational modes, as shown in **Table 19-5**. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

**Table 19-5. Operational mode configuration**

| Mode | MODE field [26:24] | Description |
|---|---|---|
| Timer | 000 | Implements a timer or counter. The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected. |
| Capture | 010 | Implements a timer or counter with capture input. The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected. When a capture event occurs, the counter value copies into the capture register. |
| Quadrature Decoder | 011 | Implements a quadrature decoder, where the counter is decremented or incremented, based on two phase inputs according to the selected (X1, X2 or X4) encoding scheme. |
| PWM | 100 | Implements edge/center-aligned PWMs with an 8-bit clock prescaler and buffered compare/period registers. |
| PWM-DT | 101 | Implements edge/center-aligned PWMs with configurable 8-bit dead time (on both outputs) and buffered compare/period registers. |
| PWM-PR | 110 | Implements a pseudo-random PWM using a 16-bit linear feedback shift register (LFSR). |

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the TCPWM_CNTx_CTRL register, as shown in **Table 19-6**.

**Table 19-6. Counting mode configuration**

| Counting modes | UP_DOWN_ MODE[17:16] | Description |
|---|---|---|
| UP Counting Mode | 00 | Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when counter reaches the period value. |
| DOWN Counting Mode | 01 | Decrements the counter from the period value until 0 is reached. A TC condition is generated when the counter reaches ‘0’. |
| UP/DOWN Counting Mode 0 | 10 | Increments the counter until the period value is reached, and then decrements the counter until ‘0’ is reached. A TC condition is generated only when ‘0’ is reached. |
| UP/DOWN Counting Mode 1 | 11 | Similar to up/down counting mode 0 but a TC condition is generated when the counter reaches ‘0’ and when the counter value reaches the period value. |

### 19.3.1 Timer mode

The timer mode is commonly used to measure time of occurrence of an event or to measure the time difference between two events.

#### 19.3.1.1 Block diagram

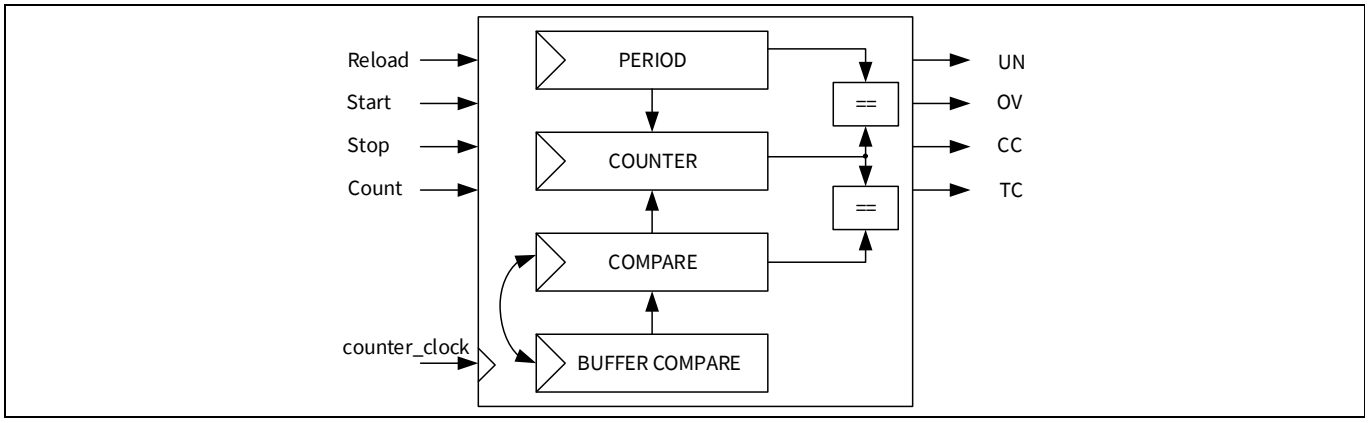



**Figure 19-4. Timer mode block diagram**

#### 19.3.1.2 How it works

The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode.

The following describes the working of the timer:

- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER).

*Note: It is not recommended to write values to this register while the counter is running.*

  - The period value for the timer is stored in the period register.
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with the value in the period register.
  - In the up/down counting modes, the count register value is not updated upon reaching the terminal values. Instead the direction of counting changes when the count value reaches 0 or the period value.
- The CC condition is generated when the count register value equals the compare register value. Upon this condition, the compare register and buffer compare register switch their values if enabled by the AUTO_RELOAD_CC bit-field of the counter control (TCPWM_CNTx_CTRL) register. This condition can be used to generate an interrupt request.

**Figure 19-5** shows the timer operational mode of the counter in four different counting modes. The period register contains the maximum counter value.

- In the up counting mode, a period value of A results in A+1 counter cycles (0 to A).
- In the down counting mode, a period value of A results in A+1 counter cycles (A to 0).
- In the two up/down counting modes (both modes 0 and 1 both), a period value of A results in 2*A counter cycles (0 to A and back to 0).

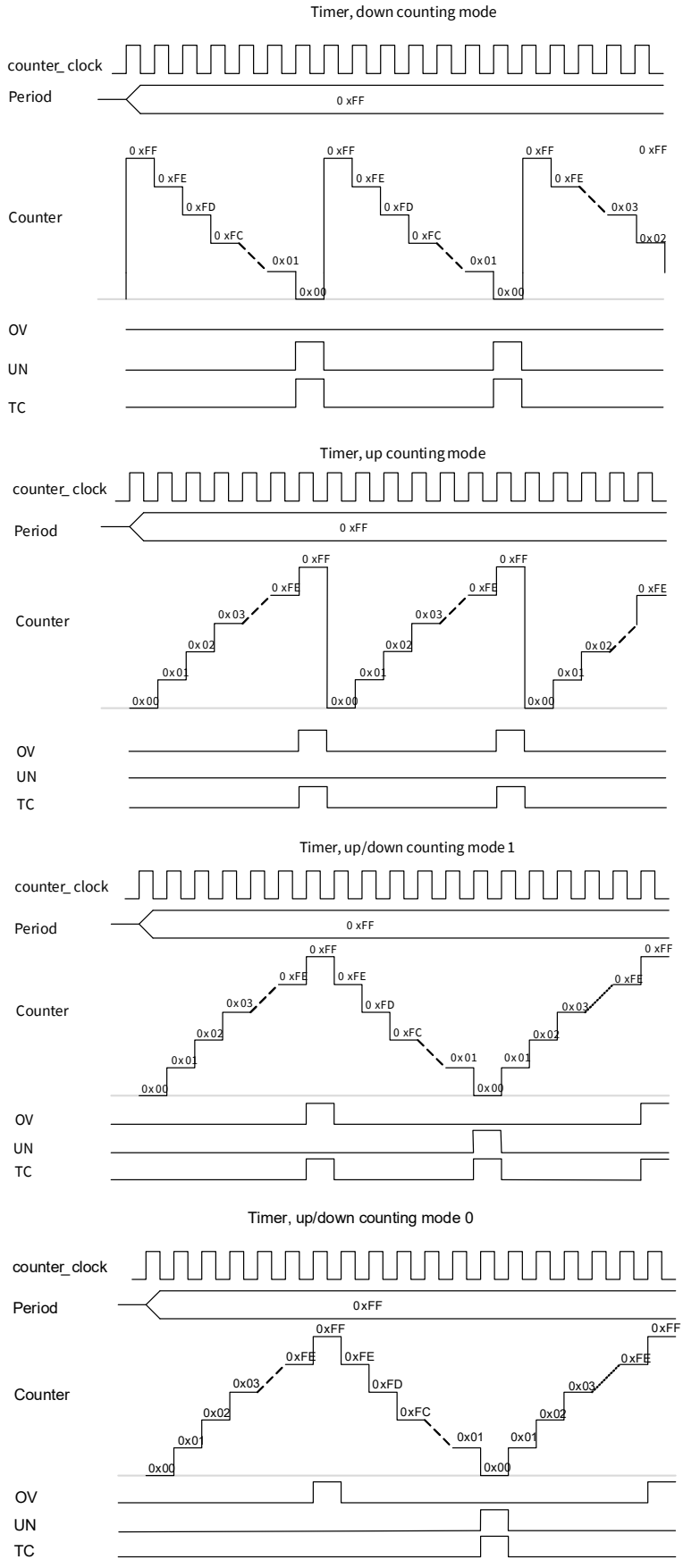



**Figure 19-5. Timing diagram for timer in multiple counting modes**

*Note: The OV and UN signals remain at logic high for one cycle of the HFCLCK, as explained in **"Signals upon trigger conditions"** on page 159. The figures in this chapter assumes that HFCLK and counter clock are the same.*

#### 19.3.1.3 Configuring counter for timer mode

The steps to configure the counter for Timer mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register. Set AUTO_RELOAD_CC field of counter control register, if required to switch values at every CC condition.
5. Set clock pre-scaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-1**.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-6**.
7. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the TCPWM_CNTx_CTRL register.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
10. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 159.
11. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

### 19.3.2 Capture mode

In the capture mode, the counter value can be captured at any time either through a firmware write to command register (TCPWM_CMD) or a capture trigger input. This mode is used for period and pulse width measurement.

#### 19.3.2.1 Block diagram

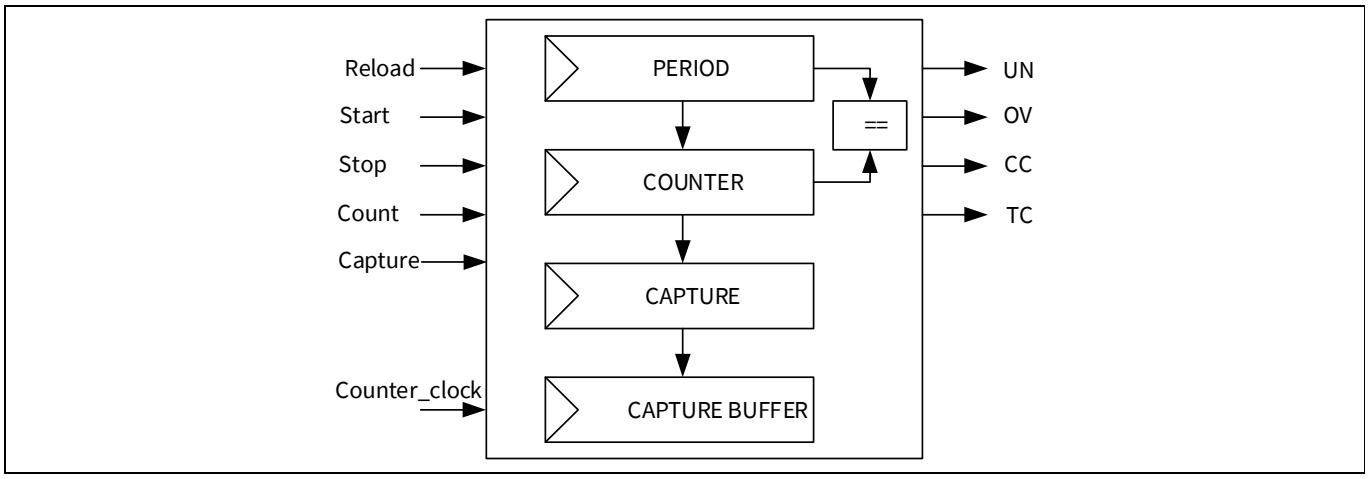



**Figure 19-6. Capture mode block diagram**

### 19.3.2.2 How it works

The counter can be set to count in up, down, and up/down counting modes by configuring the UP_DOWN_MODE[17:16] bit-field of the counter control register (TCPWM_CNTx_CTRL).

Operation in capture mode occurs as follows:

- During a capture event, generated either by hardware or software, the current count register value is copied to the capture register (TCPWM_CNTx_CC) and the capture register value is copied to the buffer capture register (TCPWM_CNTx_CC_BUFF).
- A pulse on the CC output signal is generated when the counter value is copied to the capture register. This condition can also be used to generate an interrupt request.

**Figure 19-7** illustrates the capture behavior in the up counting mode.

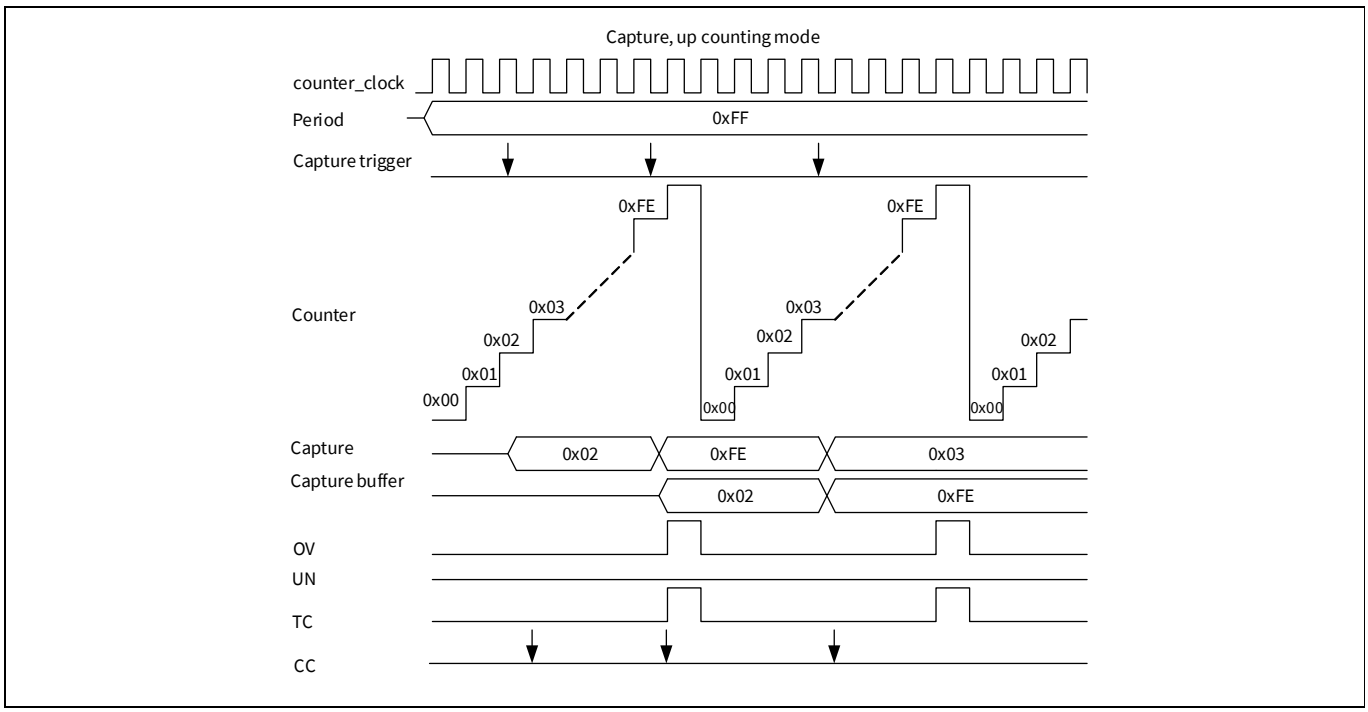



**Figure 19-7. Timing diagram of counter in capture mode, up counting mode**

In the **Figure 19-7**, observe that:

- The period register contains the maximum count value.
- Internal overflow (OV) and TC conditions are generated when the counter reaches the period value.
- A capture event is only possible at the edges or through software. Use trigger control register 1 to configure the edge detection.
- Multiple capture events in a single clock cycle are handled as:
  - Even number of capture events – no event is observed
  - Odd number of capture events – single event is observed

This happens when the capture signal frequency is greater than the counter_clock frequency.

### 19.3.2.3 Configuring counter for capture mode

The steps to configure the counter for Capture mode operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Capture mode by writing '010' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set clock pre-scaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-1**.
5. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-6**.
6. Counter can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the TCPWM_CNTx_CTRL register.
7. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, Capture, and Count).
8. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Stop, Capture, and Count).
9. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 159.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

## 19.3.3 Quadrature decoder mode

Quadrature decoders are used to determine speed and position of a rotary device (such as servo motors, volume control wheels, and PC mice). The quadrature encoder signals are used as phiA and phiB inputs to the decoder.

### 19.3.3.1 Block diagram

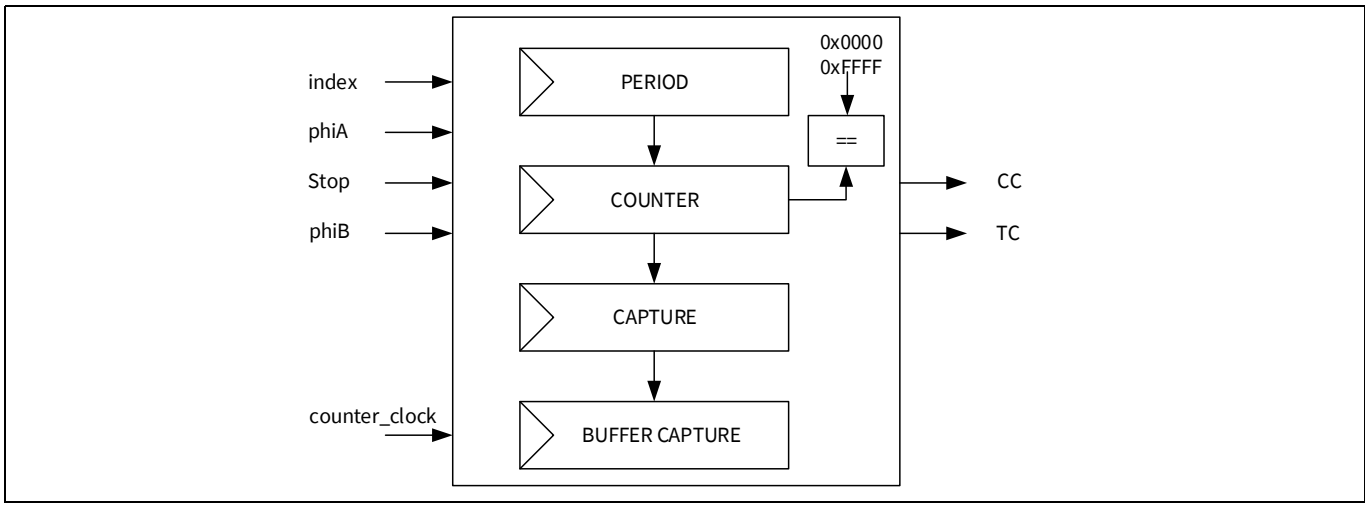



**Figure 19-8. Quadrature mode block diagram**

### 19.3.3.2 How it works

Quadrature decoding only runs on counter_clock. It can operate in three sub-modes: X1, X2, and X4. These encoding modes can be controlled by the QUADRATURE_MODE[21:20] field of the counter control register (TCPWM_CNTx_CTRL). This mode uses double buffered capture registers.

The Quadrature mode operation occurs as follows:

- Quadrature phases phiA and phiB: Counting direction is determined by the phase relationship between phiA and phiB. These phases are connected to the count and the start trigger inputs, respectively as hardware input to the decoder.
- Quadrature index signal: This is connected to the reload signal as a hardware input. This event generates a TC condition, as shown in **Figure 19-9**.
  On TC, the counter is set to 0x0000 (in the up counting mode) or to the period value (in the down counting mode).

*Note: The down counting mode is recommended to be used with a period value of 0x8000 (the mid-point value).*

- A pulse on CC output signal is generated when the count register value reaches 0x0000 or 0xFFFF. On a CC condition, the count register is set to the period value (0x8000 in this case).
- On TC or CC condition:
  - Count register value is copied to the capture register
  - Capture register value is copied to the buffer capture register
  - This condition can be used to generate an interrupt request
- The value in the capture register can be used to determine which condition caused the event and whether:
  - A counter underflow occurred (value 0)
  - A counter overflow occurred (value 0xFFFF)
  - An index/TC event occurred (value is not equal to either 0 or 0xFFFF)
- The DOWN bit field of counter status (TCPWM_CNTx_STATUS) register can be read to determine the current counting direction. Value '0' indicates a previous increment operation and value '1' indicates previous decrement operation. **Figure 19-9** illustrates quadrature behavior in the X1 encoding mode.
  - A positive edge on phiA increments the counter when phiB is '0' and decrements the counter when phiB is '1'.
  - The count register is initialized with the period value on an index/reload event.
  - Terminal count is generated when the counter is initialized by index event. This event can be used to generate an interrupt.
  - When the count register reaches 0xFFFF (the maximum count register value), the count register value is copied to the capture register and the count register is initialized with period value (0x8000).

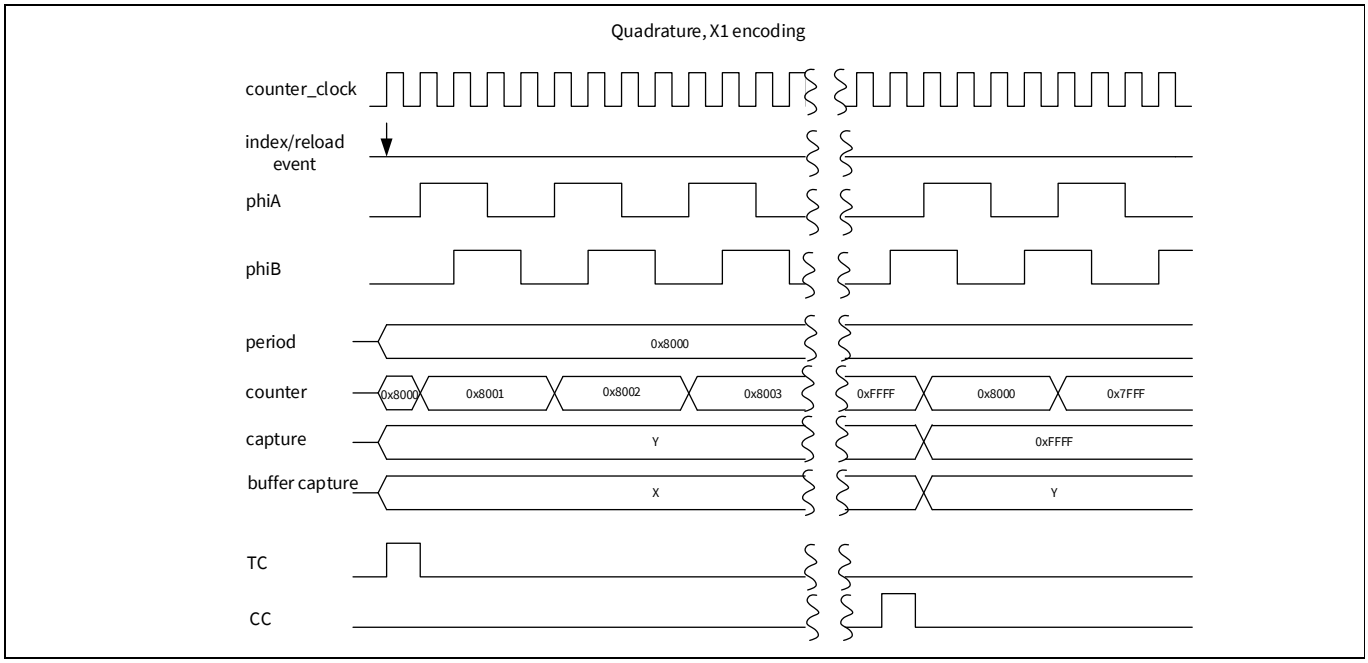



**Figure 19-9. Timing diagram for quadrature mode, X1 encoding**

The quadrature phases are detected on the counter_clock. Within a single counter period, the phases should not change value more than once.

The X2 and X4 quadrature encoding modes count twice and four times as fast as the X1 encoding mode.

**Figure 19-10** illustrates the quadrature mode behavior in the X2 and X4 encoding modes.

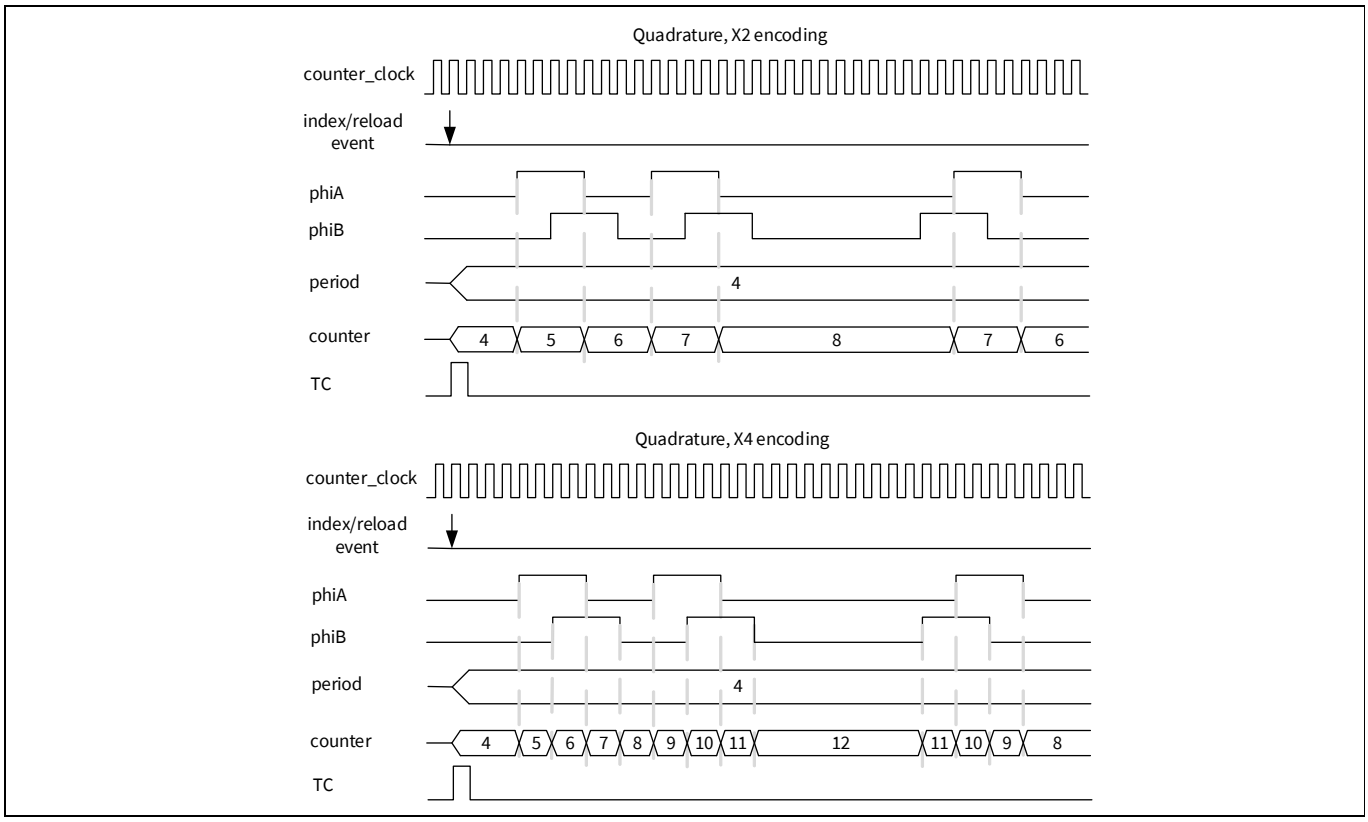



**Figure 19-10. Timing diagram for quadrature mode, X2 and X4 encoding**

### 19.3.3.3 Configuring counter for quadrature mode

The steps to configure the counter for quadrature mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Quadrature mode by writing '011' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set the required encoding mode by writing to the QUADRATURE_MODE[21:20] field of the TCPWM_CNTx_CTRL register.
5. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Index and Stop).
6. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Index and Stop).
7. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 159.
8. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

## 19.3.4 PWM mode

The PWM mode is also called the digital comparator mode. The comparison output is a PWM signal whose period depends on the period register value and duty cycle depends on the compare and period register values.

PWM period = (period value/counter clock frequency) in left- and right-aligned modes

PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

Duty cycle = (compare value/period value) in left- and right-aligned modes

Duty cycle = ((period value-compare value)/period value) in center-aligned mode

### 19.3.4.1 Block diagram

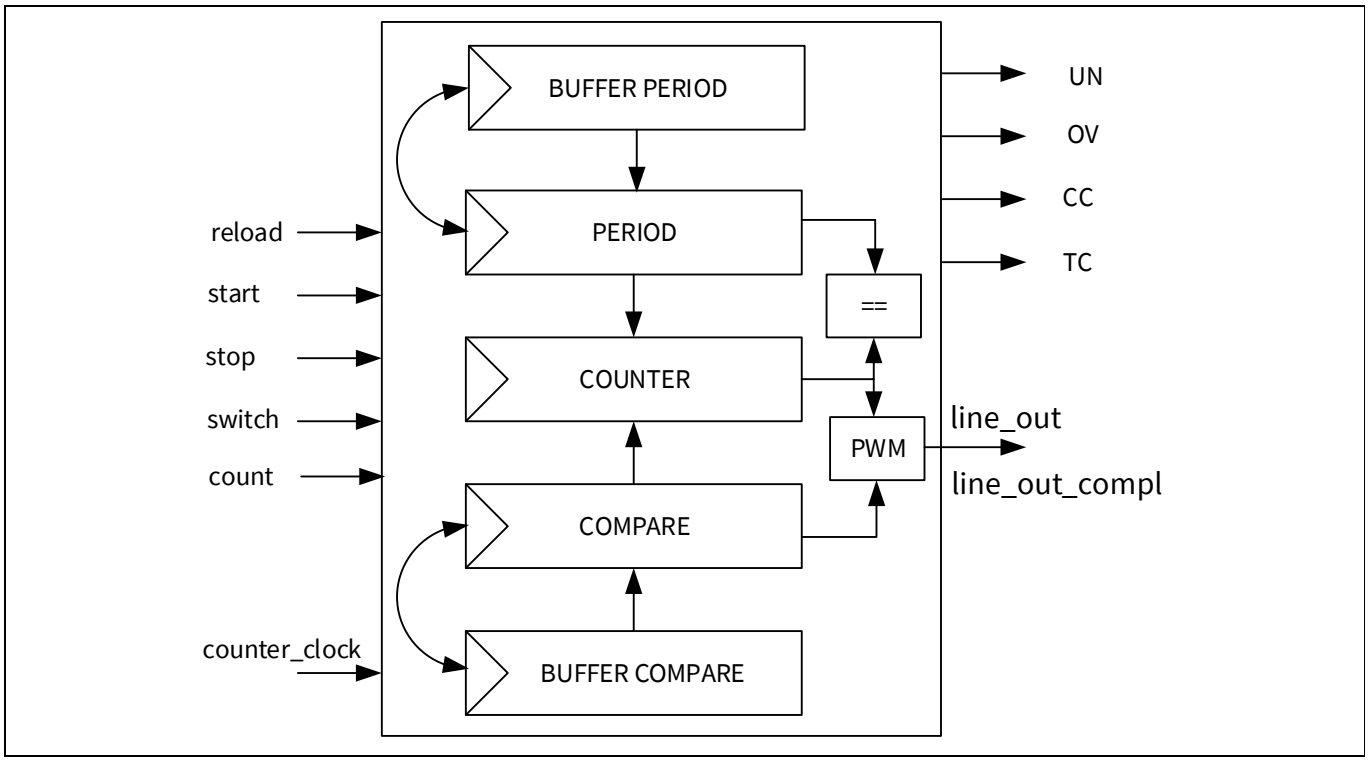



**Figure 19-11. PWM mode block diagram**

### 19.3.4.2 How it works

The PWM mode can output left, right, center, or asymmetrically aligned PWM signals. The desired output alignment is achieved by using the counter's up, down, and up/down counting modes selected using UP_DOWN_MODE [17:16] bits in the TCPWM_CNTx_CTRL register, as shown in **Table 19-6**.

This CC signal along with OV and UN signals control the PWM output line. The signals can toggle the output line or set it to a logic '0' or '1' by configuring the TCPWM_CNTx_TR_CTRL2 register. By configuring how the signals impact the output line, the desired PWM output alignment can be obtained.

The recommended way to modify the duty cycle is:

- The buffer period register and buffer compare register are updated with new values.
- On TC, the period and compare registers are automatically updated with the buffer period and buffer compare registers when there is an active switch event. The AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register are set to ‘1’. When a switch event is detected, it is remembered until the next TC event. Pass through signal (selected during event detection setting) cannot trigger a switch event.
- Updates to the buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in **Figure 19-13**.

In the center-aligned mode, the output line is set to '0' at TC and toggled at the CC condition.

At the reload event, the count register is initialized and starts counting in the appropriate mode. At every count, the count register value is compared with compare register value to generate the CC signal on match.

**Figure 19-12** illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

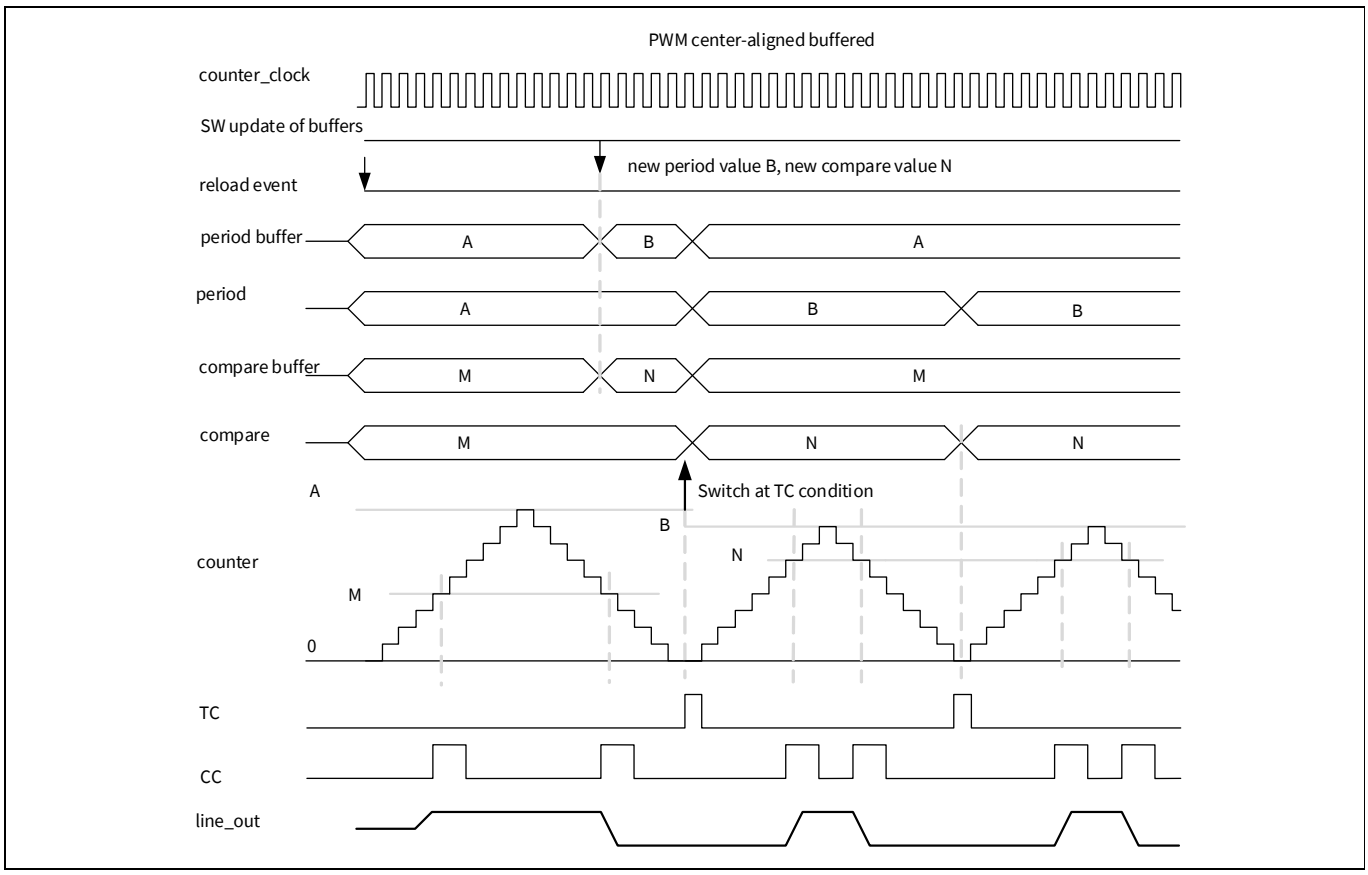



**Figure 19-12. Timing diagram for center aligned PWM**

Figure 19-13 illustrates center-aligned PWM with software generated switch events:

- Software generates a switch event only after both the period buffer and compare buffer registers are updated.
- Because the updates of the second PWM pulse come late (after the terminal count), the first PWM pulse is repeated.
- Note that the switch event is automatically cleared by hardware at TC after the event takes effect.

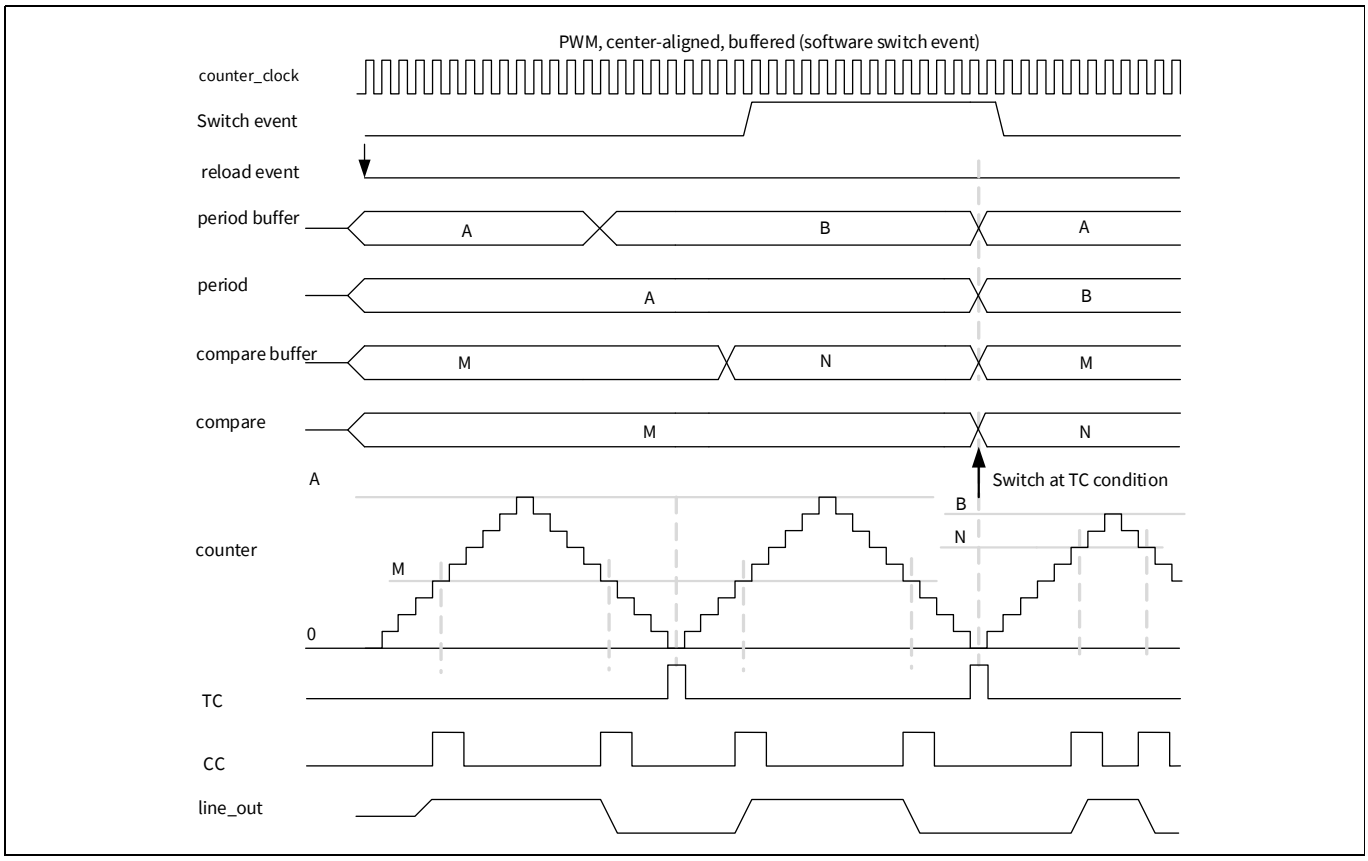



**Figure 19-13. Timing diagram for center aligned PWM (software switch event)**

### 19.3.4.3 Other configurations

- For asymmetric PWM, the up/down counting mode 1 should be used. This causes a TC when the counter reaches either ‘0’ or the period value. To create an asymmetric PWM, the compare register is changed at every TC (when the counter reaches either ‘0’ or the period value), whereas the period register is only changed at every other TC (only when the counter reaches ‘0’).
- For left-aligned PWM, use the up counting mode; configure the OV condition to set output line to '1' and CC condition to reset the output line to '0'. See **Table 19-3**.
- For right-aligned PWM, use the down counting mode; configure UN condition to reset output line to '0' and CC condition to set the output line to '1'. See **Table 19-3**.

### 19.3.4.4 Kill feature

Kill feature gives the ability to disable both output lines immediately. This event can be programmed to stop the counter by modifying the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the counter control register, as shown in **Table 19-7**.

**Table 19-7. Field setting for stop on kill feature**

| PWM_STOP_ON_KILL field | Comments |
|---|---|
| 0 | The kill trigger temporarily blocks the PWM output line but the counter is still running. |
| 1 | The kill trigger temporarily blocks the PWM output line and the counter is also stopped. |

A kill event can be programmed to be asynchronous or synchronous, as shown in **Table 19-8**.

**Table 19-8. Field setting for synchronous/asynchronous kill**

| PWM_SYNC_KILL field | Comments |
|---|---|
| 0 | An asynchronous kill event lasts as long as it is present. This event requires pass through mode. |
| 1 | A synchronous kill event disables the output lines until the next TC event. This event requires rising edge mode. |

In the synchronous kill, PWM cannot be started before the next TC. To restart the PWM immediately after the kill input is removed, the kill event should be asynchronous (see **Table 19-8**). The generated stop event disables both output lines. In this case, the reload event can use the same trigger input signal but should be used in falling edge detection mode.

### 19.3.4.5 Configuring counter for PWM mode

The steps to configure the counter for the PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set clock pre-scaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-1**.
4. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register and the buffer period value in the TCPWM_CNTx_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in **Table 19-6**.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNTx_CTRL register as required.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line _out_compl can be controlled by the TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 159.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

### 19.3.5 Pulse Width Modulation with dead time mode

Dead time is used to delay the transitions of both ‘line_out’ and ‘line_out_compl’ signals. It separates the transition edges of these two signals by a specified time interval. Two complementary output lines 'dt_line' and 'dt_line_compl' are derived from these two lines. During the dead band period, both compare output and complement compare output are at logic ‘0’ for a fixed period. The dead band feature allows the generation of two non-overlapping PWM pulses. A maximum dead time of 255 clocks can be generated using this feature.

#### 19.3.5.1 Block diagram

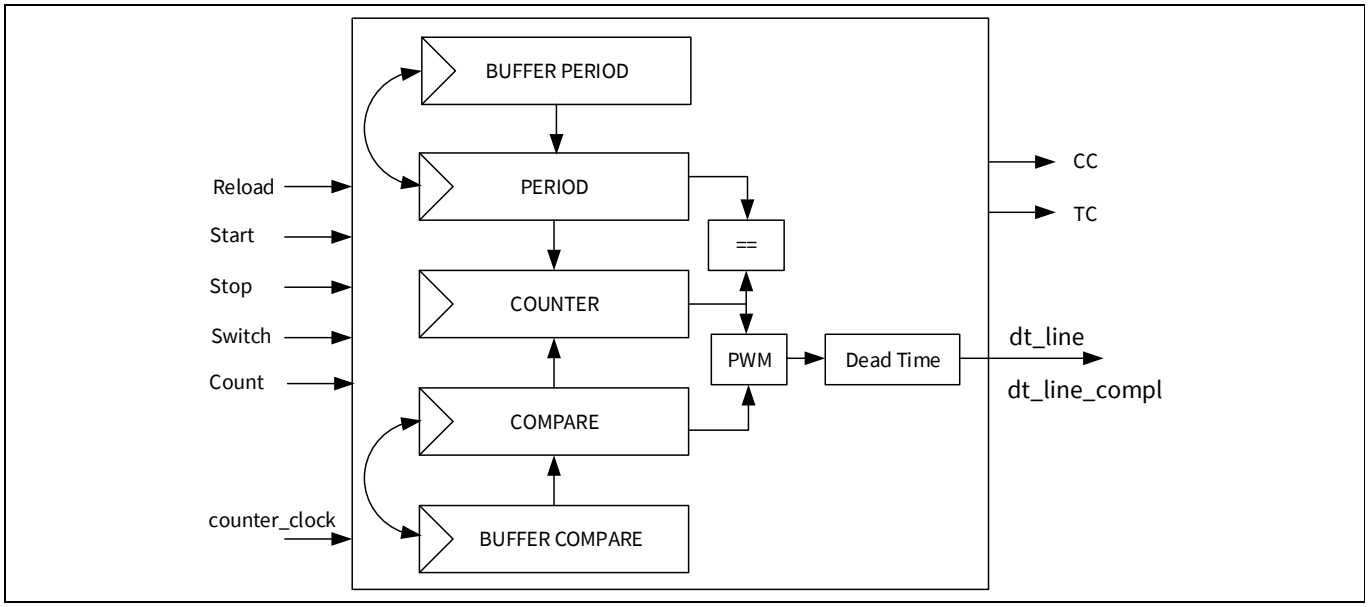



**Figure 19-14. PWM-DT mode block diagram**

#### 19.3.5.2 How it works

The PWM operation with Dead Time mode occurs as follows:

- On the rising edge of the PWM line_out, depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to '0'.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period is complete, dt_line is set to '1'.
- On the falling edge of the PWM line_out depending upon UN, OV, and CC conditions, the dead time block sets the dt_line and dt_line_compl to '0'.
- The dead band period is loaded and counted for the period configured in the register.
- When the dead band period has completed, dt_line_compl_is set to '1'.
- A dead band period of zero has no effect on the dt_line and is the same as line_out.
- When the duration of the dead time equals or exceeds the width of a pulse, the pulse is removed.

This mode follows PWM mode and supports the following features available with that mode:

- Various output alignment modes
- Two complementary output lines, dt_line and dt_line_compl, derived from PWM line_out and line _out_compl, respectively
  - Stop/kill event with synchronous and asynchronous modes
  - Conditional switch event for compare and buffer compare registers and period and buffer period registers

This mode does not support clock pre-scaling.

Figure 19-15 illustrates how the complementary output lines dt_line and dt_line_compl are generated from the PWM output line, line_out.

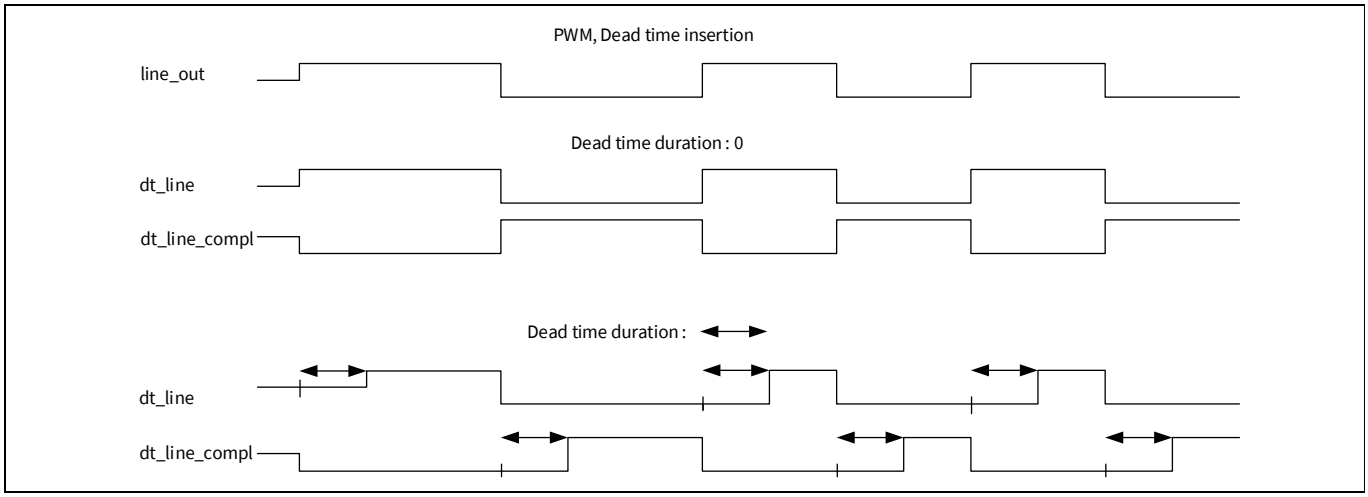



**Figure 19-15. Timing diagram for PWM, with and without dead time**

### 19.3.5.3 Configuring counter for PWM with dead time mode

The steps to configure the counter for PWM with Dead Time mode of operation and the affected register bits are as follows:

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM with Dead Time mode by writing '101' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required dead time by writing to the GENERIC[15:8] field of the TCPWM_CNTx_CTRL register, as shown in **Table 19-1**.
4. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register and the buffer period value in the TCPWM_CNTx_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in **Table 19-6**.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNTx_CTRL register as required, as shown in the **"PWM mode"** on page 169.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. dt_line and dt_line_compl can be controlled by the TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in **"Interrupts"** on page 159.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if hardware start signal is not enabled.

### 19.3.6 Pulse width modulation pseudo-random mode

This mode uses the linear feedback shift register (LFSR), whose input bit is a linear function of its previous state.

#### 19.3.6.1 Block diagram

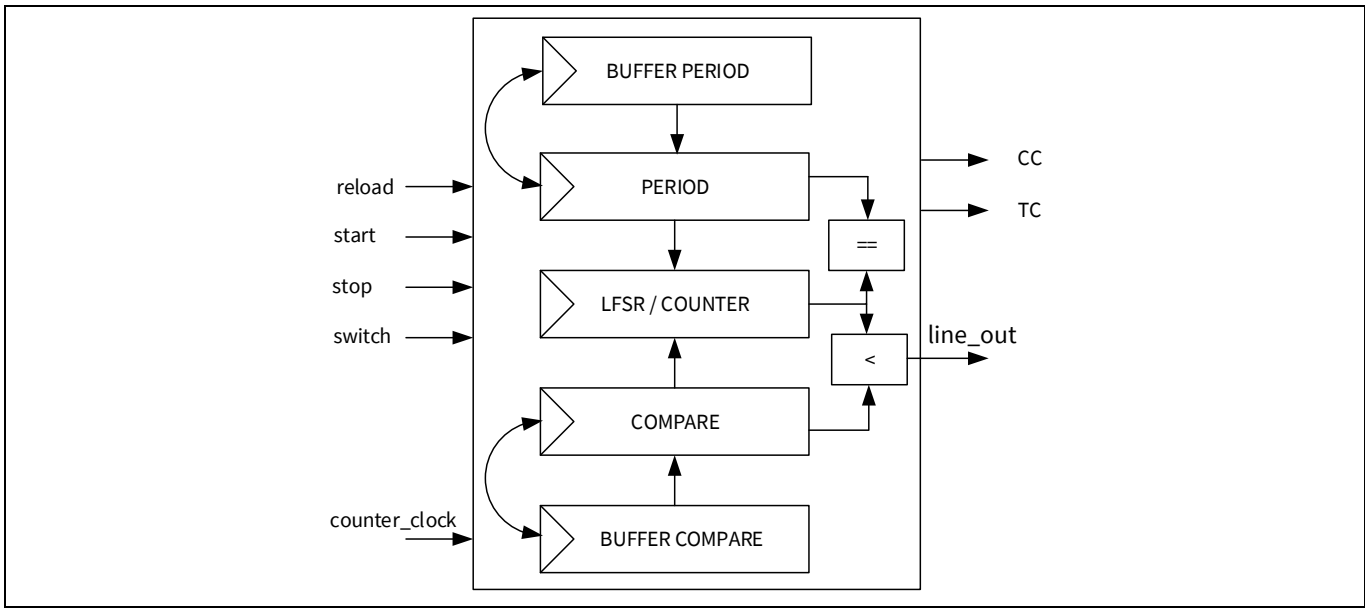



**Figure 19-16. PWM-PR mode block diagram**

#### 19.3.6.2 How it works

The counter register is used to implement LFSR with the polynomial: $x^{16}+x^{14}+x^{13}+x^{11}+1$. It generates all the numbers in the range [1, 0xFFFF] in a pseudo-random sequence. Note that the counter register should be initialized with a non-zero value.

The following steps describe the process:

- The PWM output line ‘line_out’ is driven with '1' when the lower 15-bit value of the counter register is smaller than the value in the compare register (when counter[14:0] < compare[15:0]). A compare value of ‘0x8000’ or higher always results in a '1' on the PWM output line. A compare value of ‘0’ always results in a '0' on the PWM output line.
- A reload event behaves similar to a start event; however, it does not initialize the counter.
- Terminal count is generated when the counter value equals the period value. LFSR generates a predictable pattern of counter values for a certain initial value. This predictability can be used to calculate the counter value after a certain amount of LFSR iterations ‘n’. This calculated counter value can be used as a period value and the TC is generated after ‘n’ iterations.
- At TC, a switch/capture event conditionally switches the compare and period register pairs (based on the AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register).
- A kill event can be programmed to stop the counter as described in previous sections.
- One shot mode can be configured by setting the ONE_SHOT field of the counter control register. At terminal count, the counter is stopped by hardware.
- In this mode, underflow, overflow, and trigger condition events do not occur.
- CC condition occurs when the counter is running and its value equals compare value. **Figure 19-17** illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50 percent duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

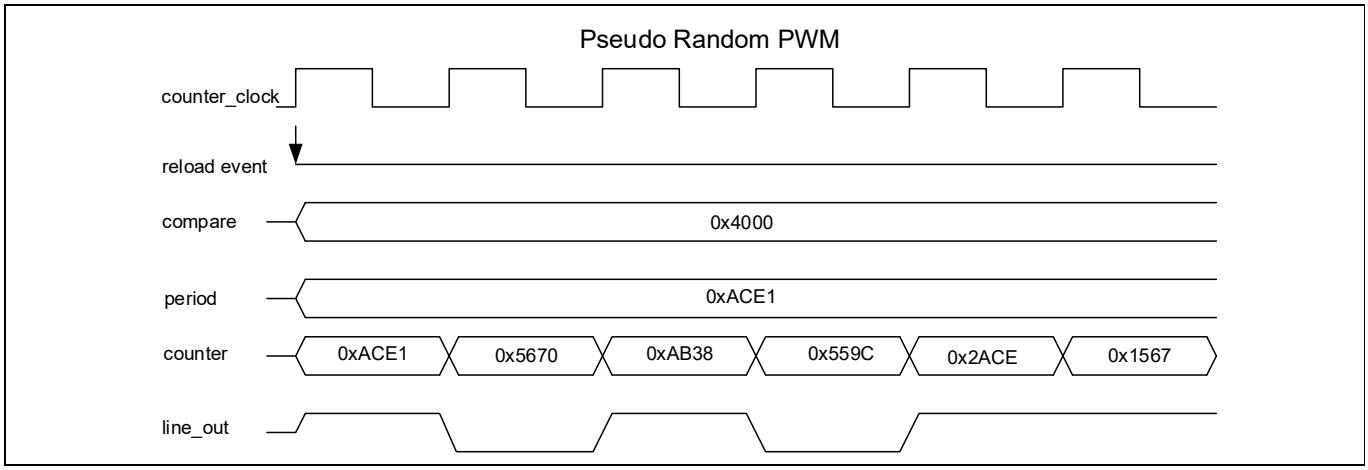



**Figure 19-17. Timing diagram for pseudo-random PWM**

A capture/switch input signal may switch the values between the compare and compare buffer registers and the period and period buffer registers. This functionality can be used to modulate between two different compare values using a trigger input signal to control the modulation.

*Note: Capture/switch input signal can only be triggered by an edge (rising, falling, or both). This input signal is remembered until the next terminal count.*

### 19.3.6.3 Configuring counter for pseudo-random PWM mode

The steps to configure the counter for pseudo-random PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNTx_PERIOD register and buffer period value in the TCPWM_CNTx_PERIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNTx_CTRL register as required.
6. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in **“Interrupts”** on page 159.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

## 19.4 Registers

**Table 19-9. TCPWM registers**

| Register | Comment | Features |
|---|---|---|
| TCPWM_CTRL | TCPWM control register | Enables the counter block |
| TCPWM_CMD | TCPWM command register | Generates software events |
| TCPWM_INTR_CAUSE | TCPWM counter interrupt cause register | Determines the source of the combined interrupt signal |
| TCPWM_CNTx_CTRL | Counter control register | Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock pre-scaling, and counting direction |
| TCPWM_CNTx_STATUS | Counter status register | Reads the direction of counting, dead time duration, and clock pre-scaling; checks if counter is running |
| TCPWM_CNTx_COUNTER | Count register | Contains the 16-bit counter value |
| TCPWM_CNTx_CC | Counter compare/capture register | Captures the counter value or compares the value with the counter value |
| TCPWM_CNTx_CC_BUFF | Counter buffered compare/capture register | Buffer register for counter CC register; switches compare value |
| TCPWM_CNTx_PERIOD | Counter period register | Contains upper value of the counter |
| TCPWM_CNTx_PERIOD_BUFF | Counter buffered period register | Buffer register for counter period register; switches period value |
| TCPWM_CNTx_TR_CTRL0 | Counter trigger control register 0 | Selects trigger for specific counter events |
| TCPWM_CNTx_TR_CTRL1 | Counter trigger control register 1 | Determines edge detection for specific counter input signals |
| TCPWM_CNTx_TR_CTRL2 | Counter trigger control register 2 | Controls counter output lines upon CC, OV, and UN conditions |
| TCPWM_CNTx_INTR | Interrupt request register | Sets the register bit when TC or CC condition is detected |
| TCPWM_CNTx_INTR_SET | Interrupt set request register | Sets the corresponding bits in the interrupt request register |
| TCPWM_CNTx_INTR_MASK | Interrupt mask register | Mask for interrupt request register |
| TCPWM_CNTx_INTR_MASKED | Interrupt masked request register | Bitwise AND of interrupt request and mask registers |

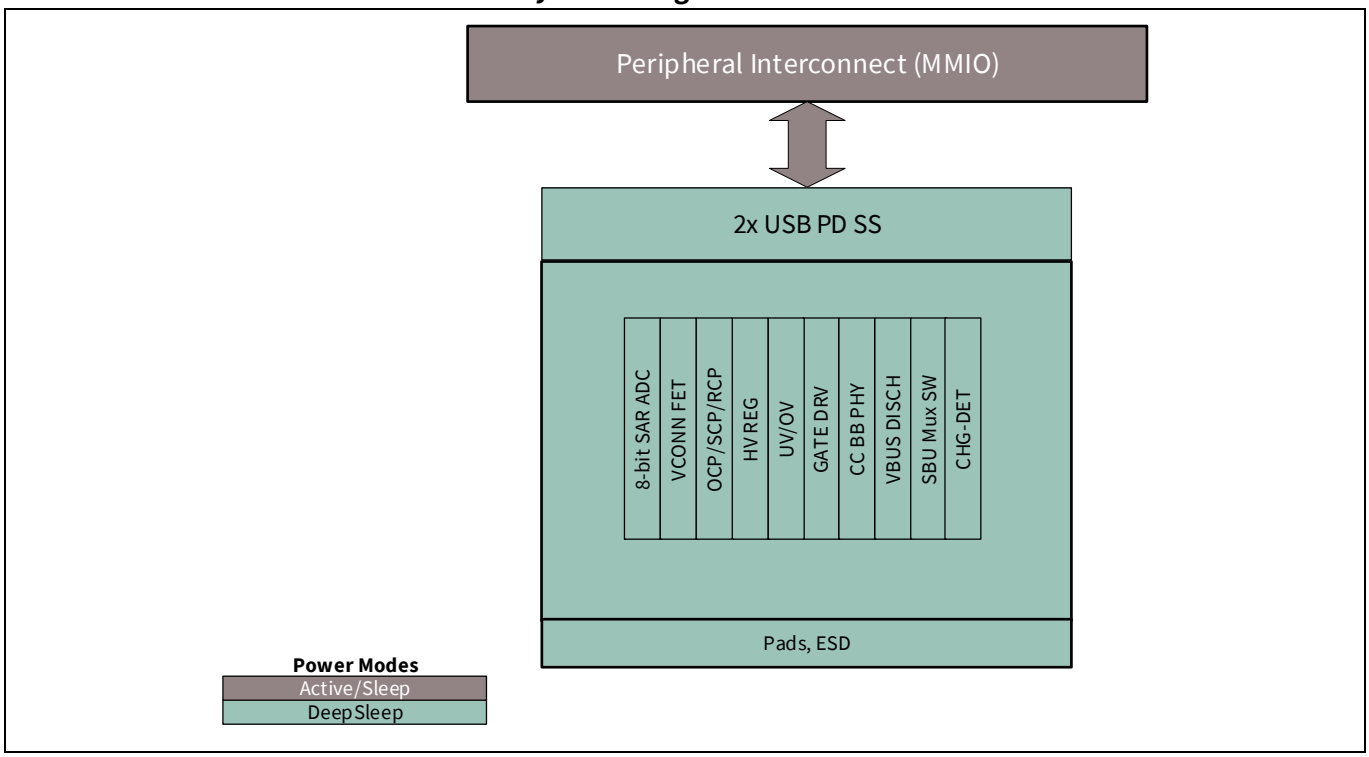
# Section F: USB power and data

This section encompasses the following chapter:


- **"USB Power Delivery (USB PD)"** on page 179
- **"USB Full Speed (USB FS)"** on page 180


## Top level architecture

**EZ-PD™ PMG1-S3 MCU USB Power Delivery block diagram**



# 20 USB Power Delivery (USB PD)

EZ-PD™ PMG1-S3 MCU has a built-in USB Power Delivery (USB PD) physical layer block which supports all the features of the USB Power Delivery 3.0 specification and USB Type-C cable and connector specification 1.3. EZ-PD™ PMG1-S3 MCU supports two USB PD port in 97-BGA package and one USB PD port in 48-QFN package.

The USB PD block implements the PD communication using a dedicated Communication Channel (CC) using Bi-phase Mark Coding (BMC). The block contains a dedicated SRAM to store USB PD RX and TX header and messages coming as a part of USB PD negotiations. It also includes the termination registers, Rp and Rd, required to implement the Type-C roles such as DFP, UFP, and DRP. The USB PD block also has up to two 8-bit SAR ADCs, one per port.

EZ-PD™ PMG1-S3 MCU also supports slew rate controlled (turn-on only) NFET gate driver for the external consumer path FET and has a load switch controller with integrated FET (5 V/3 A) in the VBUS provider path with over current, reverse current, and short circuit protection.

## 20.1 Features

The USB PD block has the following features:

- USB PD PHY implementation according to the USB Power Delivery 3.0 specification
- Complete USB PD BMC transceiver
  - BMC encoder with 300 Kbps bit rate
  - Programmable digital implementation of BMC decoder
- Power Delivery PHY and protocol layer
  - 4b/5b encoding/decoding
  - SOP/EOP detection and generation
  - 32-bit CRC generation and compare
  - 64 bytes TX and 64 bytes RX SRAM to hold USB PD header and message
  - Interrupts to report key events (CC valid data, hard reset, message received, message error, and plug insertion)
  - BIST support according to USB Power Delivery 3.0 specification
  - Automatic good CRC message transmission per data/control type
  - USBPD extended data messaging
  - USBPD data chunking
- Integrated DFP (Rp) and UFP (Rd) termination resistors
  - Dead battery Rd termination support
- Cable detection
  - Type-C connector identification
- AHB Lite interface
- Hot Plug Detect (HPD) functionality according to VESA DisplayPort Alt Mode on USB Type-C Standard
- Supports Deep-Sleep mode when block is not transmitting or receiving packets
- Integrated USB Analog Mux
- Integrated SBU Analog pass-through switches
- Integrated OV (over voltage) protection on both provider and consumer paths
- Integrated OCP protection on the VCONN
- Integrated RC (reverse current), SC (short circuit), and OC (over current) protection on VBUS provider path
- Over temperature thermal shutdown

Use the EZ-PD™ configurator provided by the ModusToolbox™ (MTB) software IDE to implement the USB PD block in designs. See the **AN232553 - Getting started with EZ-PD™ PMG1 MCU on ModusToolbox™ software** application note for more details.

# 21 USB Full Speed (USB FS)

The EZ-PD™ PMG1-S3 MCU USB block acts as a USB device that communicates with a USB host. The USB block is available as a fixed-function digital block in the EZ-PD™ PMG1-S3 MCU device. It supports full-speed communication (12 Mbps) and is designed to be compliant with the USB Specification Revision.2.0. USB devices can be designed for plug and play applications with the host and also support hot swapping. This chapter details the EZ-PD™ PMG1-S3 MCU USB block and transfer modes. For details about the USB specification, see the USB Implementers Forum web site.

## 21.1 Features

The EZ-PD™ PMG1-S3 MCU USB has these features:

- Complies with USB specification 2.0
- Supports full-speed peripheral device operation with a signaling bit rate of 12 Mbps
- Supports eight data endpoints and one control endpoint
- Supports four types of transfers – bulk, interrupt, isochronous, and control
- Supports bus- and self-powered configurations
- USB suspend mode for low power
- Supports two types of logical transfer modes:
  - Store and Forward mode
  - Cut Through mode
- Differential signal (D+ and D-) output
- Integrated 22 Ω USB termination resistors on D+ and D– lines, and 1.5-kΩ pull-up resistor on the D+ line
- Battery charger detection (BCD) compliant with USB Battery Charging Specification Rev 1.2 (peripheral detect only)
- Supports maximum packet size of 64 bytes using the Store and Forward mode and maximum packet size of 1023 for isochronous transfer using the Cut Through mode

## 21.2 Block diagram

Figure 21-1 illustrates the architecture of the USB block. It consists of the USB Physical Layer (USB PHY), Serial Interface Engine (SIE), Arbiter, and the local 512-byte memory buffer.

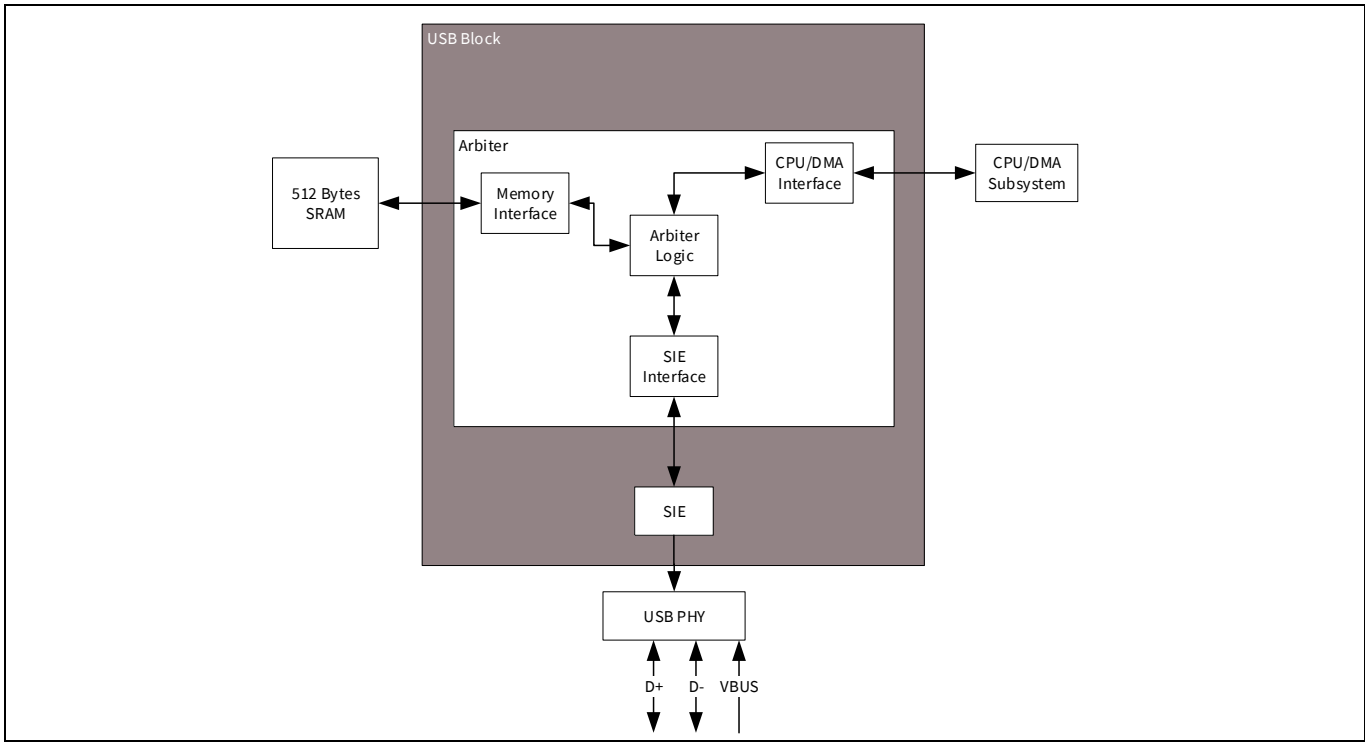



**Figure 21-1. USB block diagram**

### 21.2.1 USB physical layer (USB PHY)

This block takes care of physical layer communication with the USB host through the D+, D-, and VBUS pins. It handles the differential mode communication with the host, VBUS detection, and monitoring events such as SE0 on the USB bus.

### 21.2.2 Serial interface engine (SIE)

The Serial Interface Engine (SIE) is responsible for handling the decoding and creating of data and control packets during transmit and receive. It decodes the USB bit streams into USB packets during receive and creates USB bit streams during transmit. The following are the features of the SIE block:

- Conforms to the USB 2.0 specification
- Supports one device address
- Supports eight data endpoints and control endpoint
- Supports interrupt trigger event for each endpoint
- Integrates an 8-byte buffer in the control endpoint

The registers for this block are mainly used to configure the data endpoint operations and the control endpoint data buffers. This block also controls the interrupt events available for each endpoint.

### 21.2.3 Arbiter

The Arbiter is the block that handles access of the SRAM memory by the endpoints. The SRAM memory can be accessed by the CPU, DMA, or the SIE. The Arbiter handles the arbitration between the CPU, DMA, and the SIE. The Arbiter consists of the following blocks:

- SIE Interface Module
- CPU/DMA Interface Module
- Memory Interface
- Arbiter Logic

The Arbiter registers are used to handle the endpoint configurations, Read address, and Write address for the endpoints. It also configures the logical transfer type required for each endpoint.

#### 21.2.3.1 SIE interface module

This module handles all the transactions with the SIE block. The SIE reads data from the SRAM memory and transmits to the host. Similarly, it writes the data received from the host to the SRAM memory. These requests are registered in the SIE Interface module and are handled by this block.

#### 21.2.3.2 CPU/DMA interface block

This module handles all the transactions with the CPU and DMA. The CPU makes requests for the reads and writes to the SRAM memory for each endpoint. These requests are registered in this interface and are handled by the block. When DMA is configured, this interface is responsible for all transactions back and forth between the DMA and USB. The block supports the DMA request line for each data endpoint. The behavior of the DMA depends on the type of logical transfer mode configured in the Configuration register.

#### 21.2.3.3 Memory interface

The memory interface is used to control the interface between the USB block and the SRAM memory unit. The maximum memory size supported is 512 bytes organized as 256 x a 16-bit memory unit. This is a dedicated memory for the USB. The memory access can be requested by the SIE or by the CPU/DMA. The SIE Interface block and the CPU/DMA Interface block handle these requests.

#### 21.2.3.4 Arbiter logic

This is the main block of the Arbiter. It is responsible for arbitrations for all the transactions that happen in the Arbiter. It arbitrates the CPU, DMA, and SIE access to the memory unit and the registers. This block also handles the memory management. The memory management is either “Manual” or “Automatic.” In Manual memory management, the read and write address manipulations are done by the firmware. In Automatic management, all the memory handling is done by this block itself. This block takes care of the buffer size allocation. It also handles common memory area. This block also handles the interrupt requests for each endpoint.

## 21.3 How it works

### 21.3.1 USB physical layer (USB PHY)

The USB block includes the transmitter and the receiver (transceiver), which corresponds to the USB PHY. **Figure 21-2** shows the PHY architecture. The USB PHY in EZ-PD™ PMG1-S3 MCU also includes the pull-up resistor on the D+ line to identity the device as full-speed type to the host. The PHY integrates the 22-Ω series termination resistors on the USB lines. The signal between the USB device and the host is a differential signal. The receiver receives the differential signal from the host and converts it to a single-ended signal for processing by the SIE. The transmitter converts the single-ended signal from the SIE to the differential signal, and transmits it to the host. The differential signal is given to the upstream devices at a nominal voltage range of 0 V to 3.3 V.

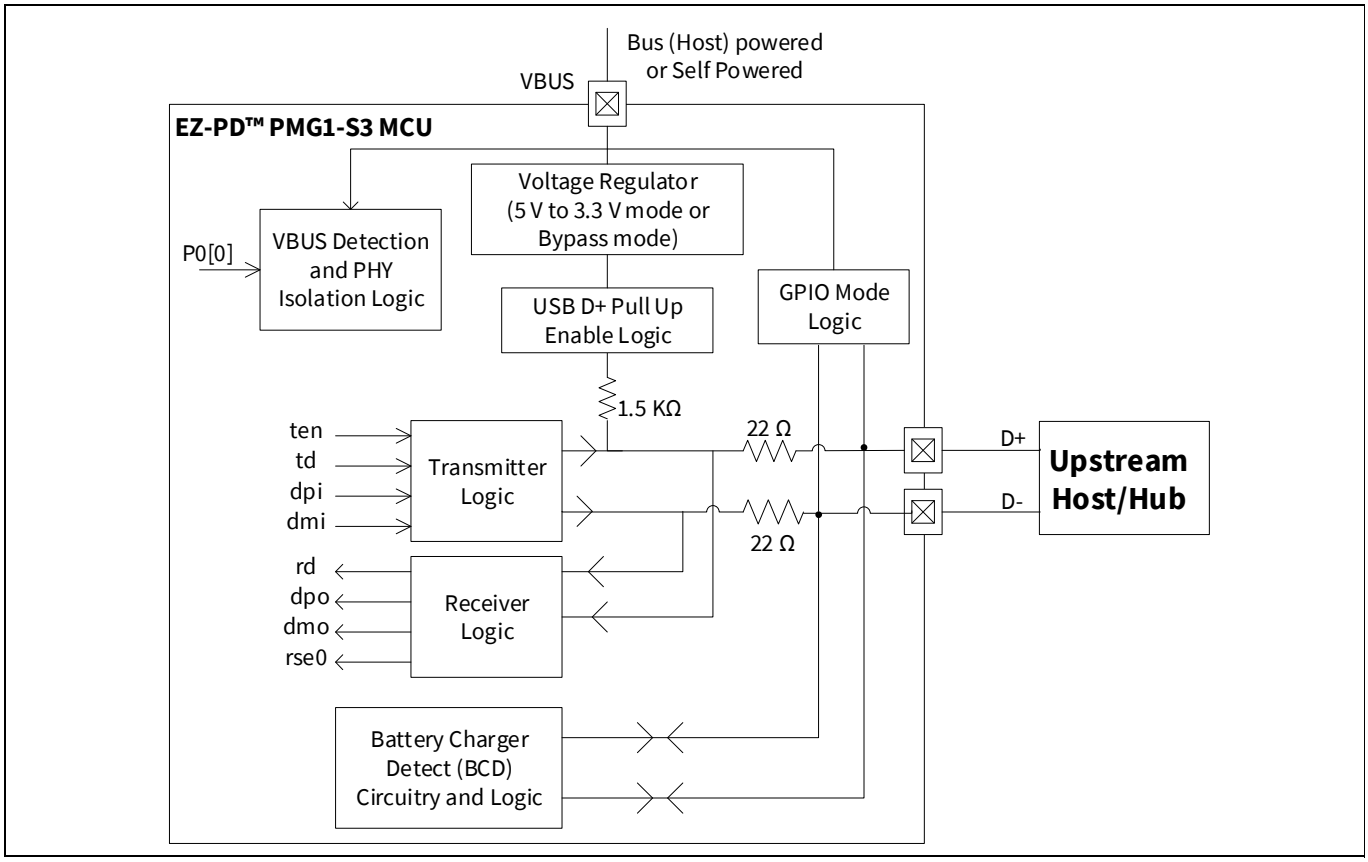



**Figure 21-2. USB PHY architecture**

#### 21.3.1.1 Power scheme

The USB PHY is powered by the VBUS power pad of the EZ-PD™ PMG1-S3 MCU device. The VBUS pad can be driven either by the host VBUS (bus-powered) or an external power supply (self-powered).

The USB PHY needs a nominal voltage of 3.3 V for its communication with the host. The REG_ENABLE bit in the USB_CR1 register controls the operation of the internal voltage regulator. For applications with VBUS supply voltage in the 5-V range (4.35 V to 5.25 V), the REG_ENABLE bit must be set high to enable the internal regulator. For VBUS supply voltage in the 3.3-V range (3.15 V to 3.45 V), this register bit must be cleared to connect the transceiver directly to the supply.

### 21.3.1.2 VBUS detection

USB devices have the option of deriving power from the VBUS power signal from the host (bus-powered configuration) or having an external power supply independent of host VBUS (self-powered configuration). The EZ-PD™ PMG1-S3 MCU USB system has two signals to detect the presence of VBUS.

**VBUS power pad:**

The VBUS power pad pin in EZ-PD™ PMG1-S3 MCU powers the USB PHY and the USB I/Os (D+ and D– pins). The VBUS power pad is designated as P13[2] in the EZ-PD™ PMG1-S3 MCU pin mapping. Note that P13[2] (VBUS power pad) cannot be used as a general-purpose I/O – it can only be used as a power signal for the USB PHY. This pin can either be powered by the host VBUS or a power supply independent of VBUS. An internal comparator in the USB PHY indicates the presence/absence of VBUS as the P13[2] GPIO status signal. This is reflected as bit 2 of the GPIO_PRT13_PS port status register. Additionally, the GPIO_PRT13_INTR_CFG register can be configured to generate PICU interrupt on the rising/falling edge of the P13[2] pin. The GPIO_PRT13_INTR interrupt status register can be read in the “GPIO Interrupt - All Port” interrupt to know if any of the events on P13[2] triggered the interrupt.

**P0[0] GPIO pin:**

P0[0] GPIO pin should be used for VBUS detection instead of the VBUS power pad when any of the following conditions are true:

- When the VBUS power pad is supplied by an external power supply independent of host VBUS, P0[0] should be used for host VBUS detection
- When the VBUS from the host is not in the 4.35 V to 5.25 V range, P0[0] should be used for VBUS detection. P0[0] is powered by $V_{DDD}$ and is a GPIO. Refer to the EZ-PD™ PMG1-S3 MCU **device datasheet** for the GPIO specifications on input voltage threshold levels and maximum input voltage on the GPIO pin. When using P0[0] for VBUS detection, appropriate overvoltage protection circuitry such as a resistive divider should be placed while connecting the host VBUS to P0[0] pin.

Similar to the P13[2] (VBUS pad), the firmware can read the P0[0] pin status and configure the PICU interrupt based on the Port 0 registers.

### 21.3.1.3 USB PHY isolation logic

The USB PHY has an isolation logic to disable the PHY output signals when the host VBUS is not present.

**Figure 21-3** shows the isolation logic signal (iso_n) generation. iso_n is an active low signal. The device firmware can set the ISOLATE bit field in the USB_POWER_CTRL register to isolate the PHY outputs irrespective of the state of VBUS. For dynamic isolation based on the state of the VBUS, the device firmware should configure the VBUS_VALID_OVR[1:0] bit fields in the USB_POWER_CTRL register to select either the VBUS pad (P13[2]) or the P0[0] GPIO signal depending on which of the two I/Os are used for VBUS detection.

- The device firmware should enable the USB block operation only after the host VBUS is present – detected by reading the pin status register of P13[2] or P0[0].
- When VBUS is present, the device firmware should clear the ISOLATE bit field in the USB_POWER_CTRL register after a delay of at least 2 µs. Thereafter, the signal selected by VBUS_VALID_OVR[1:0] will take care of dynamic PHY output isolation. When the USB block is stopped in the device firmware, the ISOLATE bit field should be set as part of the stop sequence.

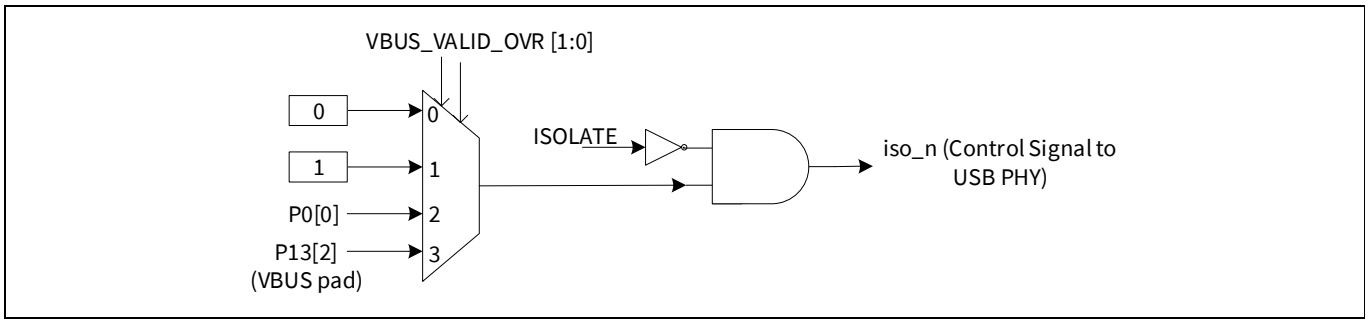



**Figure 21-3. USB PHY output isolation logic**

### 21.3.1.4 USB D+ pin pull-up enable logic

When a USB device is self-powered, the USB specification warrants that the device enable the pull-up resistor on its D+ pin to identify itself as a full-speed device to the host. When the host VBUS is removed, the device should disable the pull-up resistor on the D+ line so as to not back power the host. The USB PHY block includes an internal 1.5-kΩ pull-up resistor on the D+ line to indicate to the host that the EZ-PD™ PMG1-S3 MCU is a full-speed device. The pull-up resistor can be enabled or disabled by configuring the USBPUEN bit in the USBIO_CR1 register. Depending on whether the VBUS pad (P13[2]) or the P0[0] pin is used for VBUS detection, the firmware should read that pin status to enable/disable the pull-up resistor on the D+ pin.

### 21.3.1.5 Transmitter and receiver logic

The transmitter logic takes care of differential transmission to the USB host.

The transmitter can be manually forced to transmit signals. The USB_USBIO_CR0 register is used to manually transmit the signals. Examples are as follows:

- When the manual transmission is enabled, the register can be configured to transmit Single-Ended Zero signal (that is, D+ and D– are low).
- Configurable to transmit the USB signals. The USB signals can be two types:
  - D+ low and D– high = J
  - D+ high and D– low = K
- The register also has a bit, which is used to read the received signal levels. The bit can show if D+ < D– or D+ > D–.

### 21.3.1.6 Battery charger detection (BCD)

The USB PHY provides support to detect the type of host downstream port. The details of the detection algorithm and downstream port types are available in Revision 1.2 of the Battery Charging Specification, which can be downloaded from **www.usb.org/home**. The USB PHY can detect the following types of downstream ports – standard downstream port (SDP), charging downstream port (CDP), and dedicated charging port (DCP). The sequence of steps to detect the port type are as follows.

1. VBUS Detection: The device is required to detect VBUS assertion as the first step of the BCD algorithm. This should be implemented as an interrupt event to the CPU using rising/falling edge signal on the VBUS monitoring pin. The VBUS monitoring pin can either be the VBUS power pad or P0[0] pin as discussed in **"VBUS detection"** on page 184.
1. Configuration of the USB IP for BCD: The next step is to ensure the USB IP is properly configured to perform BCD. This includes the following:
   a) All the DMA channels associated with USB endpoints and the three USB interrupt vectors need to be disabled.
   b) The USB_ENABLE bit in the USBDEVv2_CR0 register should be cleared. The pull-up resistor on the D+ line should be disabled by clearing the USBPUEN bit in USBDEVv2_USBIO_CR1 register.
   c) The CSR_CLK_EN bit in USBDEVv2_USB_CLK_EN should be set to enable the clocking for the block. The D+ and D– pins should be configured for bit-banged mode by clearing the IOMODE bit in the USBDEVv2_USBIO_CR1 register.
   d) Configure the USBDEVv2_USB_POWER_CTRL register as follows: Disable the pull-down resistors on the VBUS power pad and D– pin. Configure the VBUS_VALID_OVR[1:0] bits to generate the vbus_valid signal from either the VBUS power pad (PHY detector) or the P0[0] GPIO pin depending on the pin used for VBUS monitoring. If VBUS monitoring is not used in the design, configure the VBUS_VALID_OVR[1:0] bits to force vbus_valid signal to always be 1. Set the SUSPEND, SUSPEND_DEL, ENABLE_RCVR, ENABLE_DPO, ENABLE_DMO, ENABLE_CHGDET, and ENABLE bits.
   e) Clear the SUSPEND bit in the USBDEVv2_USB_POWER_CTRL register; after a 2 µs delay, clear the SUSPEND_DEL bit.
2. Data line Contact Detect (DCD): The USB device is required to detect data line contact before performing charger detection in case there is a time lag between VBUS contact and data line contact. This time lag is accounted for by introducing a hardcoded delay of 400 ms before proceeding to the next step of the BCD algorithm.
3. Primary Detection: Primary detection is used to differentiate between SDP and DCP/CDP ports. To perform primary detection, set only the following bits in the USBDEVv2_USB_CHGDET_CTRL register: REF_EN, REF_DP, COMP_EN, and COMP_DM. The remaining register bits should be cleared. After a delay of 40 ms, the status of the D– line should be monitored by reading the DMO bit in the USBDEVv2_USBIO_CR1 register. The comparator output is indicated by the COMP_OUT bit in the USBDEVv2_USB_CHGDET_CTRL register. The following table specifies the port type for the different combinations of these two signals.

**Table 21-1. Primary detection port type for different combinations**

| Comparator output | D– State | Port type |
|---|---|---|
| 0 | 0 | SDP |
| 0 | 1 | Not a possible combination. Abort charger detection. |
| 1 | 0 | DCP or CDP. Need to perform secondary detection. |
| 1 | 1 | May be connected to a proprietary charger, where D+ and D– are pulled up. CPU should also read D+ status to confirm. |

4. Secondary Detection: Secondary detection is used to differentiate between CDP and DCP ports. To perform secondary detection, set only the following bits in the USBDEVv2_USB_CHGDET_CTRL register: REF_EN, REF_DM, COMP_EN, and COMP_DP. The remaining register bits should be cleared. After a delay of 40 ms, the status of the D+ line should be monitored by reading the DPO bit in the USBDEVv2_USBIO_CR1 register. The comparator output is indicated by the COMP_OUT bit in the USBDEVv2_USB_CHGDET_CTRL register. The following table specifies the port type for the different combinations of these two signals.

**Table 21-2. Secondary detection port type for different combinations**

| Comparator output | D+ State | Port type |
|---|---|---|
| 0 | 0 | CDP |
| 0 | 1 | Not a possible combination. Abort charger detection. |
| 1 | 0 | DCP |
| 1 | 1 | May be connected to a proprietary charger, where D+ and D– are pulled up. CPU should also read D– status to confirm. |

5. Restoring Register Values: After the port type detection, all the bits in the USBDEVv2_USB_CHGDET_CTRL register should be cleared. The ENABLE_CHGDET bit in the USBDEVv2_USB_POWER_CTRL register should be cleared. The USBDEVv2_USB_POWER_CTRL register should be restored to the value it had before the start of the BCD algorithm. The CSR_CLK_EN bit in USBDEVv2_USB_CLK_EN should be cleared. The IOMODE bit in the USBDEVv2_USBIO_CR1 register should be set. The USB device can undergo the enumeration process if the port type detected is SDP or CDP.

#### 21.3.1.7 Link power management (LPM)

The USB PHY supports Link Power Management (LPM), which is similar to the suspend mode, but has transitional latencies in tens of microseconds between power states compared to the greater than 20 ms associated with suspend/resume modes. More details on LPM can be found in the USB 2.0 specification (LPM ECN document and LPM errata document). The following features are supported for LPM.

- The USBDEVv2_USB_LPM_CTRL register should be configured to enable/disable LPM, type of response when LPM is enabled, and response when a sub PID other than the LPM token is received from the host.
- The USBDEVv2_USB_LPM_STAT register stores the values of the Best Effort Service Latency (BESL) and the remote wakeup feature as sent by the host. The firmware should read this register on the LPM interrupt event and enter the appropriate low-power mode (Deep-Sleep or Sleep) based on the BESL value from the host.

### 21.3.2 Endpoints

The SIE and arbiter support eight data endpoints (EP1 to EP8) and one control endpoint (EP0). The data endpoints share the SRAM memory area of 512 bytes. The endpoint memory management can be either manual or automatic. The endpoints are configured for direction and other configuration using the SIE and arbiter registers. The endpoint read address and write address registers are accessed through the arbiter.

The endpoints can be individually made active. In the Auto Management mode, the register USB_EP_ACTIVE is written to control the active state of the endpoint. The endpoint activation cannot be dynamically changed during runtime. In Manual Memory Management mode, the firmware decides the memory allocation, so it is not required to specify the active endpoints. The USB_EP_ACTIVE register is ignored during the manual memory management mode. The USB_EP_TYPE register is used to control the transfer direction (IN, OUT) for the endpoints. The control endpoint has separate eight bytes for its data (USB_EP0_DR registers).

### 21.3.3 Transfer types

The EZ-PD™ PMG1-S3 MCU USB supports full-speed transfers and is compliant with the USB 2.0 Specification. It supports four types of transfers:

- Interrupt Transfer
- Bulk Transfer
- Isochronous Transfer
- Control Transfer

For further details about these transfers, refer to the USB specification 2.0.

### 21.3.4 Interrupts

The USB block has three general-purpose interrupt lines – INTR_LO, INTR_MED, and INTR_HI. Refer to the **"Interrupts"** on page 38 for the vector numbers of these interrupt lines. The USB block has a predefined set of 13 interrupt trigger events that can be mapped to either one of the three interrupts. The interrupt line to which each of the trigger event is mapped is configured through the USB_INTR_LVL_SEL register. Each of these three interrupt lines has a status register to identify the event that cause of the interrupt. These are the USB_INTR_CAUSE_LO, USB_INTR_CAUSE_MED, and USB_INTR_CAUSE_HI status registers.

The following 12 events can be used to generate an interrupt on one of the three interrupt lines.

- USB Start of Frame (SOF) Event
- USB Bus Reset Event
- Eight Data Endpoint (EP1 – EP8) interrupt events
- Control Endpoint (EP0) interrupt event
- Arbiter Interrupt event
- Link Power Management (LPM) event

The following sections provide details on each of these interrupt events.

#### 21.3.4.1 Data endpoint interrupt events

These are eight interrupt events corresponding to each data endpoint (EP1-EP8). Each of the endpoint interrupt events can be enabled/disabled by using the corresponding bit in the USB_SIE_EP_INT_EN register. The interrupt status of each endpoint can be known by reading the USB_SIE_EP_INT_SR status register. An endpoint whose interrupt is enabled can trigger the interrupt on the following events:

- Successful completion of an IN/OUT transfer
- NAK-ed IN/OUT transaction if the corresponding NAK_INT_EN bit in the SIE_EPx_CR0 register is set
- When there is an error in the transaction, the bit ERR_IN_TXN in the SIE_EPx_CR0 register is set and interrupt is generated. Refer to the register description for details on the conditions under which this bit can be set.
- If the STALL bit in SIE_EPx_CR0 is set, then stall events can generate interrupts. This stall event can happen if an OUT packet is received for an endpoint whose mode bits in SIE_EPx_CR0 are set to ACK_OUT or if an IN packet is received with mode bits set to ACK_IN.

#### 21.3.4.2 Control endpoint (EP0) interrupt event

The interrupt event corresponding to the control endpoint (EP0) is generated under the following events:

- Successful completion of an IN/OUT transfer
- When a SETUP packet is received on the control endpoint

#### 21.3.4.3 Arbiter interrupt event

The arbiter generates an interrupt event for the endpoints during the following events – the final arbiter interrupt event is the logical OR of these events.

**DMA grant:**

This event is applicable in any of the DMA modes – Store and Forward DMA mode (Mode 2) or Cut Through mode (Mode 3).This event is triggered when the DMA controller pulses the "Burstend" signal corresponding to that endpoint, for which a DMA request had been raised to the DMA controller earlier. The request could have been either via a Manual DMA Request or an automatic Arbiter DMA Request. A common grant status exists for both modes of requests. This grant status indicates completion of the DMA transaction. This status indication can be used by firmware to determine when the next Manual DMA Request can be raised. Multiple requests raised for the same endpoint before the DMA Grant status is set will be dropped by the block. Only the first of multiple requests will be transmitted to DMA controller.

**IN endpoint local buffer full:**

This event status is set on different conditions in Store and Forward modes (Modes 1 and 2) and the Cut Through Mode (Mode 3).

- Store and Forward Mode: This status is set when the entire packet data has been transferred to the local memory. The check is that data written for the particular endpoint is equal to the programmed Byte Count for that endpoint in the USB.SIE_EPx.CNT0 and USB.SIE_EPx.CNT1 registers.
- Cut Through Mode: In this mode, the IN buffer full status is set when the IN endpoint's dedicated buffer is filled with the packet data. The size of this buffer is determined by the value programmed in bits [3:0] of the USB_BUF_SIZE register. This status indication can be used to determine when the Mode value in the USB_SIE_EPx_CR0 register can be programmed to acknowledge an IN token for that endpoint.

**Buffer overflow:**

This event is applicable only in the Cut Through Mode. This can occur due to buffer overflow either in the dedicated buffer space or the common buffer space. The buffer overflow conditions are explained here.

- In the case of an IN endpoint, the dedicated buffer can overflow if the DMA transfer writes a larger number of bytes than the space available in the dedicated buffer. Until an IN token has been received for that endpoint, it cannot use the common buffer area, hence resulting in an overflow of data. The possible causes of this buffer overflow can be incorrect programming of either the DMA transfer descriptor transfer size or the USB_BUF_SIZE register.
- In the case of an OUT endpoint, the dedicated buffer can overflow if two OUT transactions occur back to back. The data from the previous transaction is still present in the common area and the current ongoing transaction fills up the OUT endpoint's dedicated buffer space and overflows. The possible causes of this overflow can be the overall DMA bandwidth constraint due to other DMA transactions or reduced size of the dedicated OUT buffer size.
- In the case of an IN endpoint, the common buffer can overflow if the DMA transfer writes a larger number of bytes than the space available in the common area. The possible causes of this buffer overflow can be incorrect programming of either the DMA transfer descriptor transfer size or the USB_DMA_THRES and USB_DMA_THRES_MSB registers.
- In the case of an OUT endpoint, the common buffer area can overflow if the data written in to the common area has not yet been read and the new data begins to overwrite the existing data. The possible causes of this overflow can be the overall DMA bandwidth constraint due to other DMA transactions or the lower OUT endpoint DMA priority.

**Buffer underflow:**

This event is applicable only in the Cut Through Mode. This underflow condition can occur only for an IN endpoint. The underflow condition can occur either in the dedicated buffer space or common buffer space. The underflow condition on the dedicated buffer space can either be due to the reduced dedicated IN buffer size or DMA bandwidth constraint. The underflow condition can occur on the common buffer space due to DMA bandwidth constraint and/or lower DMA channel priority.

Setting up and Processing the Arbiter Interrupt Event: There is only one arbiter interrupt event common for all the endpoints – this is the logical OR of the above listed events for each of the endpoints. The master enable for arbiter interrupt event for each endpoint can be configured using the USB_ARB_INT_EN register. The status register ARB_INT_SR can be used to identify the endpoints that caused the arbiter interrupt event to trigger.

The register USB_ARB_EPx_INT_EN (where x = 1 to 8 for each endpoint) is used to enable or disable the arbiter interrupt events for each endpoint. The status of each of these sub events can be read using the USB_ARB_EPx_INT_SR register.

### 21.3.4.4 USB start of frame (SOF) event

- Generated whenever the SOF is received. The SOF interrupt is enabled by setting the SOF_INTR_MASK bit in the USB_INTR_SIE_MASK register. The SOF interrupt status is reflected in the SOF_INTR status bit in the USB_INTR_SIE status register.
- The SOF interrupt status is also available in the SOF_INTR_MASKED bit of the USB_INTR_SIE_MASKED register – this bit is the logical AND of corresponding SOF bits in the USB_INTR_SIE_MASK register and the USB_INTR_SIE register.
- The USB packet decoder in the SIE decodes the SOF PID and asserts a pulse of width one 12-MHz clock cycle. This pulse is synchronized to the system clock (SYSCLK) and can be routed as a digital signal through the Digital System Interconnect (DSI) interface. This routing feature is in addition to the interrupt feature.

### 21.3.4.5 USB bus reset event

- Generated whenever a USB bus reset condition occurs. The bus reset interrupt is enabled by setting the BUS_RESET_INTR_MASK bit in the USB_INTR_SIE_MASK register. The bus reset interrupt status is reflected in the BUS_RESET_INTR status bit in the USB_INTR_SIE status register.
- The bus reset interrupt status is also available in the BUS_RESET_INTR_MASKED bit of the USB_INTR_SIE_MASKED register – this bit is the logical AND of corresponding bus reset bits in the USB_INTR_SIE_MASK register and the USB_INTR_SIE register.
- The 32-kHz low-frequency clock (LFCLK) is used to detect the USB bus reset condition. It is required that the LFCLK is enabled in the application firmware. The SIE logic triggers on counter running on LFCLK when a SE0 condition is detected on the USB bus. When the counter reaches the count value configured in the USB_BUS_RST_CNT register, the bus reset interrupt is triggered. It is recommended to set the count value in the USB_BUS_RST_CNT register to three to detect the bus reset condition.

### 21.3.4.6 Link power management (LPM) event

Generated whenever the LPM token packet is received. The LPM interrupt is enabled by setting the LPM_INTR_MASK bit in the USB_INTR_SIE_MASK register. The LPM interrupt status is reflected in the LPM_INTR status bit in the USB_INTR_SIE status register.

The LPM interrupt status is also available in the LPM_INTR_MASKED bit of the USB_INTR_SIE_MASKED register; this bit is the logical AND of corresponding LPM bits in the USB_INTR_SIE_MASK and USB_INTR_SIE registers.

The firmware needs to read the USBDEVv2_USB_LPM_STAT register to read the BESL, remote wakeup values and appropriately enter the desired low-power mode. It is recommended to enter the low-power mode in the main code. The exit from LPM is identical to the resume event wakeup in the case of suspend mode.

### 21.3.5 DMA support

Each of the eight data endpoints has one DMA channel available to transfer data between the endpoint buffer and the SRAM memory. The USB IP generates the DMA request signals (usb.dma_req[7:0]) to the respective DMA channels to initiate the data transfer for the endpoint. This goes to the Trigger Group 2 multiplexer as input triggers that can be routed to one of the trigger inputs of DMA channels 8-15. Details on Trigger Groups can be referred to in the **"Trigger multiplexer block"** on page 105. The transfer complete signals from the endpoint DMA channels (cpuss.dmac_tr_out[8:15]) are routed to the USB IP as input signals usb.dma_burstend[0:7]. The eight output signals of these trigger multiplexers are usb.dma_burstend[0:7]. Refer to **"Trigger multiplexer block"** on page 105 for details on trigger sources and destinations.

## 21.4 Logical transfer modes

The USB block in EZ-PD™ PMG1-S3 MCU devices supports two types of logical transfers. The logical transfers can be configured using the register setting for each endpoint. Any of the logical transfer methods can be adapted to support the three types of data transfers (Interrupt, Bulk, and Isochronous) mentioned in the USB 2.0 Specification. The Control transfer is mandatory in any USB device.

The logical transfer mode is a combination of memory management and DMA configurations. The Logical Transfer modes are related to the data transfer within the USB block (to and from the SRAM memory unit for each endpoint). It does not represent the transfer methods between the device and the host (the transfer types specified in the USB 2.0 Specification).

The USB block supports two basic types of transfer modes, which are detailed in **Table 21-3**.

- Store and Forward mode
- Cut Through mode

**Table 21-3. USB transfer modes**

| Feature | Store and forward mode | Cut through mode |
|---|---|---|
| SRAM Memory Usage | Requires more memory | Requires less memory |
| SRAM Memory Management | Manual | Auto |
| SRAM Memory Sharing | 512 bytes of SRAM shared between endpoints. Sharing is done by firmware. | Each endpoint is allocated less share of memory automatically by the block. Rest of memory is available as "Common Area." This Common Area is used during the transfer. |
| IN Command | Entire packet present in SRAM memory before the IN command is received. | Memory filled with data only when SRAM IN command is received. Data is given to host when enough data is available (based on DMA configuration). Does not wait for the entire data to be filled. |
| OUT Command | Entire packet is written to SRAM memory on OUT command. After entire data is available, it is copied from SRAM memory to the USB device. | Waits only for enough bytes (depends on DMA configuration) to be written in SRAM memory. When enough bytes are present, it is immediately copied from SRAM memory to the USB device. |
| Transfer of Data | Data is transferred when all bytes are written to the memory. | Data is transferred when enough bytes are available. It does not wait for the entire data to be filled. |

**Table 21-3. USB transfer modes** (continued)

| Feature | Store and forward mode | Cut through mode |
|---|---|---|
| Types Based on DMA | No DMA mode<br>Manual DMA mode | Only Auto DMA mode |
| Supported Transfer Types | Best suited for Interrupt and Bulk transfers | Best suited for Isochronous transfer |

Every endpoint has a set of registers that need to be handled during the modes of operation, as detailed in **Table 21-4**.

**Table 21-4. Endpoint registers**

| Register | Comment | Content | Usage |
|---|---|---|---|
| ARB_RWx_WA | Endpoint Write Address register | Address of the SRAM | This register indicates the SRAM location to which the data in the Data register is to be written. |
| ARB_RWx_RA | Endpoint Read Address register | Address of the SRAM | This register indicates the SRAM location from which the data must be read and stored to the Data register. |
| ARB_RWx_DR | Endpoint Data Register | 8-Bit Data | Data register is read/ written to perform any transaction.<br>IN command: Data written to the Data register is copied to the SRAM location specified by the WA register. After write, the WA value is automatically incremented to point to the next memory location.<br>OUT command: Data available in the SRAM location pointed by the RA register is read and stored to the DR. When the DR is read, the value of RA is automatically incremented to point to the next SRAM memory location that must be read. |
| SIE_EPx_CNT0 and SIE_EPx_CNT1 | Endpoint Byte Count Register | Number of Bytes | Holds the number of bytes that can be transferred.<br>IN command: Holds the number of bytes to be transferred to host.<br>OUT command: Holds the maximum number of bytes that can be received. The firmware programs the maximum number of bytes that can be received for that endpoint. The SIE updates the register with the number of bytes received for the endpoint. |
| “Mode” bits in SIE_EPx_CR0 | Mode Values | Response to the Host | Controls how the USB device responds to the USB traffic and the USB host. Some examples of mode include ACK, NAK, STALL, etc. See **Table 21-3** for additional details. |

### USB Full Speed (USB FS)

In Manual Memory Management, the endpoint read and endpoint write address registers are updated by the firmware. So the memory allocation can be done as required by the user and the memory allocation decides which endpoints are active; that is, the user can decide to share the 512 bytes for all the eight endpoints or a lesser number of endpoints.

In Automatic memory management, the endpoint read and endpoint write address registers are updated by the USB block. The block assigns memory to the endpoints that are activated using the USB_EP_ACTIVE register. The size of memory allocated depends on the value in the USB_BUF_SIZE register. The rest of the memory, after allocation, is called the “Common Area” memory and used for the transfer of data.

In all of these modes, either the 8-bit endpoint data register or the 16-bit endpoint data register can be used to read/write to the endpoint buffer. While transferring data to the 16-bit data registers, it must be ensured that the corresponding SRAM memory address locations are also 16-bit aligned.

In the following text, the algorithm for the IN and OUT transaction for each mode is discussed. An IN transaction is when the data is read by the USB host (for example, PC). An OUT transaction is when the data is written by the USB host to the USB device. The choice of using the DMA and memory management can be configured using the USB_ARB_CFG register and the mode is common to all endpoints.

## 21.4.1 Store and forward mode

### 21.4.1.1 No DMA access

This is the Manual Memory Management mode with no DMA access.

IN Transaction (CPU Write, SIE Read): The steps for an IN transaction on an IN endpoint are shown in **Figure 21-4**. OUT Transaction (CPU Read, SIE Write): The steps for an OUT transaction on an OUT endpoint are shown in **Figure 21-5**.

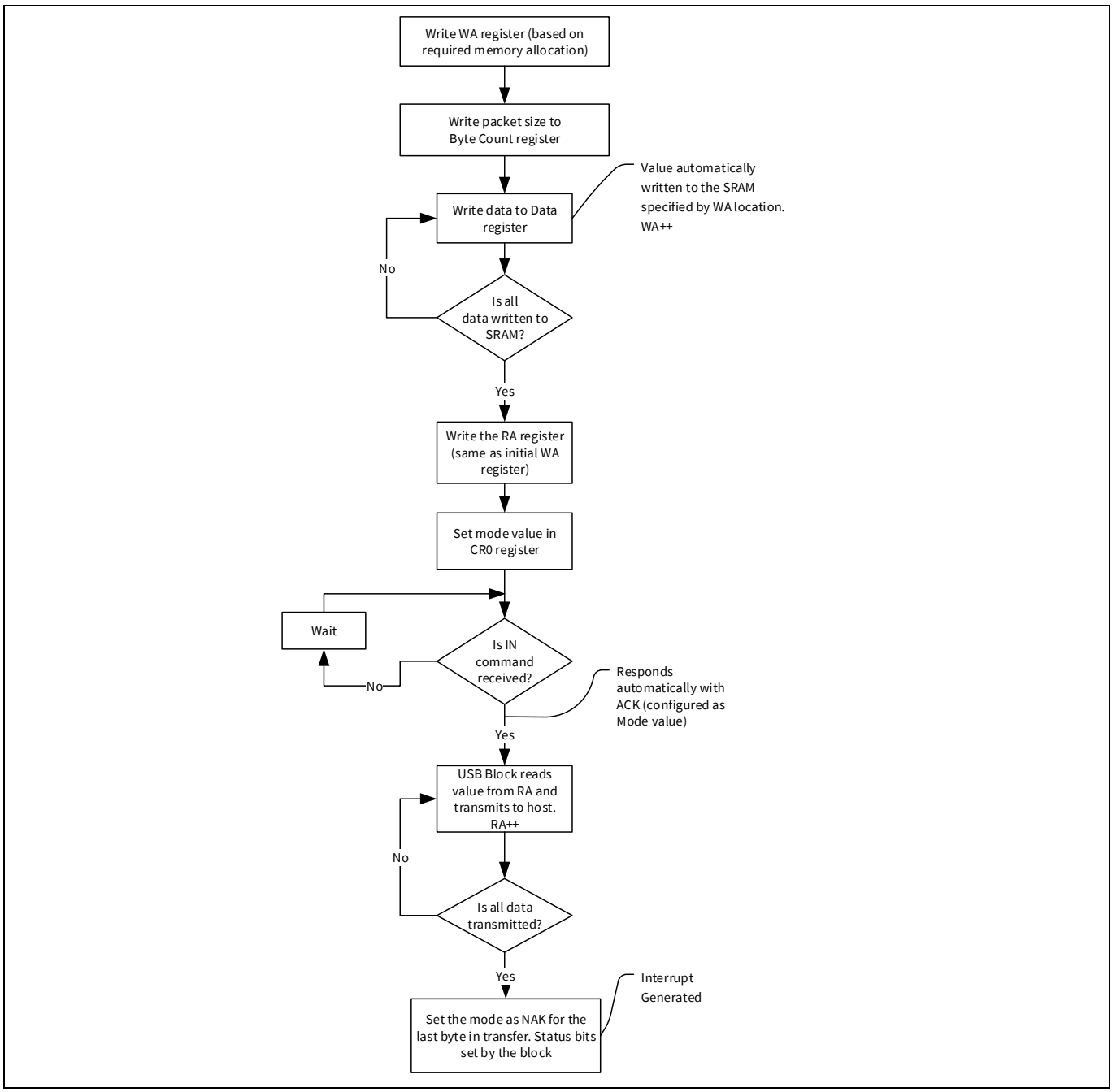



**Figure 21-4. No DMA access IN transaction**

## USB Full Speed (USB FS)

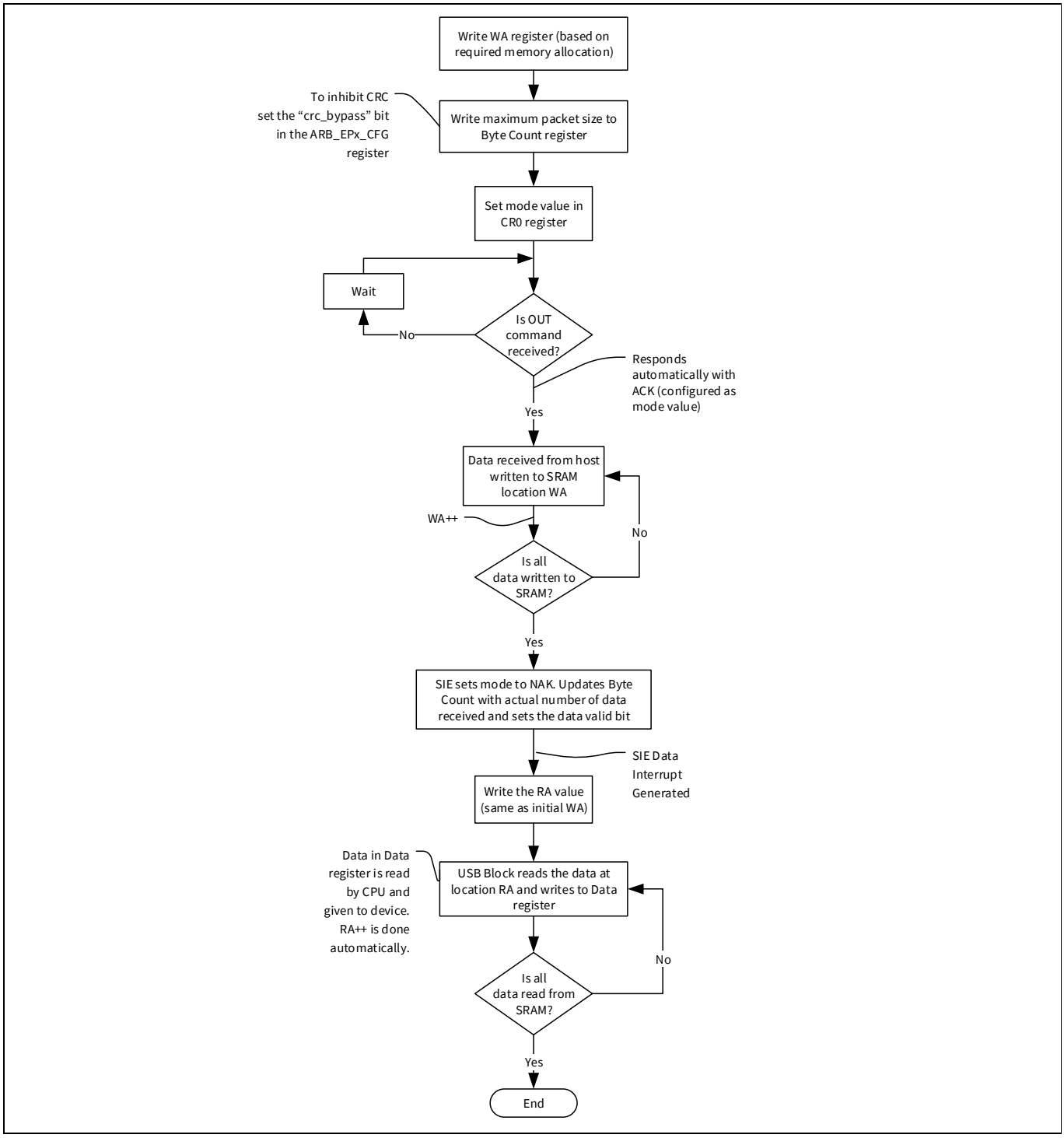



**Figure 21-5. No DMA access OUT transaction**

### 21.4.1.2 Manual DMA access

This is the Manual Memory Management mode with Manual DMA Access. This mode requires the configuration of the DMA controller. For information on DMA configuration, refer to the **"DMA controller modes"** on page 49. This mode is similar to the No DMA Access except that the write/read of packets is performed by DMA. A DMA request for an endpoint is generated by setting the DMA_CFG bit in the USB_ARB_EPx_CFG register. When the DMA service is granted and is done (DMA_GNT), an arbiter interrupt can be programmed to occur. The transfer is done using a single DMA cycle or multiple DMA cycles. After completion of every DMA cycle the arbiter interrupt (DMA_GNT) is generated. Similarly, when all the bytes of data (programmed in the byte count) are written to the memory, the arbiter interrupt occurs and the IN_BUF_FULL bit is set.

IN Transaction (CPU Write, SIE Read). The steps for an IN transaction on an IN endpoint are shown in **Figure 21-6**.

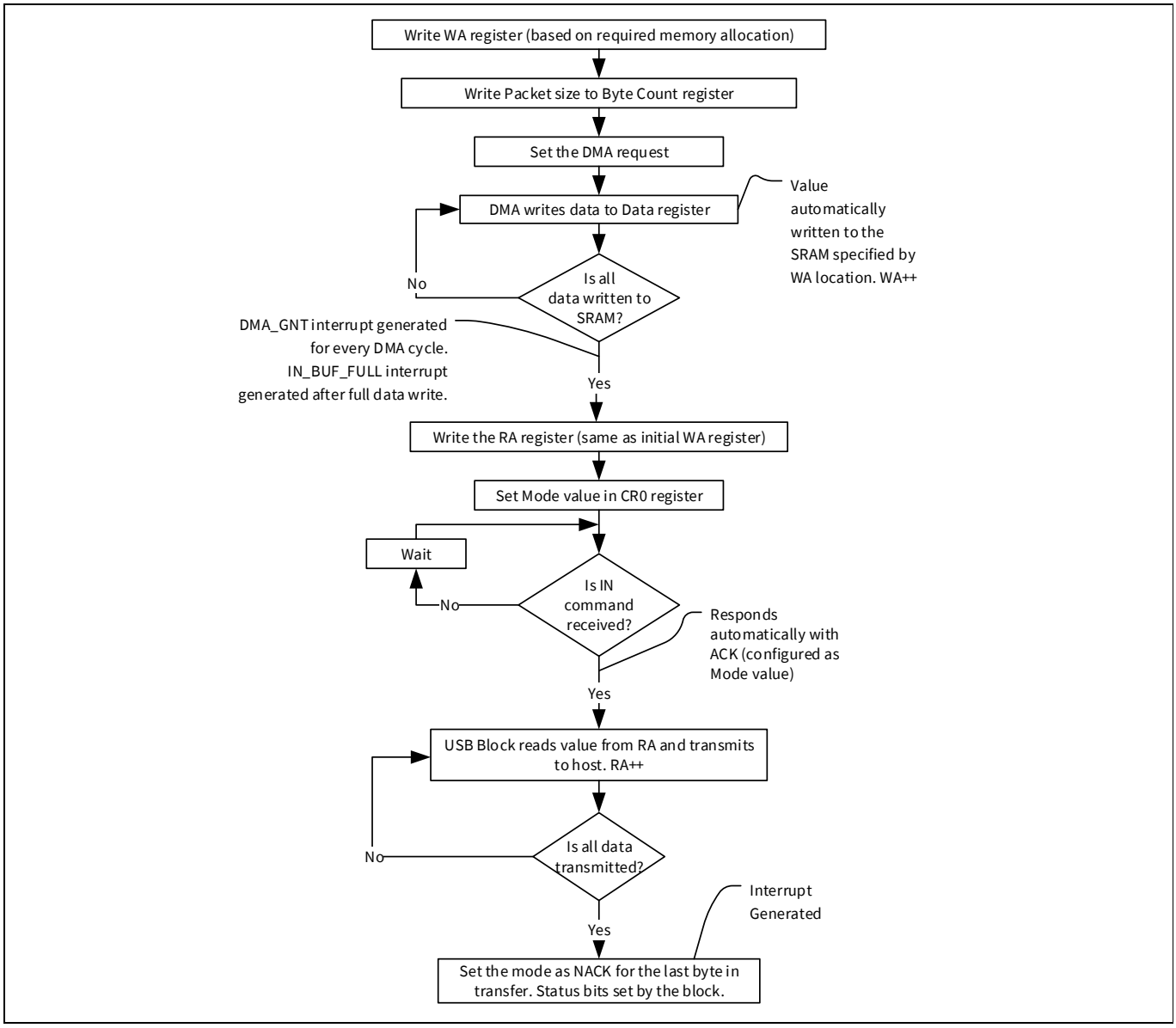



**Figure 21-6. Manual DMA IN transaction**

OUT Transaction (CPU Read, SIE Write): The steps for an OUT transaction on an OUT endpoint are shown in **Figure 21-7**.

## USB Full Speed (USB FS)

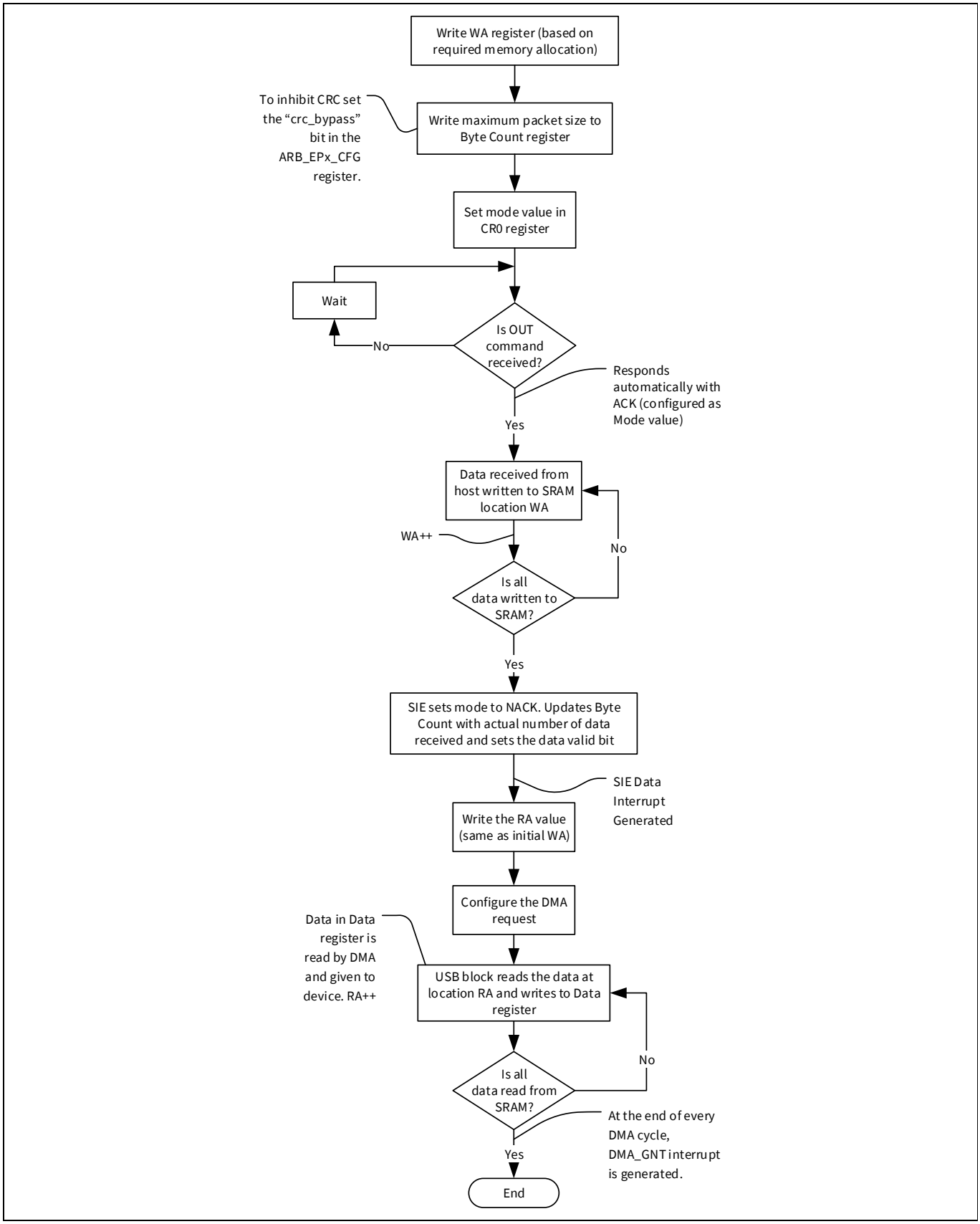



**Figure 21-7. Manual DMA OUT transaction**

### 21.4.2 Cut through mode

This is the Auto Memory Management mode with Auto DMA Access. The CPU programs the initial buffer size requirement for IN/OUT packets and informs the Arbiter block of the endpoint configuration details for the particular application being considered. The block then controls memory partitioning and handling of all memory pointers. During memory allocation, each active IN endpoint (set by the USB_EP_ACTIVE and USB_EP_TYPE registers) is allocated a small amount of memory configured using the USB_BUF_SIZE register (32 bytes for each of the eight endpoints). The remaining memory (256 bytes) is left as "Common Area" and is common for all endpoints.

In this mode, the memory requirement is less and it is suitable for the full-speed isochronous transfer up to 1023 bytes.

When an IN command is sent by the host, the device responds with the data present in the dedicated memory area for that endpoint. It simultaneously issues a DMA request for more data for that EP. This data fills up in the common area. The device does not wait for the entire packet of data to be available. It only waits for the (USB_DMA_THRES_MSB, USB_DMA_THRES) number of data available in the SRAM memory and begins the transfer from the common area.

Similarly, when an OUT command is received, the data for the OUT endpoint is written to the common area. When some data (greater than USB_DMA_THRES_MSB, USB_DMA_THRES) is available in the common area, the Arbiter block initiates a DMA request and the data is immediately written to the device. The device does not wait for the common area to be filled.

This mode requires the configuration of the USB_DMA_THRES and USB_DMA_THRES_MSB registers to hold the number of bytes that can be transferred in one DMA transfer (32 bytes). Similarly, the burst count of the DMA should always be equal to the value set in the USB_DMA_THRES registers. Apart from the DMA configuration, this mode also needs the configuration of the USB_BUF_SIZE for the IN and the OUT buffers and the USB_EP_ACTIVE and the USB_EP_TYPE registers.

Each DMA channel has two descriptors and both of them are used in this mode. Each descriptor is considered as a data chunk of 32 bytes and it executes according to the trigger mechanism. The descriptors are chained and hence 64 bytes can be transferred without firmware interaction. When both descriptors complete, the endpoint DMA done interrupt and the DMA error interrupt triggers (due to the lack of data to transfer). The descriptors are updated to advance the source SRAM (IN endpoint) or destination SRAM (OUT endpoint) pointer locations and then enabled again. The firmware also triggers the DMA transfer in software. This sequence continues till all data is transferred. See the **“DMA controller modes”** on page 49 for details on DMA configuration.

IN Transaction (CPU Write, SIE Read): The steps for an IN transaction on an IN endpoint are shown in **Figure 21-8**.

## USB Full Speed (USB FS)

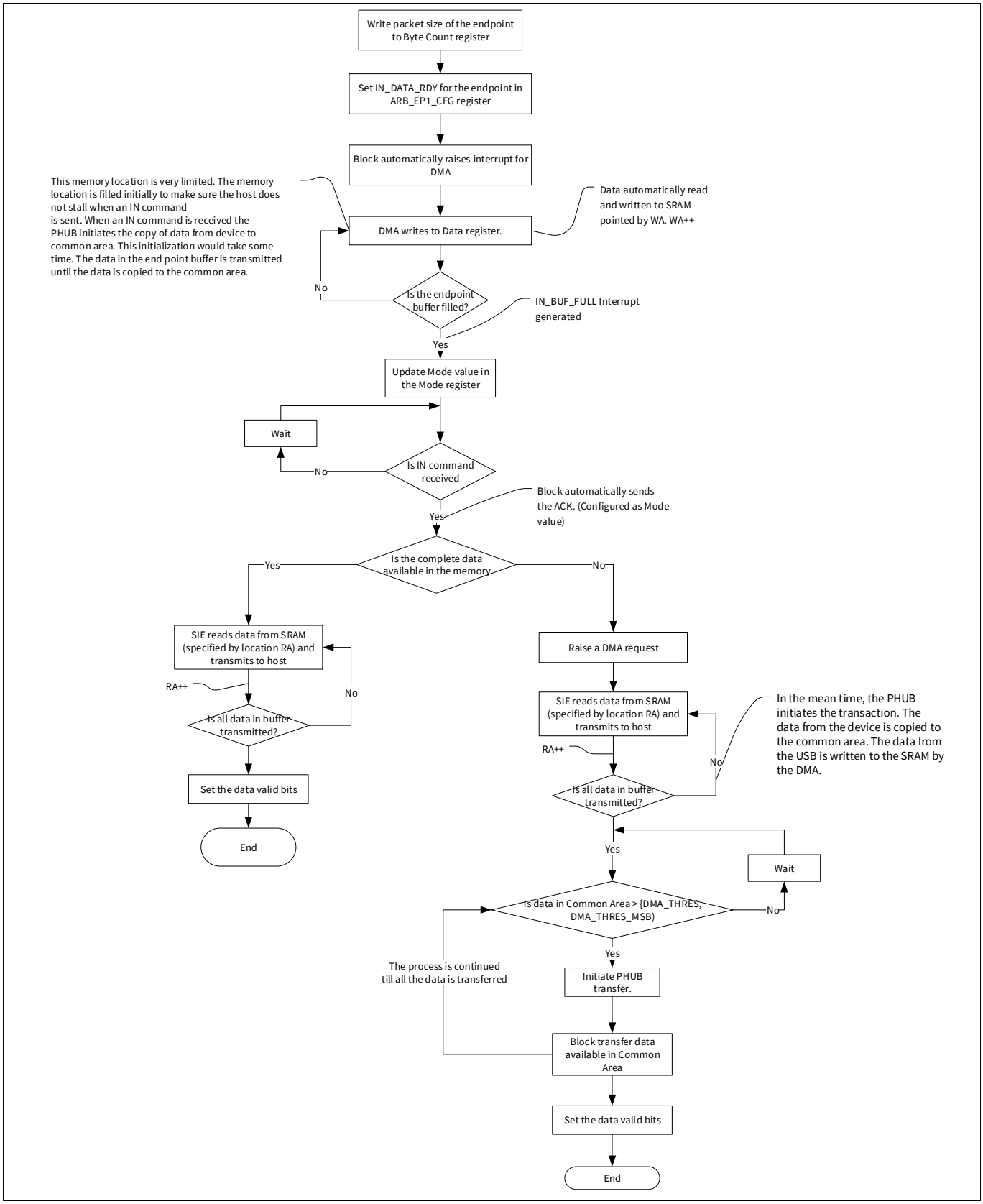



**Figure 21-8. Cut through mode IN transaction**

OUT Transaction (CPU Read, SIE Write). The steps for an OUT transaction on an OUT endpoint are shown in **Figure 21-9**.

## USB Full Speed (USB FS)

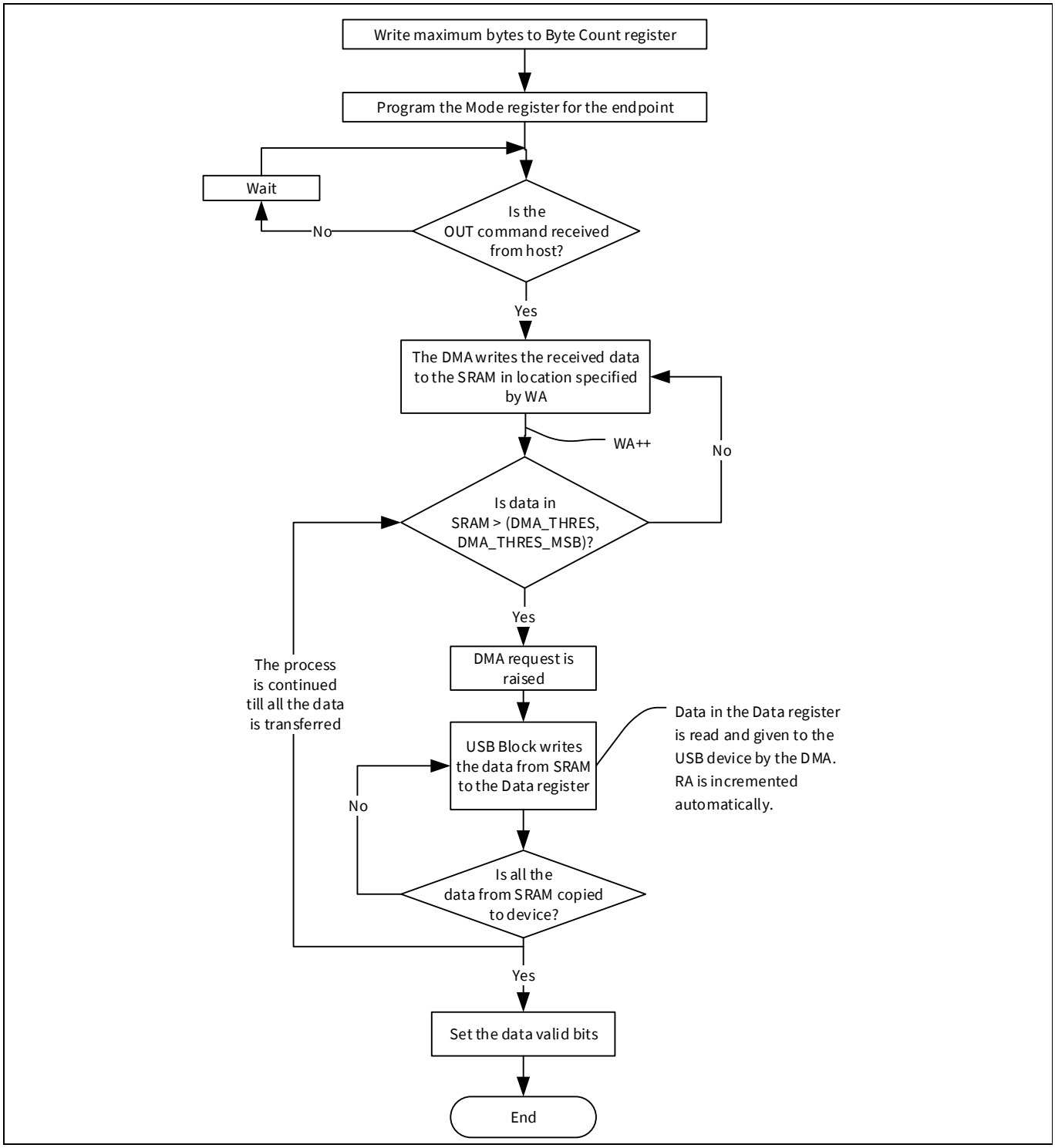



**Figure 21-9. Cut through mode OUT transaction**

### 21.4.3 Control endpoint logical transfer

The control endpoint has a special logical transfer mode. It does not share the 512 bytes of memory. Instead, it has a dedicated 8-byte register buffer (USB_EP0_DRx registers). The IN and OUT transaction for the control endpoint is detailed in the following figures.

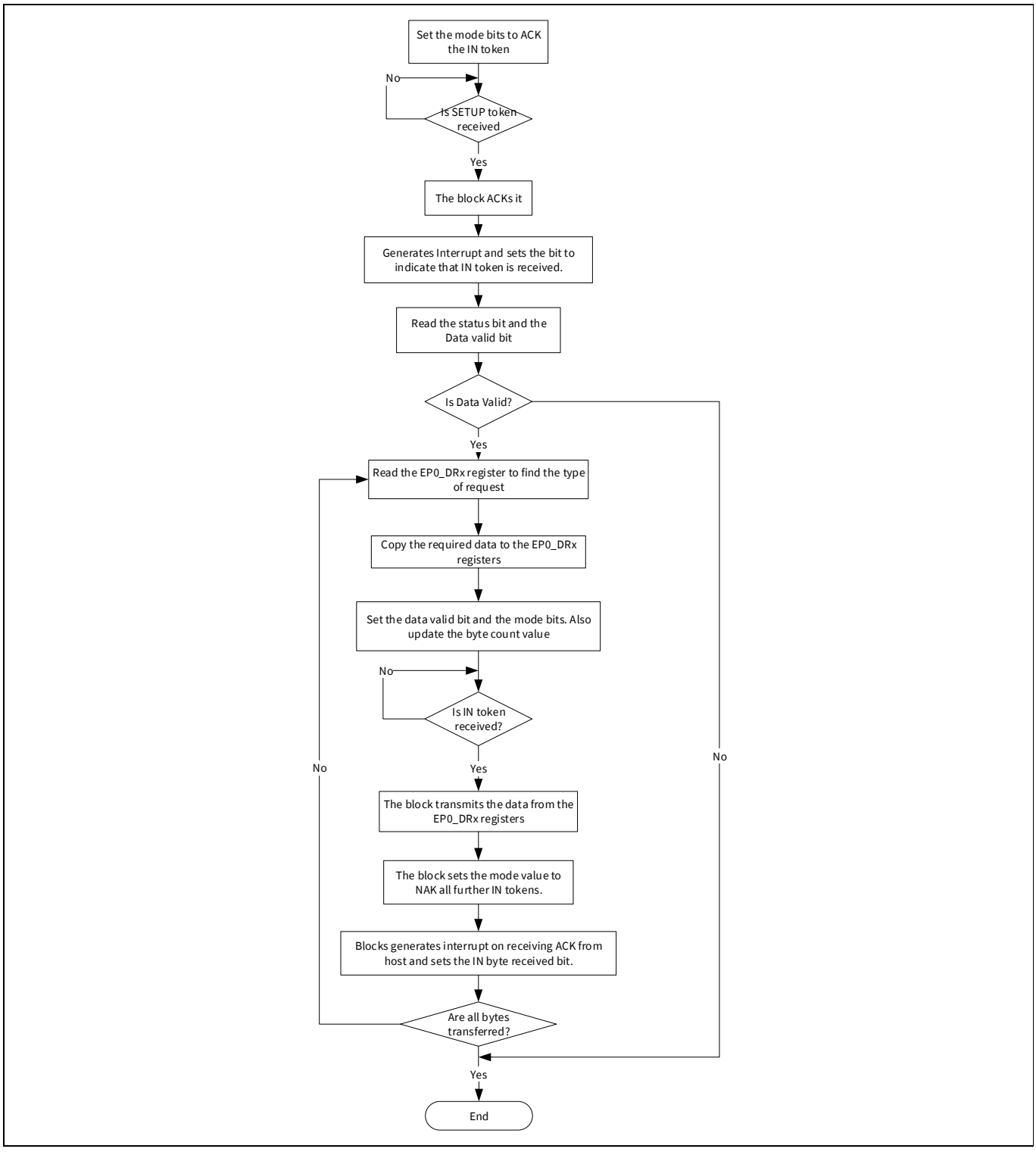



**Figure 21-10. Control endpoint IN transaction**

## USB Full Speed (USB FS)

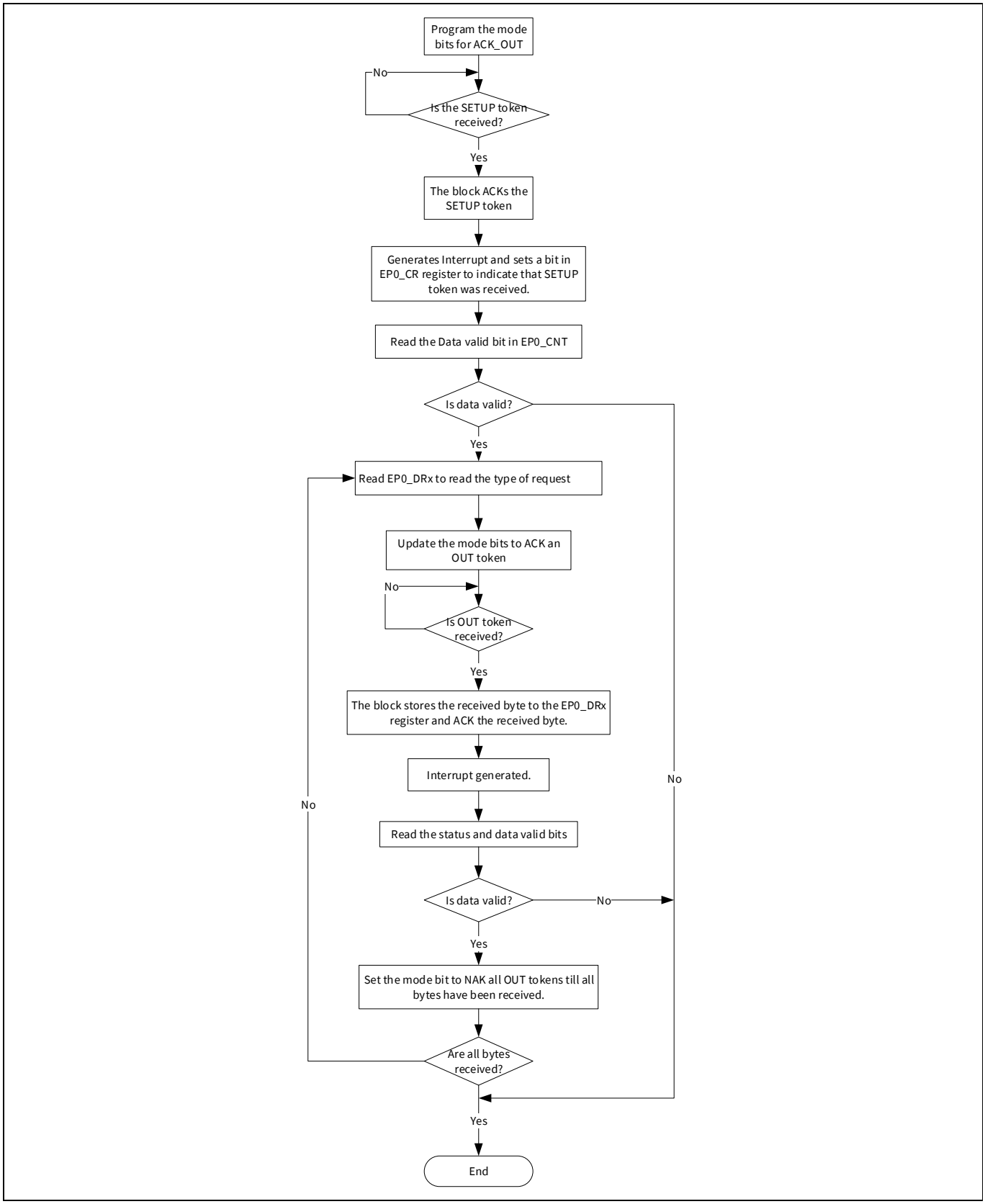



**Figure 21-11. Control endpoint OUT transaction**

## 21.5 Registers

**Table 21-5. USB FS registers**

| Register name | Description |
|---|---|
| USBDEVv2_EP0_DRx | Control Endpoint EP0 Data Register. 'x' can be 0–7 for the 8-byte EP0 data buffer |
| USBDEVv2_CR0 | USB Control 0 Register. Register to enable the USB device and specify the 7-bit device address |
| USBDEVv2_CR1 | USB Control 1 Register. Register to configure the USB block regulator, read bus activity status, and enable the internal oscillator to the USB traffic |
| USBDEVv2_SIE_EP_INT_EN | USB SIE Data Endpoints Interrupt Enable Register |
| USBDEVv2_SIE_EP_INT_SR | USB SIE Data Endpoint Interrupt Status |
| USBDEVv2_SIE_EPx_CNT0 | Non-control Endpoint Byte Count Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register stores the most significant 3-bits of the 11-bit byte counter, the toggle state of the data packet, and data valid status. |
| USBDEVv2_SIE_EPx_CNT1 | Non-control Endpoint Byte Count Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register stores the lower eight bits of the 11-bit byte count value. |
| USBDEVv2_SIE_EPx_CR0 | Non-control Endpoint Control Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register contains the endpoint operating mode, error status, stall control, and the NAK interrupt generation. |
| USBDEVv2_USBIO_CR0 | USBIO Control 0 Register. This register contains the control and configuration bits for the USB I/Os (D+ and D– pins respectively) for single-ended and differential mode operation. |
| USBDEVv2_USBIO_CR1 | USBIO Control 1 Register. This register contains the control and configuration bits for the USB I/Os (D+ and D– pins respectively) for selecting the USB operating mode, reading the single ended USBIO receiver outputs, and enabling pull-up resistor on the D+ line. |
| USBDEVv2_DYN_RECONFIG | USB Dynamic Reconfiguration Register. This register is used to control the status of dynamic reconfiguration of an endpoint. |
| USBDEVv2_SOF0 | Start Of Frame Register. This register contains the lower eight bits [7:0] of the SOF frame number. |
| USBDEVv2_SOF1 | Start Of Frame Register. This register contains the upper three bits [10:8] of the SOF frame number. |
| USBDEVv2_OSCLK_DR0 | Oscillator Lock Data Register 0. This register contains the lower eight bits of the oscillator locking circuit's adder output. |
| USBDEVv2_OSCLK_DR1 | Oscillator Lock Data Register 1. This register contains the upper seven bits of the oscillator locking circuit's adder output. |
| USBDEVv2_EP0_CR | Endpoint0 Control Register. This register contains operating mode of the control endpoint, and the status bits for different packet conditions on the control endpoint. |
| USBDEVv2_EP0_CNT | Endpoint0 Count Register. This register stores the 4-bit byte counter, the toggle state of the data packet, and data valid status. |

**Table 21-5. USB FS registers** (continued)

| Register name | Description |
|---|---|
| USBDEVv2_ARB_EPx_CFG | Endpoint Configuration Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. This register contains the settings to reset the endpoint buffer pointers, CRC bypass feature, manual DMA request, and the data ready status. |
| USBDEVv2_ARB_EPx_INT_EN | Endpoint Interrupt Enable Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. Register to configure the conditions under which an interrupt should be generated for an endpoint. |
| USBDEVv2_ARB_EPx_SR | Endpoint Interrupt Status Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. Register to read the interrupt status of an endpoint. |
| USBDEVv2_ARB_RWx_WA | Endpoint Write Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Lower eight bits of the 9-bit write address pointer. |
| USBDEVv2_ARB_RWx_WA_MSB | Endpoint Write Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Most significant bit of the 9-bit write address pointer. |
| USBDEVv2_ARB_RWx_RA | Endpoint Read Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Lower eight bits of the 9-bit read address pointer. |
| USBDEVv2_ARB_RWx_RA_MSB | Endpoint Read Address value. 'x' can be 1–8 corresponding to one of the eight data endpoints. Most significant bit of the 9-bit read address pointer. |
| USBDEVv2_ARB_RWx_DR | Endpoint Data Register. 'x' can be 1–8 corresponding to one of the eight data endpoints. |
| USBDEVv2_BUF_SIZE | Dedicated Endpoint Buffer Size Register |
| USBDEVv2_EP_ACTIVE | Endpoint Active Indication Register |
| USBDEVv2_EP_TYPE | Endpoint Type (IN/OUT) Indication register |
| USBDEVv2_ARB_CFG | Arbiter Configuration Register. This register is used to configure the buffer management mode of the USB block. |
| USBDEVv2_USB_CLK_EN | USB Block Clock Enable Register |
| USBDEVv2_ARB_INT_EN | Arbiter Interrupt Enable Register. This register contains the configuration to enable arbiter interrupt generation for each endpoint. |
| USBDEVv2_ARB_INT_SR | Arbiter Interrupt Status Register. This register contains the status of arbiter interrupt generation due to each endpoint. |
| USBDEVv2_CWA | Common Area Write Address Register. This register contains the lower eight bits of the 9-bit write address pointer for the common area. |
| USBDEVv2_CWA_MSB | Common Area Write Address Register. This register contains the most significant bit of the 9-bit write address pointer for the common area. |
| USBDEVv2_DMA_THRES | DMA Burst/Threshold Configuration Register. This register contains the lower eight bits of the 9-bit threshold byte count value. |
| USBDEVv2_DMA_THRES_MSB | DMA Burst/Threshold Configuration Register. This register contains the most significant bit of the 9-bit threshold byte count value. |
| USBDEVv2_BUS_RST_CNT | Bus Reset Count Register. This register contains the number of clock cycles of LFCLK that should elapse to detect a bus reset condition. |

**Table 21-5. USB FS registers** (continued)

| Register name | Description |
|---|---|
| USBDEVv2_MEM_DATAx | This is the 512-byte data buffer for storing the non-control endpoint data. 'x' can be 0 to 511. |
| USBDEVv2_SOF16 | 16-bit version of the Start Of Frame Register |
| USBDEVv2_OSCLK_DR16 | 16-bit version of the Oscillator Lock Data Register |
| USBDEVv2_ARB_RWx_WA16 | 16-bit version of the Endpoint Write Address Value Register. 'x' can be 1 to 8. |
| USBDEVv2_ARB_RWx_RA16 | 16-bit version of the Endpoint Read Address Value Register. 'x' can be 1 to 8. |
| USBDEVv2_CWA16 | 16-bit version of the Common Area Write Address Register |
| USBDEVv2_ARB_RWx_DR16 | 16-bit version of the Endpoint Data Register. 'x' can be 1 to 8 |
| USBDEVv2_DMA_THRES16 | 16-bit version of the DMA Burst/Threshold Configuration Register |
| USBDEVv2_USB_POWER_CTRL | Power Control Register |
| USBDEVv2_USB_CHGDET_CTRL | Charger Detection Control Register |
| USBDEVv2_USB_USBIO_CTRL | USB I/O Control Register |
| USBDEVv2_USB_FLOW_CTRL | Flow Control Register |
| USBDEVv2_USB_LPM_CTRL | LPM Control Register |
| USBDEVv2_USB_LPM_STAT | LPM Status Register |
| USBDEVv2_USB_INTR_SIE | USB SOF, BUS RESET, and EP0 Interrupt Status Register |
| USBDEVv2_USB_INTR_SIE_SET | USB SOF, BUS RESET, and EP0 Interrupt Set Register |
| USBDEVv2_USB_INTR_SIE_MASK | USB SOF, BUS RESET, and EP0 Interrupt Mask Register |
| USBDEVv2_USB_INTR_SIE_MASKED | USB SOF, BUS RESET, and EP0 Interrupt Masked Register |
| USBDEVv2_USB_INTR_LVL_SEL | USB Interrupt Level Select Register |
| USBDEVv2_USB_INTR_CAUSE_HI | High Priority Interrupt Cause Register |
| USBDEVv2_USB_INTR_CAUSE_MED | Medium Priority Interrupt Cause Register |
| USBDEVv2_USB_INTR_CAUSE_LO | Low Priority Interrupt Cause Register |
| USBDEVv2_USB_PHY_TRIM0 | PHY Trim Control Register |
| USBDEVv2_USB_PHY_TRIM1 | PHY Trim Control Register |
| USBDEVv2_USB_PHY_TRIM2 | PHY Trim Control Register |
| USBDEVv2_USB_PHY_TRIM3 | PHY Trim Control Register |
| USBDEVv2_USB_CHGDET_TRIM | Charger Detect Trim Values Register |
| USBDEVv2_USB_TRIM | Trim Values Register |
| USBDEVv2_USB_USBIO_TRIM | Trim Values for I/Os Register |

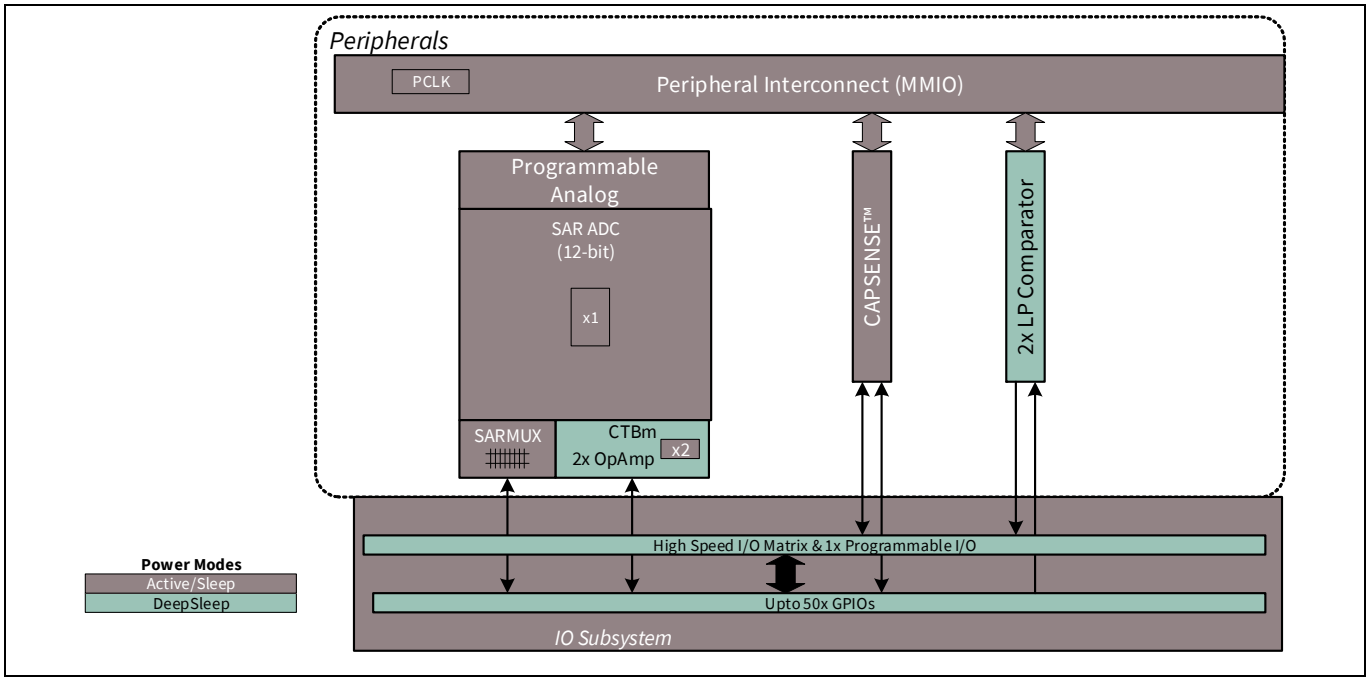
# Section G: Analog system

This section encompasses the following chapters:


- **"12-bit SAR ADC"** on page 207
- **"Continuous Time Block mini (CTBm)"** on page 237
- **"Low-power comparator"** on page 246
- **"Temperature sensor"** on page 252
- **"CAPSENSE™"** on page 255


## Top level architecture

### Analog system diagram



# 22 12-bit SAR ADC

The EZ-PD™ PMG1-S3 MCU has one 12-bit successive approximation register analog-to-digital converter (SAR ADC). The SAR ADC is designed for applications that require moderate resolution and high data rate. It consists of the following blocks (see **Figure 22-2**):

- SARMUX
- SAR ADC core
- SARREF
- SARSEQ

The SAR ADC core is a fast 12-bit ADC with sampling rate of 1 Msps. Preceding the SAR ADC is the SARMUX, which can route external pins and internal signals (AMUXBUS-A/-B, CTBm, temperature sensor output) to the 16 internal channels of SAR ADC. SARREF is used for multiple reference selection. The sequencer controller SARSEQ is used to control SARMUX and SAR ADC to do an automatic scan on all enabled channels without CPU intervention and for pre-processing, such as averaging the output data.

The result from each channel is double-buffered and a complete scan may be configured to generate an interrupt at the end of the scan. The sequencer may also be configured to flag overflow, collision, and saturation errors that can be configured to assert an interrupt.

For more flexibility, it is also possible to control most analog switches, including those in the SARMUX with the firmware. This makes it possible to implement an alternative sequencer with the firmware.

## 22.1 Features

- Operates across the entire device power supply range
- Maximum 1 Msps sample rate
- Sixteen individually configurable channels and one injection channel
  - Each channel has the following features:
  - Input from external pin (only for eight channels in single-ended mode and four channels in differential mode) or
  - internal signal (AMUXBUS/CTBm/temperature sensor)
  - Programmable acquisition times
  - Selectable 8-, 10-, and 12-bit resolution
  - Single-ended or differential measurement
  - Averaging
  - Results are double-buffered
  - Result may be left or right aligned
- Scan triggered by firmware, timer, CTBm comparator, low-power comparator, and by SAR end of conversion signal
  - Hardware/firmware trigger (one shot), and free-running (continuous conversion) modes
- Hardware averaging support
  - First order accumulate
  - Samples averaging from 2 to 256 (powers of 2)
- Results represented in 16-bit sign extended values
- Selectable voltage references
  - Internal VDDA and VDDA /2 references
  - Internal 1.2-V reference with buffer
  - External reference
- Interrupt generation
  - Finished scan conversion
  - Saturation detect and over-range (configurable) detect for every channel
  - Scan results overflow

  - Collision detect
- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences (tailgating)
  - Selectable sample time, resolution, single-ended or differential, averaging
- Low-power modes
  - ADC core and reference voltage have dedicated low power modes

## 22.2 Analog Multiplexed Bus (AMUX BUS)

EZ-PD™ PMG1-S3 MCU has two concentric independent buses that go around the periphery of the chip. These buses (called amux buses) are connected to firmware-programmable analog switches that allow the chip's internal resources (SAR ADCs, comparator, CSD, and Op-Amps) to connect to any pin on the I/O ports using HSIOM_PORT_SELx register.

The two amux can also be split in 3 to isolate CSD, ADC and GPIO connectivity using register HSIOM_AMUX_SPLIT_CTLx This helps in reducing cross-talk between CSD, ADCs and other peripherals. **Figure 22-1** shows the amux bus routing and splitting.

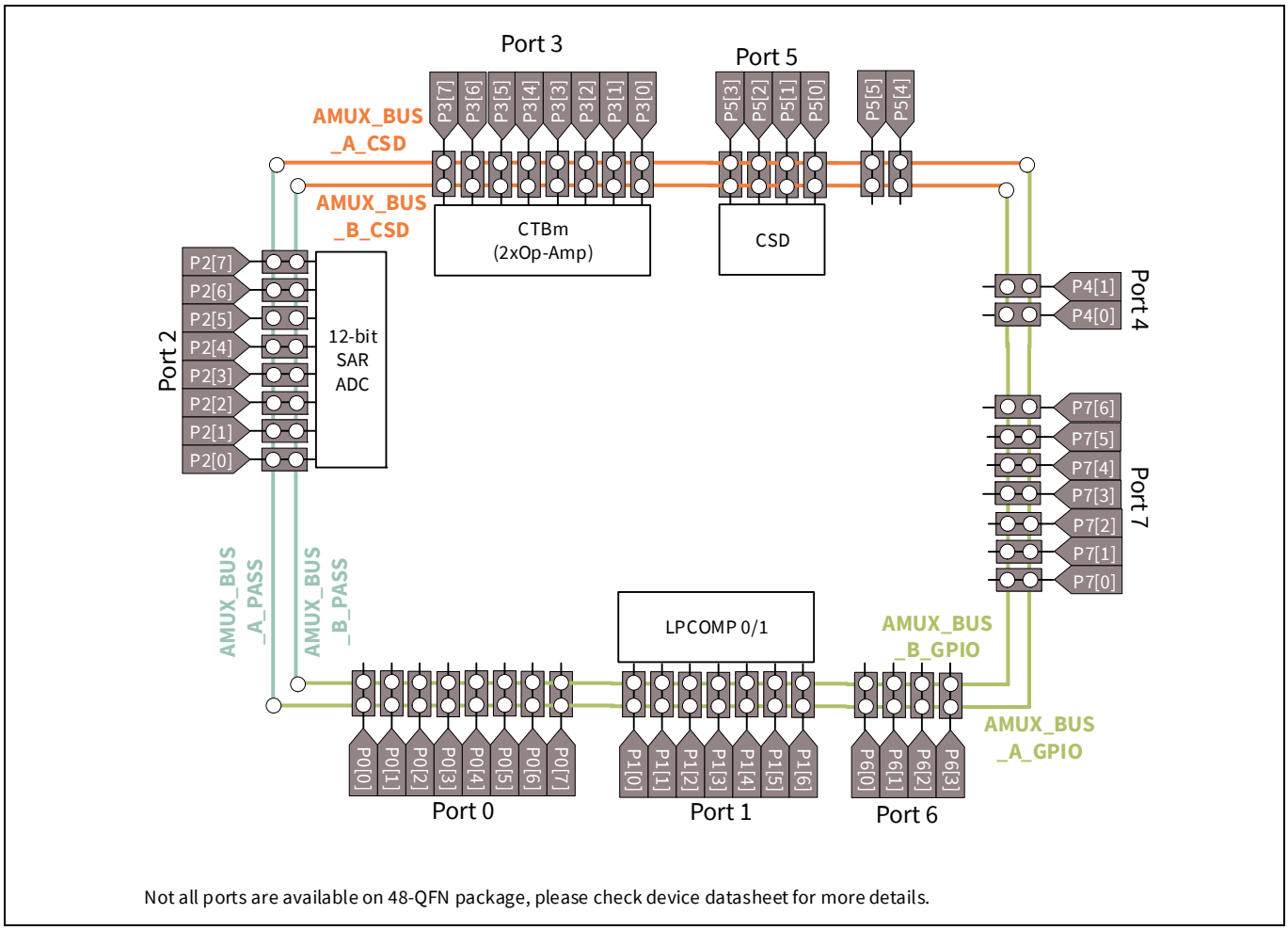



**Figure 22-1. AMUX BUS A and B routing**

## 22.3 Block diagram

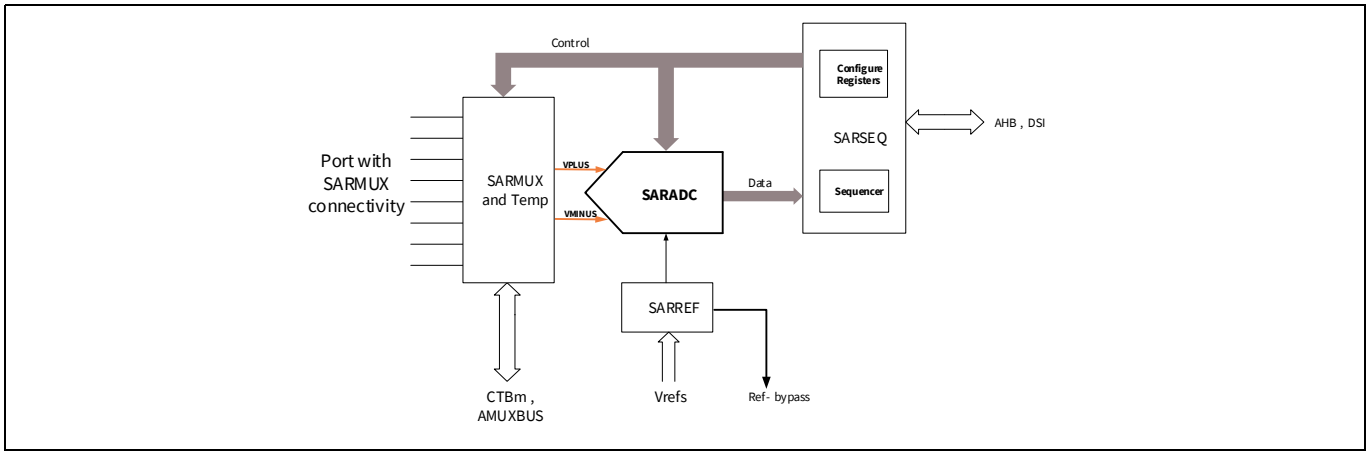



**Figure 22-2. Block diagram**

## 22.4 How it works

This section includes the following contents:

- Introduction of each block: SAR ADC core, SARMUX, SARREF, and SARSEQ
- SAR ADC system resource: Interrupt, low-power mode, and SAR ADC status
  - System operation
- Configuration examples

### 22.4.1 SAR ADC core

EZ-PD™ PMG1-S3 MCU SAR ADC core is a 12-bit SAR ADC. The maximum sample rate for this ADC is 1 Msps. The SAR ADC core has the following features:

- Fully differential architecture; also supports single-ended mode
- 12-bit resolution and a selectable alternate resolution:
- either 8-bit or 10-bit
- Programmable acquisition time
- Programmable power mode (full, one-half, one-quarter)
- Supports single and continuous conversion mode

#### 22.4.1.1 Single-ended and differential mode

EZ-PD™ PMG1-S3 MCU SAR ADC can operate in single-ended and differential mode. It is designed in a fully differential architecture, optimized to provide 12-bit accuracy in the differential mode of operation. It gives full range output (0 to 4095) for differential inputs in the range of –VREF to +VREF. SAR ADC can be configured in single-ended mode by fixing the negative input. Differential or single-ended mode can be configured by channel configuration register, SAR_CHANx_CONFIG.

The single-ended mode options of negative input include: VSSA, VREF, or an external input from any of the eight pins with SARMUX connectivity. See the EZ-PD™ PMG1-S3 MCU datasheet for the pin details. This mode is configured by the global configuration register SAR_CTRL. When Vminus is connected to these SARMUX pins, the single-ended mode is equivalent to differential mode. However, when the odd pin of each differential pair is connected to the common alternate ground, these conversions are 11-bit, because measured signal value (SARMUX.vplus) cannot go below ground.

To get a single-ended conversion with 12 bits, it is necessary to connect VREF to the negative input of the SAR ADC; then, the input range can be from 0 to 2 × VREF.

Note that temperature sensor can only be used in single-ended mode; it will override the SAR_CTRL [11:9] to 0. The differential conversion is not available for temperature sensors; the result is undefined.

### 22.4.1.2 Input range

All inputs should be in the range of VSSA to VDDA. Input voltage range is also limited by VREF. If voltage on negative input is Vn and the ADC reference is VREF, the range on the positive input is Vn ± VREF. This criteria applies for both single-ended and differential modes. In single-ended mode, Vn is connected to VSSA, VREF or an external input.

Note that Vn ± VREF should be in the range of VSSA to VDDA. For example, if negative input is connected to VSSA, the range on the positive input is 0 to VREF, not –VREF to VREF. This is because the signal cannot go below VSSA. Only half of the ADC range is usable because the positive input signal cannot swing below VSS, which effectively only generates an 11-bit result.

### 22.4.1.3 Result data format

Result data format is configurable from two aspects:

- Signed/unsigned
- Left/right alignment

When the result is considered signed, the most significant bit of the conversion is used for sign extension to 16 bits with MSB. For an unsigned conversion, the result is zero extended to 16-bits. It can be configured by SAR_SAMPLE_CTRL [3:2] for differential and single-ended conversion, respectively.

The sample value can either be right-aligned or left-aligned within the 16 bits of the result register. By default, data is right-aligned in data[11:0], with sign extension to 16 bits, if required. A lower resolution combined with left-alignment will cause lower significant bits to be made zero.

Combined with signed and unsigned, and left and right alignment for 12-, 10-, and 8-bit conversion, the result data format can be shown as follows:

**Table 22-1. Result data format**

| Alignment | Signed/ unsigned | Resolution | Result Register | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Right | Unsigned | 12 | – | – | – | – | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 10 | – | – | – | – | – | – | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 8 | – | – | – | – | – | – | – | – | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Right | Signed | 12 | 11 | 11 | 11 | 11 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 10 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | 8 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Left | – | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | – | – | – | – |
| | | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | – | – | – | – | – | – |
| | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | – | – | – | – | – | – | – | – |

### 22.4.1.4 Negative input selection

The negative input connection choice affects the voltage range, SNR, and effective resolution (**Table 22-2**). In single-ended mode, negative input of the SAR ADC can be connected to VSSA, VREF, or any of the eight pins with SARMUX connectivity.

**Table 22-2. Negative input selection comparison**

| Single-ended/ differential | Signed/ unsigned | SARMUX Vminus | SARMUX Vplus Range | Result register | Maximum SNR |
|---|---|---|---|---|---|
| Single-ended | N/A[a)] | $V_{SSA}$ | $+V_{REF}$<br>$V_{SSA} = 0$ | 0x7FF<br>0x000 | Better |
| Single-ended | Unsigned | $V_{REF}$ | $+2 \times V_{REF}$<br>$V_{REF}$<br>$V_{SSA} = 0$ | 0xFFF<br>0x800<br>0 | Good |
| Single-ended | Signed | $V_{REF}$ | $+2 \times V_{REF}$<br>$V_{REF}$<br>$V_{SSA} = 0$ | 0x7FF<br>0x000<br>0x800 | Good |
| Single-ended | Unsigned | Vx | $Vx + V_{REF}$<br>Vx<br>$Vx - V_{REF}$ | 0xFFF<br>0x800<br>0 | Best |
| Single-ended | Signed | Vx | $Vx + V_{REF}$<br>Vx<br>$Vx - V_{REF}$ | 0x7FF<br>0x000<br>0x800 | Best |
| Differential | Unsigned | Vx | $Vx + V_{REF}$<br>Vx<br>$Vx - V_{REF}$ | 0xFFF<br>0x800<br>0 | Best |
| Differential | Signed | Vx | $Vx + V_{REF}$<br>Vx<br>$Vx - V_{REF}$ | 0x7FF<br>0x000<br>0x800 | Best |

a) For single-ended mode with Vminus connected to $V_{SSA}$, conversions are effectively 11-bit because voltages cannot swing below $V_{SSA}$ on any EZ-PD™ PMG1-S3 MCU pin. Because of this, the global configuration bit SINGLE_ENDED_SIGNED (SAR_SAMPLE_CTRL[2]) will be ignored and the result is always (0x000-0x7FF).

To get a single-ended conversion with 12-bits, it is necessary to connect VREF to the negative input of the SAR ADC; then, the input range can be from 0 to 2 × VREF.

Note that single-ended conversions with Vminus connected to the pins with SARMUX connectivity are electrically equivalent to differential mode. However, when the odd pin of each differential pair is connected to the common alternate ground, these conversions are 11-bit, because measured signal value (SARMUX.vplus) cannot go below ground.

### 22.4.1.5 Resolution

EZ-PD™ PMG1-S3 MCU supports 12-bit resolution (default) and a selectable alternate resolution: either 8-bit or 10-bit for each channel. Resolution affects conversion time:

Conversion time (sar_clk) = resolution (bit) + 2

Total acquisition and conversion time (sar_clk) = acquisition time + resolution (bit) + 2

For 12-bit conversion and acquisition time = 4, 18 sar_clk is required. For example, if sar_clk is 18 MHz, 18 sar_clk is required for conversion and you will get 1 Msps conversion rate. Lower resolution results in higher conversion rate.

### 22.4.1.6 Acquisition time

Acquisition time is the time taken by sample and hold (S/H) circuit inside SAR ADC to settle. After acquisition time, the input signal source is disconnected from the SARADC core, and the output of the S/H circuit will be used for conversion. Each channel can select one from four acquisition time options, from 4 to 1023 SAR clock cycles defined in global configuration registers SAR_SAMPLE_TIME01 and SAR_SAMPLE_TIME23.

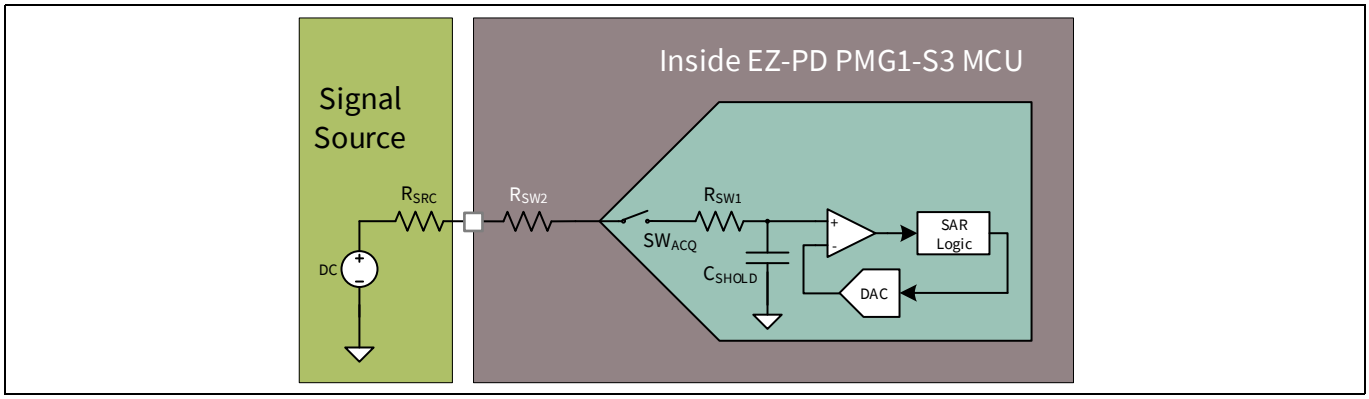



**Figure 22-3. Acquisition time**

The acquisition time should be sufficient to charge the internal hold capacitor of the ADC through the resistance of the routing path, as shown in **Figure 22-3**. The recommended value of acquisition time is:

tACQ 9 × (RSRC + RSW2 + RSW1) × CSHOLD

Where:

CSHOLD ~= 10 pF

RSW2 + RSW1 = ~ 500 to 1000 ohms, depending on the routing path (See **"Analog routing"** on page 214 for details).

RSRC = series resistance of the signal source

### 22.4.1.7 SAR ADC clock

SAR ADC clock frequency must be between 1 MHz and 18 MHz, which comes from the HFCLK via a clock divider. Note that a fractional divider is not supported for SAR ADC. To get a 1-Msps sample rate, an 18-MHz SAR ADC clock is required. To achieve this, the system clock (HFCLK) must be set to 36 MHz rather than 48 MHz. A 12-bit ADC conversion with the minimum acquisition time of four clocks (at 18 MHz) requires 18 clocks in which to complete. A 10-bit and 8-bit conversion requires 16 and 14 clocks respectively. Note that the minimum acquisition time of four clock cycles at 18 MHz is based on the minimum acquisition time supported by the SAR block (RSW1 and CSHOLD in **Figure 22-3**), which is 194 ns.

### 22.4.1.8 SAR ADC timing

**Figure 22-4** shows the SAR ADC timing diagram. A 12-bit resolution conversion needs 14 clocks (one bit needs one sar_clk, plus two excess sar_clk for G and F state). With acquisition time equal to four sar_clk cycles by default, 18 clock sar_clk cycles are required for total ADC acquisition and conversion. After sample (acquisition), it will output the next pulse. The SARMUX can route to another pin and signal; this will be done automatically with sequencer control (see **"SARSEQ"** on page 221 for details).

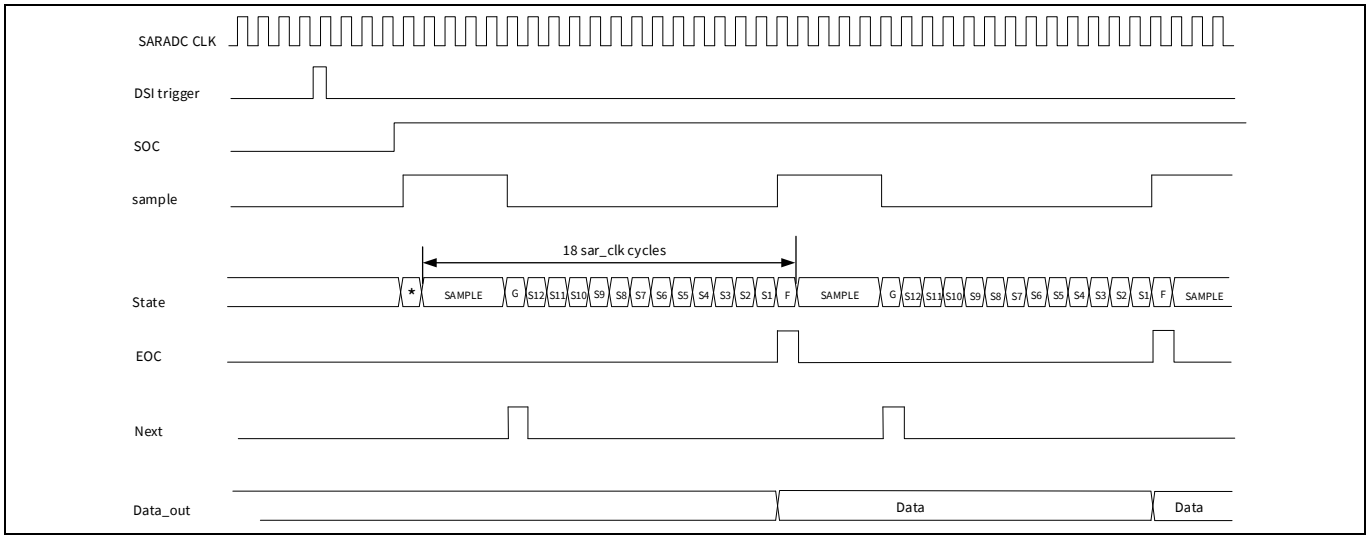



**Figure 22-4. SAR ADC timing**

## 22.4.2 SARMUX

SARMUX is an analog dedicated programmable multiplexer. The main features of SARMUX are:

- Switch on resistance: 600 Ohm (maximum)
- Internal temperature sensor
- Controlled by sequencer controller block (SARSEQ) or firmware
- Charge pump inside:
  - If VDDA < 4.0 V, charge pump should be turned on to reduce switch resistance
  - If VDDA
  - >= 4.0 V, charge pump is turned off and delivers VDDA as its output
- Multiple inputs:
  - Analog signals from pins (port 2)
  - Temperature sensor output
  - CTBm output via sarbus0/1 (not fast enough to sample at 1 Msps)
  - AMUXBUS_A/B (not fast enough to sample at 1 Msps)

### 22.4.2.1 Analog routing

SARMUX has many switches that may be controlled by SARSEQ block (sequencer controller) or firmware. The sequencer is the hardware control method, which can be masked by the hardware control bit in the register, SAR_MUX_SWITCH_HW_CTRL. Different control methods have different control capability on the switches. See **Figure 22-5**.

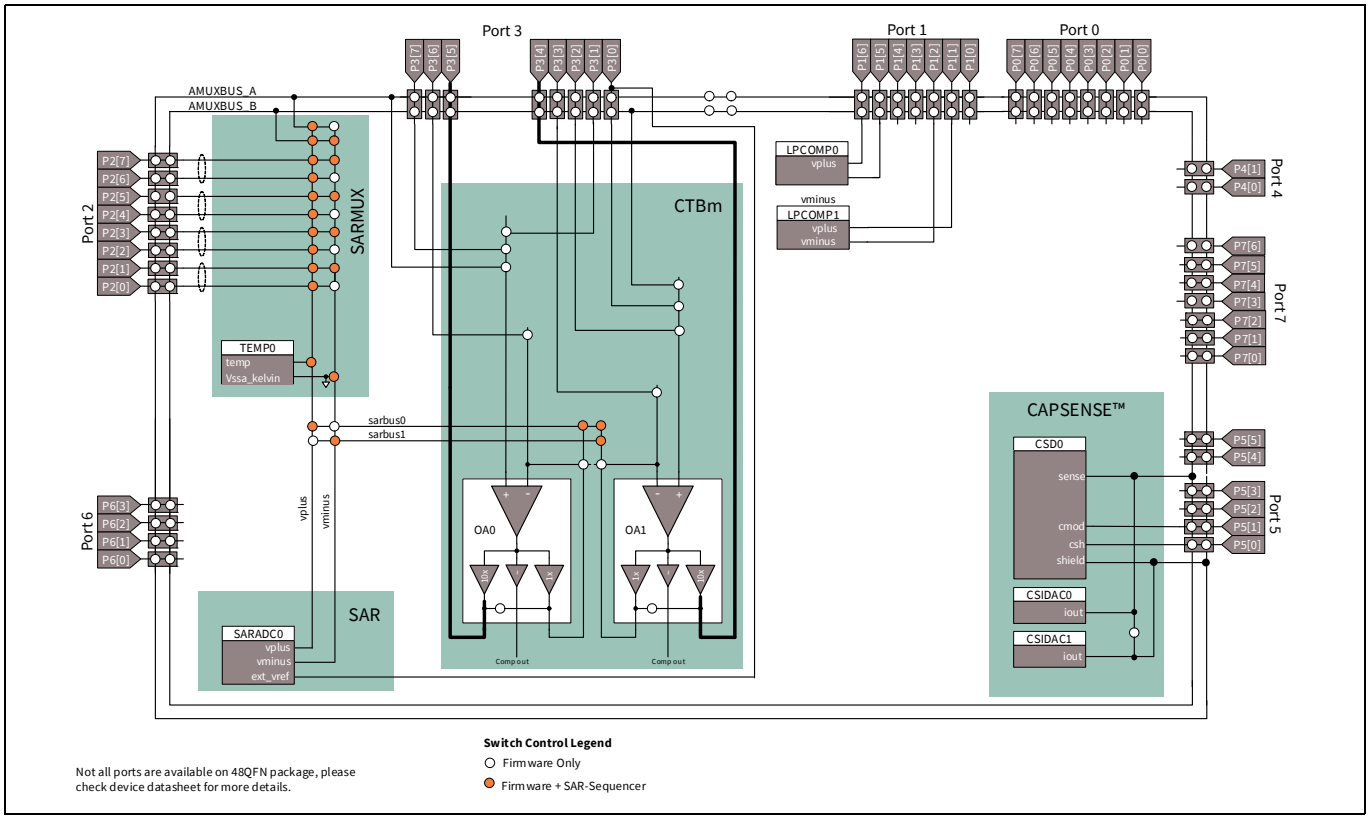



**Figure 22-5. SARMUX switches and control capability**

**Sequencer control:** The switches are controlled by the sequencer in SARSEQ block. After configuring each channel's analog routing, it enables multi-channel automatic scan in a round-robin fashion, without CPU intervention. Not every switch can be controlled by the sequencer; see **Figure 22-5**. The corresponding registers are: SAR_CHANx_CONFIG, SAR_MUX_SWITCH0, SAR_CTRL, and SAR_MUX_SWITCH_HW_CTRL.

**Firmware control:** Programmable registers directly define the VPLUS/VMINUS connection. It can control every switch in SARMUX; see **Figure 22-5**. For example, in firmware control, it is possible to do a differential measurement between any two pins or signals, not just two adjacent pins (as in sequencer control). However, it needs CPU intervention for multi-channel acquisition. The corresponding registers are: SAR_MUX_SWITCH0, SAR_MUX_SWITCH_HW_CTRL. and SAR_CTRL.

### 22.4.2.2 Analog interconnection

EZ-PD™ PMG1-S3 MCU analog interconnection is very flexible. SAR ADC can be connected to multiple inputs via SARMUX, including both external pins and internal signals. For example, it can connect to a neighbouring block such as CTBm. It can also connect to other pins except port 2 through AMUXBUS_A/B, at the expense of scanning performance (more parasitic coupling, longer RC time to settle).

Several cases are discussed here to provide a better understanding of analog interconnection.

#### Input from external pins

**Figure 22-6** shows how two GPIOs that support SARMUX are connected to SAR ADC as a differential pair (Vpuls/Vminus) via switches. These two switches can be controlled by sequencer, or firmware. The pins are arranged in adjacent pairs; for example, in SARMUX port P2[0] and P2[1], P2[2] and P2[3], and so on. If you need to use pins that are not paired as a differential pair, such as P2[1] and P2[2], the sequencer does not work; use firmware.

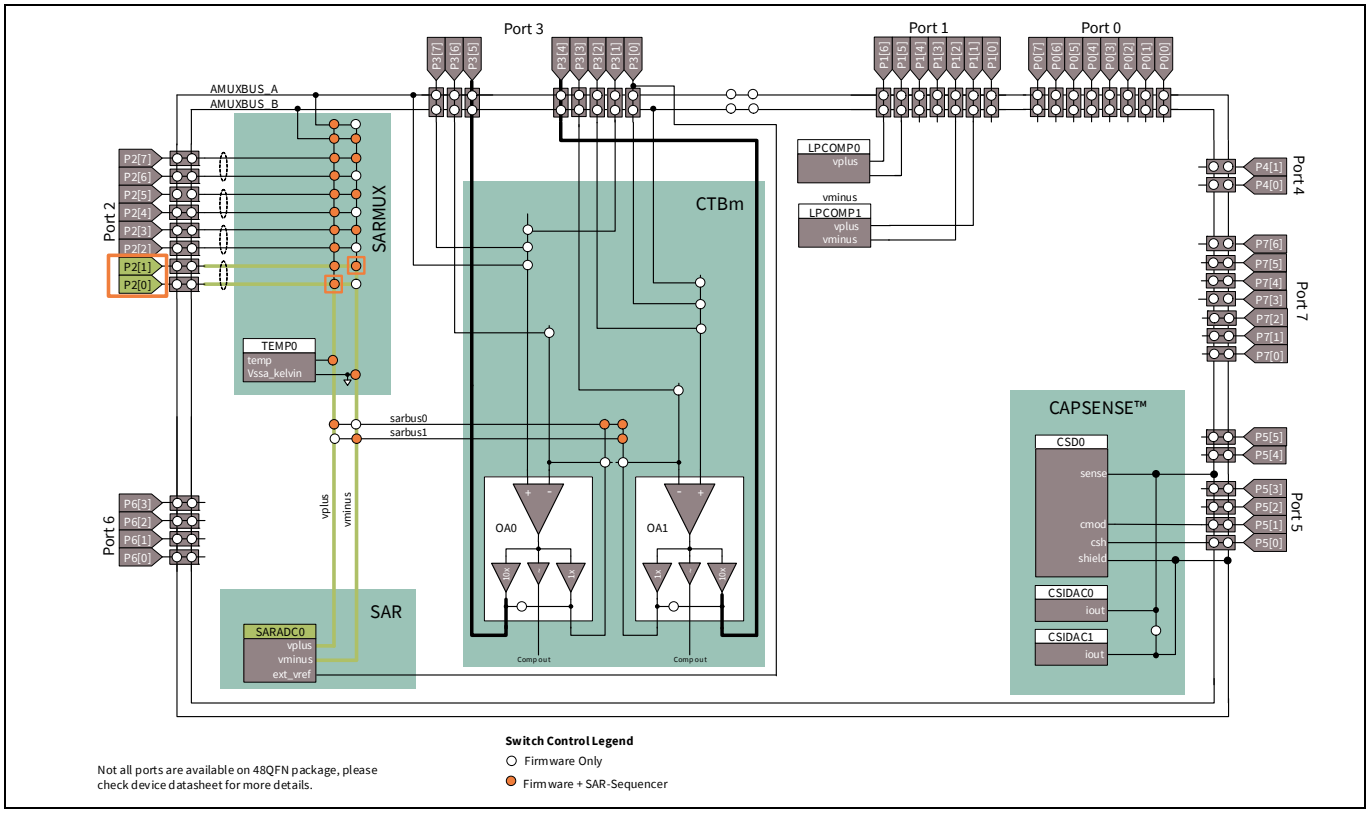



**Figure 22-6. Input from external pins**

## 12-bit SAR ADC

### Input from analog bus (AMUXBUS_A/B)

Figure 22-7 shows how two pins that do not support SARMUX connectivity are connected to ADC as a differential pair. Additional switches must connect these two pins to AMUXBUS_A and AMUX-BUS_B, and then connect AMUXBUS_A and AMUXBUS_B to ADC.

The additional switches reduce the scanning performance (more parasitic coupling, longer RC time to settle) – it is not fast enough to sample at 1 Msps. This is not recommended for external signals; the dedicated SARMUX port should be used, if possible.

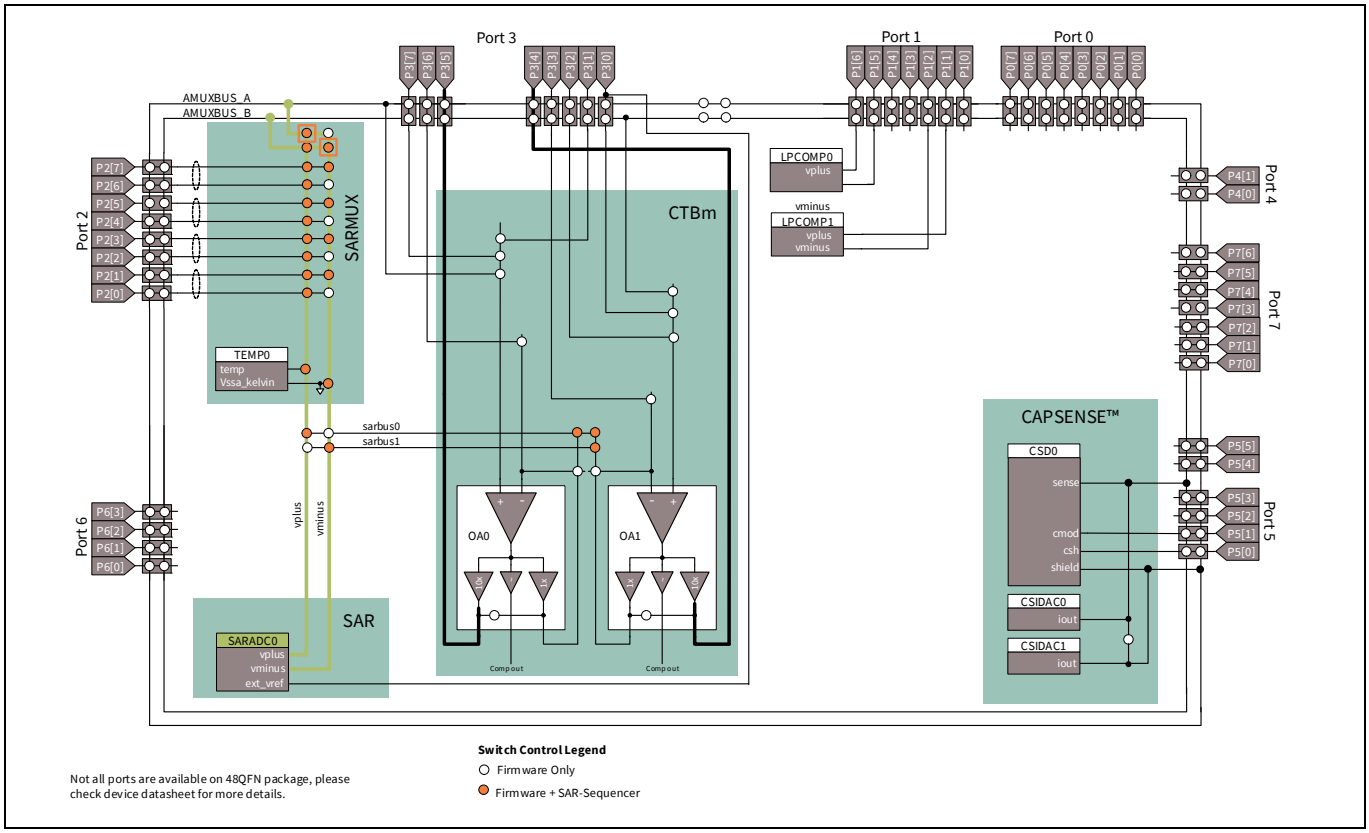



**Figure 22-7. Input from analog bus**

## 12-bit SAR ADC

### Input from CTBm output via sarbus

SAR ADC can be connected to CTBm output via sarbus 0/1. **Figure 22-8** shows how to connect an opamp (configured as a follower) output to a single-ended SAR ADC. Negative terminal is connected to VREF. **Figure 22-9** shows how to connect two opamp outputs to SAR ADC as a differential pair. It must connect opamp output to sarbus 0/1, then connect SAR ADC input to sarbus 0/1. Because there are also additional switches, it is not fast enough to sample at 1 Msps. However, the on-chip opamps add value for many applications.

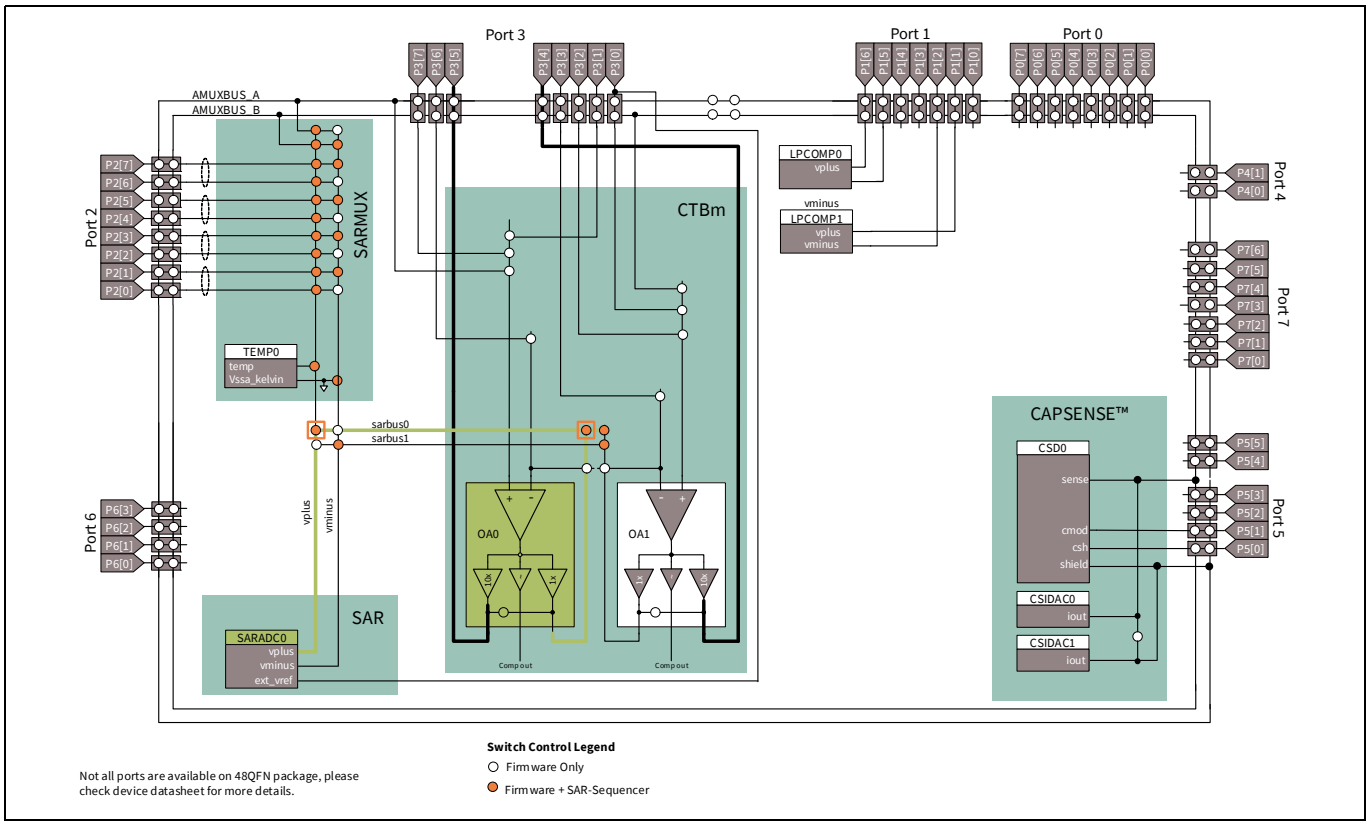



**Figure 22-8. Input from CTBm output via sarbus**

## 12-bit SAR ADC

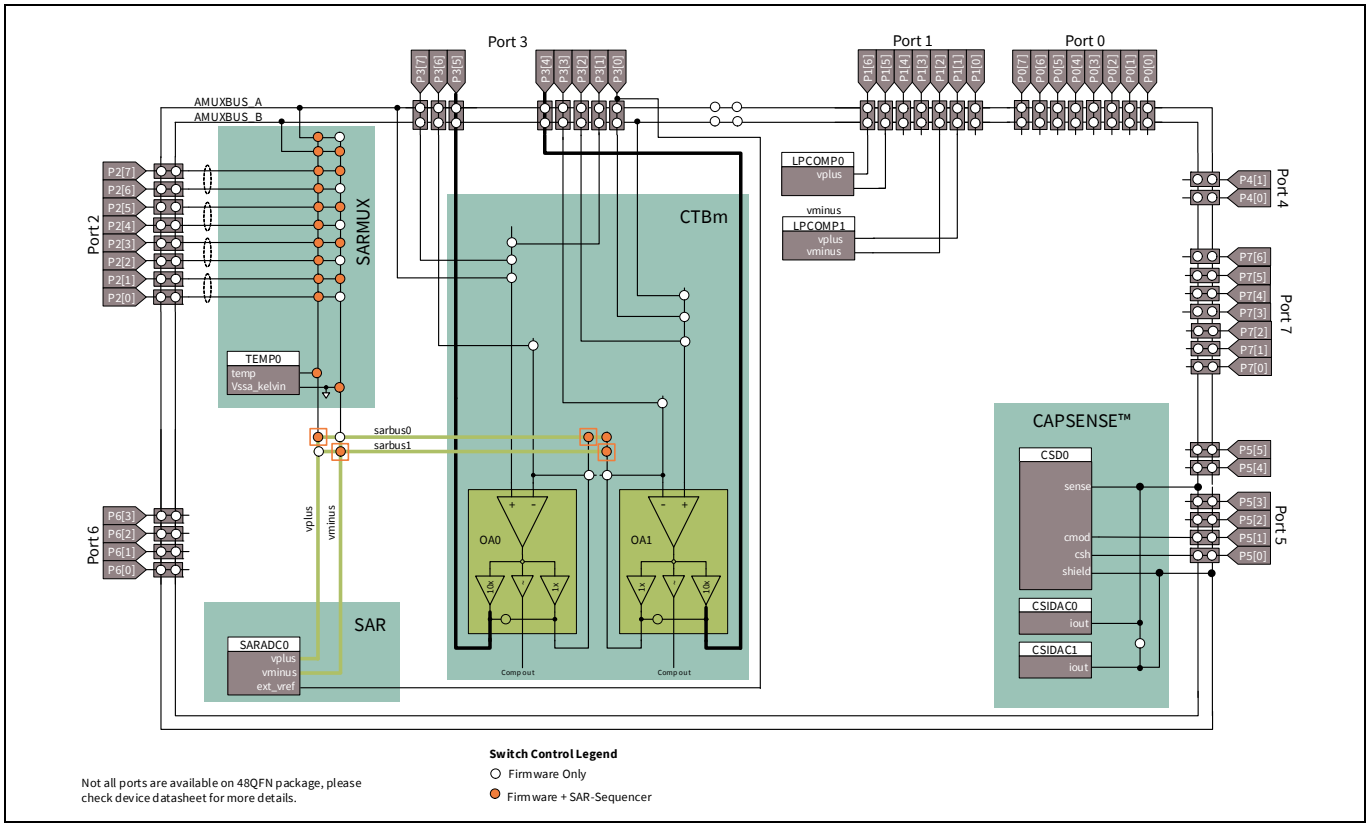



**Figure 22-9. Inputs from CTBm output via sarbus0 and sarbus1**

## 12-bit SAR ADC

### Input from temperature sensor

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Note for temperature sensor, differential conversions are not available (conversion result is undefined), thus always use it in singled-ended mode.

As **Figure 22-10** shows, temperature sensor can be routed to positive input of SAR ADC via switch, which can be controlled by sequencer, firmware. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit will disable temperature sensor by cutting its bias current.

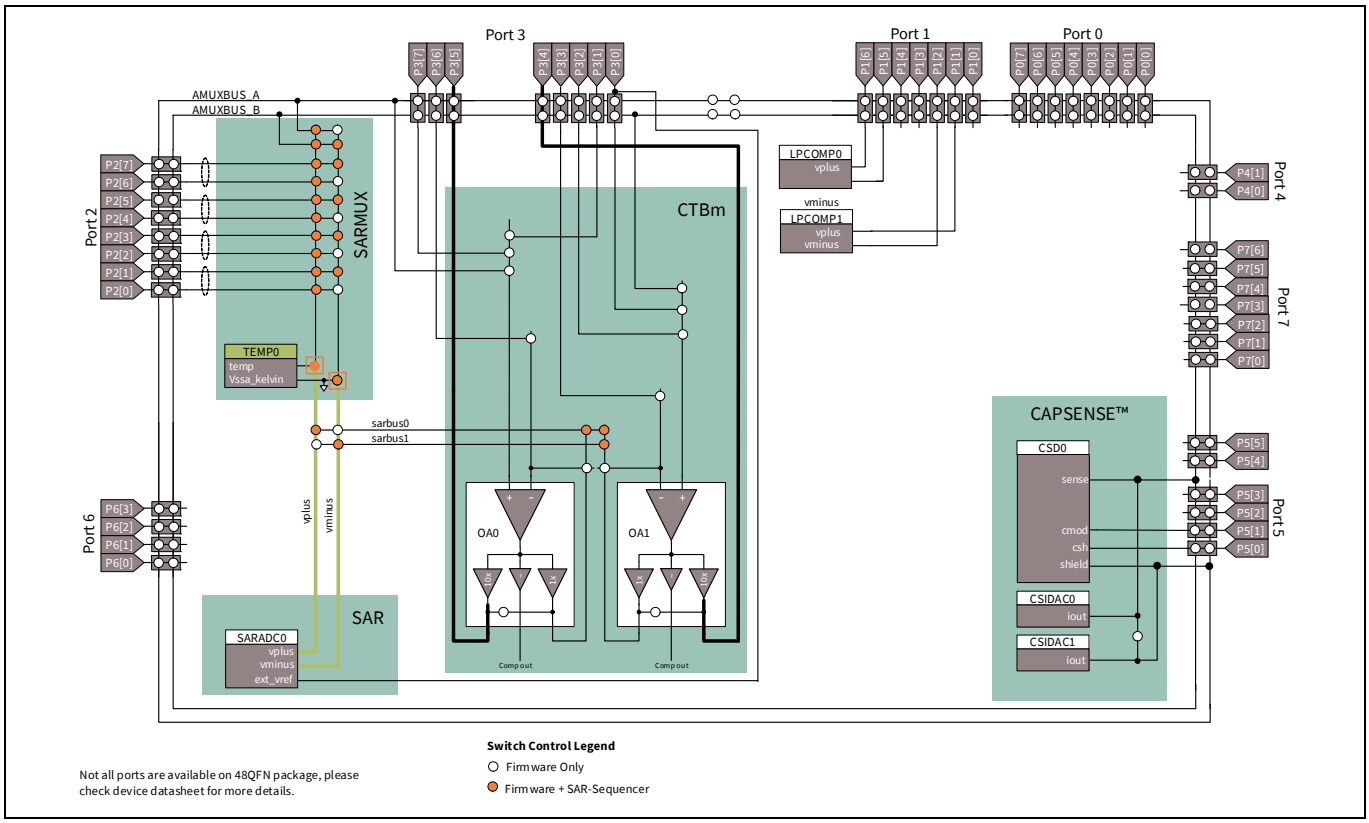



**Figure 22-10. Inputs from temperature sensor**

### 22.4.3 SARREF

The main features of SARREF are:

- Reference options: VDDA, VDDA /2, 1.2-V bandgap (±1 percent), external reference
- Reference buffer + bypass cap to enhance internal reference drive capability

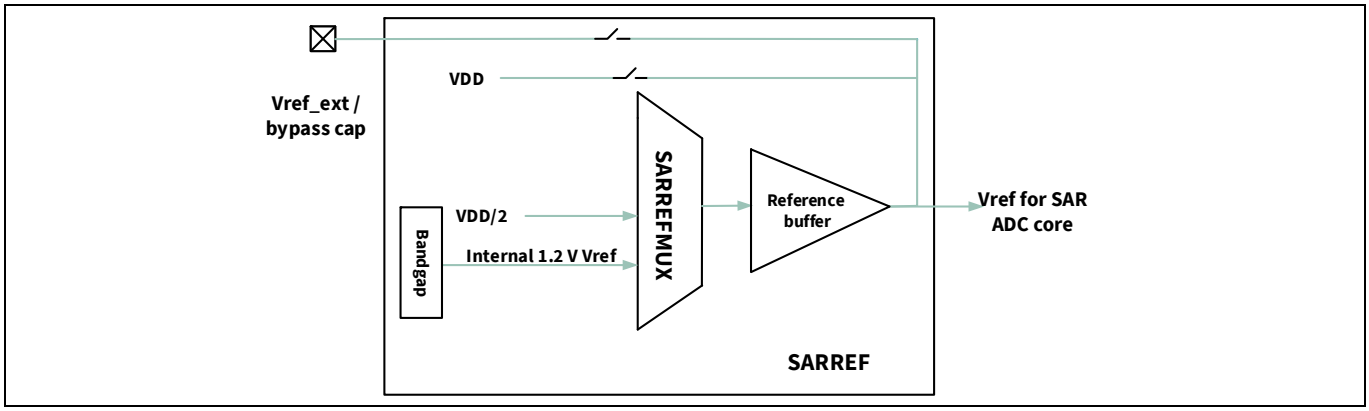



**Figure 22-11. SARREF block diagram**

#### 22.4.3.1 Reference options

The reference voltage selection for the SAR ADC consists of a reference mux and switches inside the SARREF. The selection allows connecting VDDA, VDDA /2, and 1.2-V internal reference from a bandgap or an external VREF connected to an Ext Vref/SAR bypass pin (see the **device datasheet** for details). The control for the reference mux in SARREF is in the global configuration register SAR_CTRL [6:4].

#### 22.4.3.2 Bypass capacitors

The internal references, 1.2 V from bandgap or VDDA /2 are buffered with the reference buffer. This reference may be routed to the Ext Vref/SAR bypass pin where an external capacitor can be used to filter internal noise that may exist on the reference signal. The SAR ADC sample rate is limited to 100 ksps (at 12-bit) without an external reference bypass capacitor. For example, without a bypass capacitor and with 1.2-V internal VREF, the maximum SAR ADC clock frequency is 1.6 MHz. When using an external reference, it is recommended that an external capacitor is used. Bypass capacitors can be enabled by setting SAR_CTRL [7]. **Table 22-3** lists different reference modes and its maximum frequency/sample rate for 12-bit continuous mode operation.

**Table 22-3. Reference modes**

| Reference mode | Reference SAR_CTRL [6:4] | Bypass cap SAR_CTRL[7] | Buffer | Max frequency | Max sample rate (12-bit) |
|---|---|---|---|---|---|
| 1.2 V internal $V_{REF}$ without bypass cap | 4 | 0 | Yes | 1.6 MHz | 100 ksps |
| 1.2 V internal $V_{REF}$ with bypass cap | 4 | 1 | Yes | 18 MHz | 1 Msps |
| External $V_{REF}$ (low-impedance path) | 5 | X | No | 18 MHz | 1 Msps |
| $V_{DDA}$/2 without bypass cap | 6 | 0 | Yes | 1.6 MHz | 100 ksps |
| $V_{DDA}$/2 with bypass cap | 6 | 1 | Yes | 18 MHz | 1 Msps |
| $V_{DDA}$ | 7 | X | No | 9 MHz | 500 ksps |

1.2-V internal VREF startup time varies with the different bypass capacitor size, **Table 22-4** lists two common values for the bypass capacitor and its startup time specification. If reference selection is changed between scans or when scanning after Sleep/Deep-Sleep, make sure the 1.2-V internal VREF is settled when SAR ADC starts sampling. The worst case settling time (when VREF is completely discharged) is the same as the startup time.

**Table 22-4. Bypass capacitor values**

| Internal $V_{REF}$ startup time | Maximum specification |
|---|---|
| Startup time for reference with external capacitor (1 uF) | 2 ms |
| Startup time for reference with external capacitor (100 nF) | 200 µs |

#### 22.4.3.3 Input range versus reference

All inputs should be between VSSA and VDDA. The ADCs input range is limited by VREF selection. If negative input is Vn and the ADC reference is VREF, the range on the positive input is Vn ± VREF. This criteria applies for both single-ended and differential modes as long as both negative and positive inputs stay within VSSA to VDDA.

### 22.4.4 SARSEQ

SARSEQ is a dedicated sequencer controller that automatically sequences the input mux from one channel to the next while placing the result in an array of registers, one per channel.

- Controls SARMUX analog routing automatically without CPU intervention
- Controls SAR ADC core (such as resolution, acquisition time, and reference)
- Receives data from SAR ADC and does pre-processing (average, range detect)
- Uses double buffer to store the result so the CPU can safely read the results of the last scan while the next scan is in progress.

The features of SARSEQ are:

- Sixteen channels can be individually enabled as an automatic scan without CPU intervention
- An additional channel (injection channel) for infrequent
- signal to insert in an automatic scan
- Each channel has the following features:
  - Single-ended or differential mode
  - Input from external pin (only for eight channels in
  - single-ended mode and four channels in differential
  - mode) or internal signal (AMUXBUS/CTBm/temperature sensor)
  - Up to four programmable acquisition time
  - Default 12-bit resolution, selectable alternate resolution: either 8-bit or 10-bit
  - Result averaging
- Scan triggering
  - One shot, periodic, or continuous mode
  - Triggered by TCPWM block, CTBm comparator, low-power comparator, SAR ADC end of conversion signal, and by firmware
- Hardware averaging support
  - First order accumulate
  - From 2 to 256 samples averaging (powers of 2)
  - Results in 16-bit representation
- Double buffering of output data
  - Left or right adjusted results
  - Results in working register and result register
- Interrupt generation
  - Finished scan conversion
  - Channel saturation detect in all control modes
  - Over range (configurable) detect for every channel
  - Scan results overflow
  - Collision detect

## 12-bit SAR ADC

- Configurable injection channel
  - Triggered by firmware
  - Can be interleaved between two scan sequences
  - (tailgating)
  - Selectable sample time, resolution, single ended, or differential, averaging

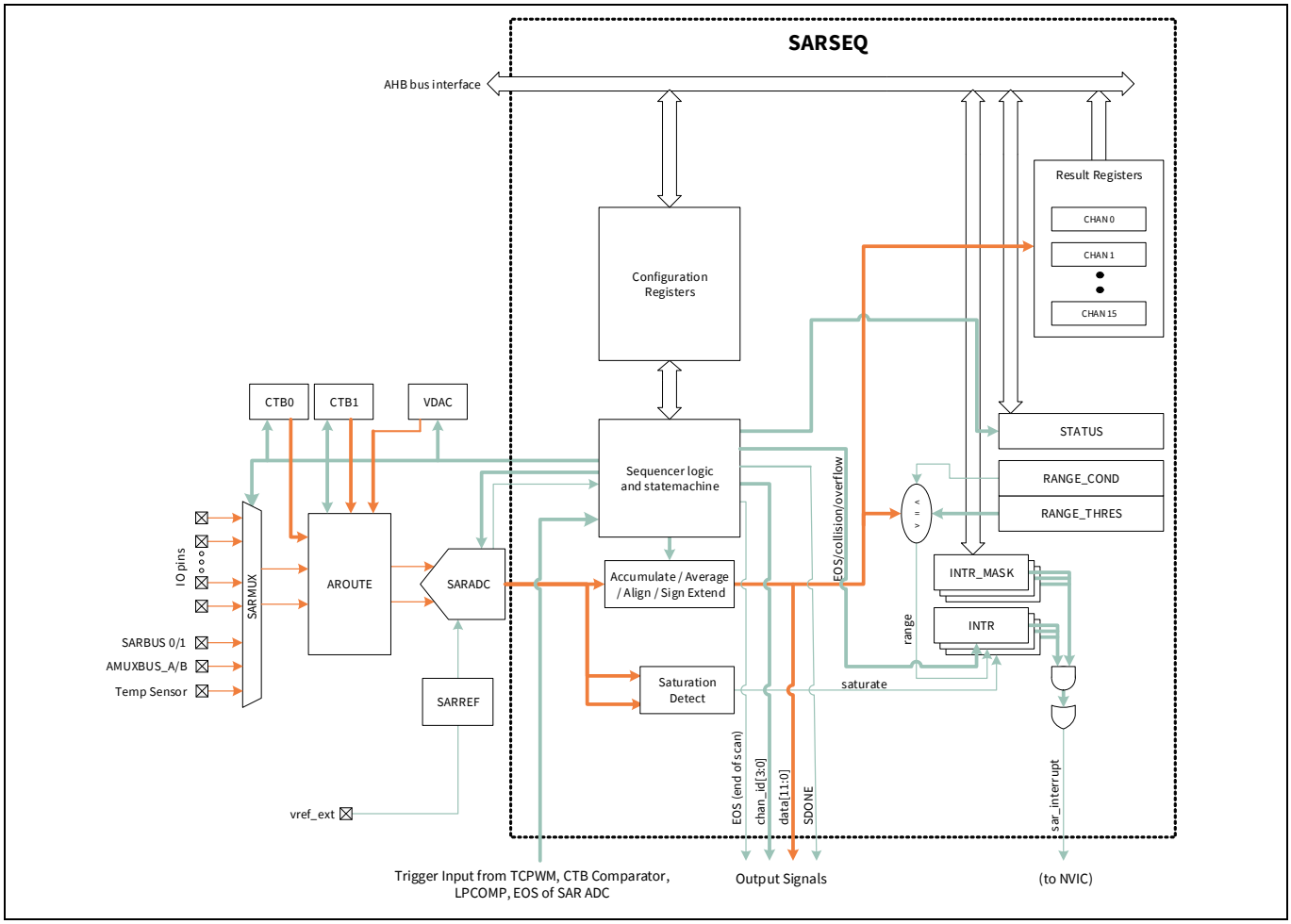



**Figure 22-12. SARSEQ block diagram**

### 22.4.4.1 Averaging

The SARSEQ block has a 20-bit accumulator and shift register to implement averaging. Averaging is after signed extension. The global configuration SAR_SAMPLE_CTRL register specifies the details of averaging.

Channel configuration SAR_CHAN_CONFIG register has an enable bit (AVG_EN) to enable averaging.

In global configuration, AVG_CNT (SAR_SMAPLE_CTRL [6:4]) specifies the number of samples (N) according to this formula:

N=2^(AVG_CNT+1) N range = [2..256]

For example, if AVG_CNT (SAR_SMAPLE_CTRL [6:4]) = 3, then N = 16.

AVG_SHIFT bit (SAR_SAMPLE_CTRL[7]) is used to shift the result to get averaged; it should be set if averaging is enabled.

If a channel is configured for averaging, the SARSEQ will take N consecutive samples of the specified channel in every scan. Because the conversion result is 12-bit and the maximum value of N is 256 (left shift 8 bits), the 20-bit accumulator will never overflow.

If AVG_SHIFT in SAR_SAMPLE_CTRL register is set, SAR sequencer performs sign extension and then accumulation. The accumulated result is shifted right AVG_CNT + 1 bits to get averaged. If it is not, the result is forced to shift right to ensure it fits in 16 bits. Right shift is done by maximum (0, AVG_CNT-3) – if the number of samples is more than 16 (AVG_CNT >3), then the accumulation result is shifted right AVG_CNT-3bits; it AVG_CNT<3, the result is not shifted.

Note in this case, the average result is bigger than expected; it is recommended to set AVG_SHIFT. This mode always uses the selected resolution of ADC (12, 10, or 8 bits).

### 22.4.4.2 Range detection

The SARSEQ supports range detection to allow automatic detection of result values compared to two programmable thresholds without CPU involvement. Range detection is defined by the SAR_RANGE_THRES register. The RANGE_LOW field (SAR_RANGE_THRES [15:0]) value defines the lower threshold and RANGE_HIGH field (SAR_RANGE_THRES [31:16]) defines the upper threshold of the range.

The SAR_RANGE_COND bits define the condition that triggers a channel maskable range detect interrupt (RANGE_INTR). The following conditions can be selected:

0: result < RANGE_LOW (below range)

1: RANGE_LOW ?result < RANGE_HIGH (inside range)

2: RANGE_HIGH ?result (above range)

3: result <RANGE_LOW || RANGE_HIGH <= result (outside range)

See **“Range detection interrupts”** on page 226 for details.

### 22.4.4.3 Double buffer

Double buffering is used so that firmware can read the results of a complete scan while the next scan is in progress. The SAR ADC results are written to a set of working registers until the scan is complete, at which time the data is copied to a second set of registers where the data can be read by the user's application. This action allows sufficient time for the firmware to read the previous scan before the present scan is completed. All input channels are double buffered with 16 registers, except the injection channel. The injection channel is not required to be doubled buffered because it is not normally part of a normal channel scan.

### 22.4.4.4 Injection channel

The injection channel is similar to the other channels, with the exception that it is not part of a regular scan. The injection channel is used for incidental or rare conversions; for example, sampling the temperature sensor every two seconds. Note that if SAR is operating in continuous mode, enabling the injection channel will change the sample rate.

The injection channel can only be controlled by the firmware with a firmware trigger (one-shot). This means the injection channel does not support continuous trigger. Because the only trigger is one-shot, there is no need for double buffering or an overflow interrupt.

The conversions for the injection channel can be configured in the same way as the regular channels by setting SAR_INJ_CHAN_CONFIG register, it supports:

- Pin or signal selection
- Single-ended or differential selection
- Choice of resolution between 12-bit or the globally specified SUB_RESOLUTION
- Sample time select from one of the four globally specified sample times
- Averaging select

It supports the same interrupts as the regular channel except the overflow interrupt.

- Maskable end-of-conversion interrupt INJ_EOC_INTR
- Maskable range detect interrupt INJ_RANGE_INTR

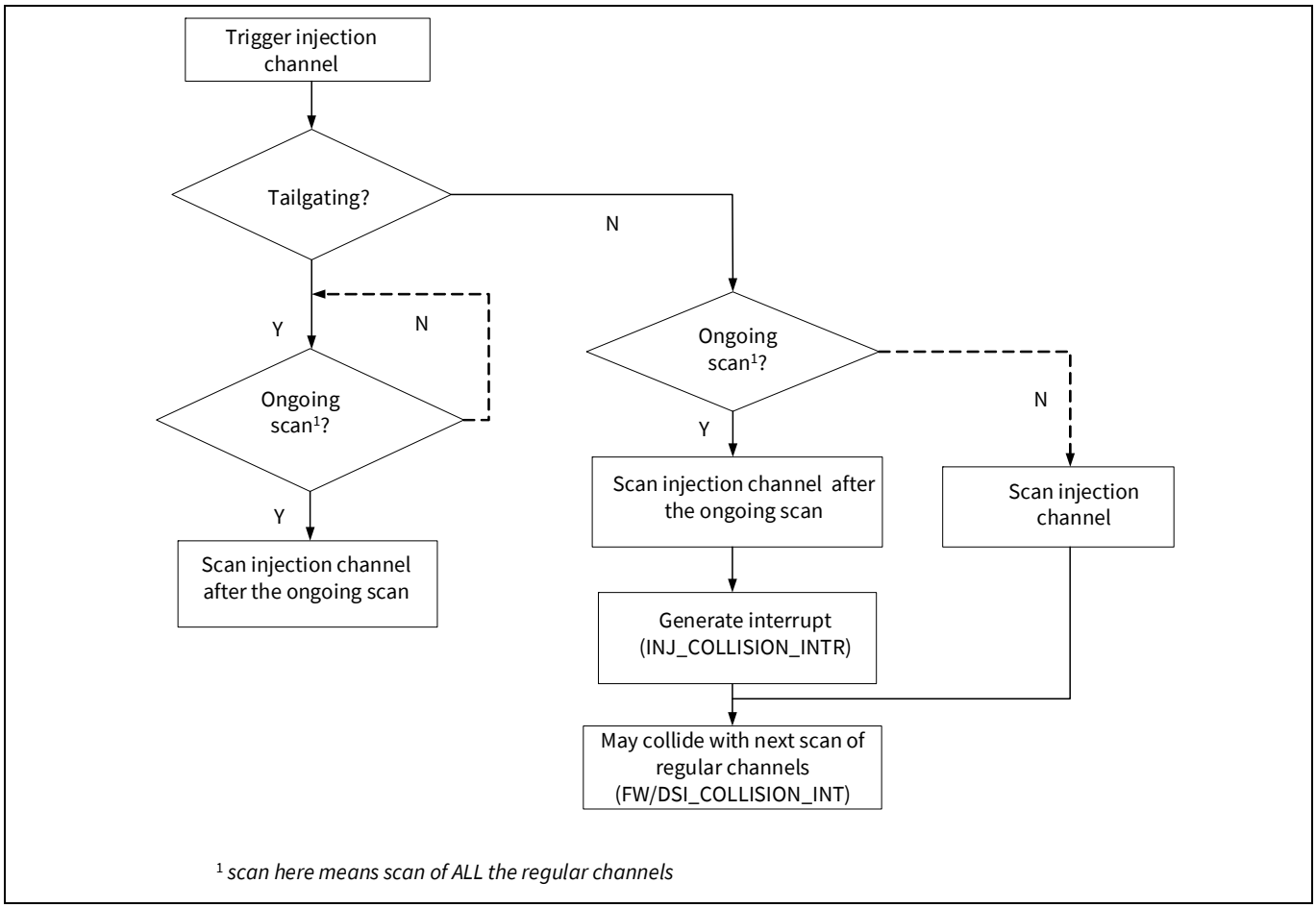



**Figure 22-13. Injection channel flow chart**

The disadvantage of tailgating is that it may be a long time before the next trigger occurs. If there is no risk of colliding or causing jitter on the regular channels, the injection channel can be used safely without tailgating.

After completing the conversion for the injection channel, the end-of conversion interrupt (INJ_EOC_INTR) is set and the INJ_START_EN bit is cleared. The conversion data of the injection is put in the SAR_INJ_RESULT register. Similar to the SAR_CHAN_RESULT, the registers contain mirror bits for “valid” (=INJ_EOC_INTR), range detect, saturation detect interrupt, and a mirror bit of the collision interrupt (INJ_COLLISSION_INTR).

**Figure 22-14** is an example when injection channel is enabled during a continuous scan (channel 1, 3, 5, and 7 are enabled), and tailgating is enabled. Note that the INJ_START_EN bit is immediately cleared when the SAR is disabled (but only if it was enabled before).

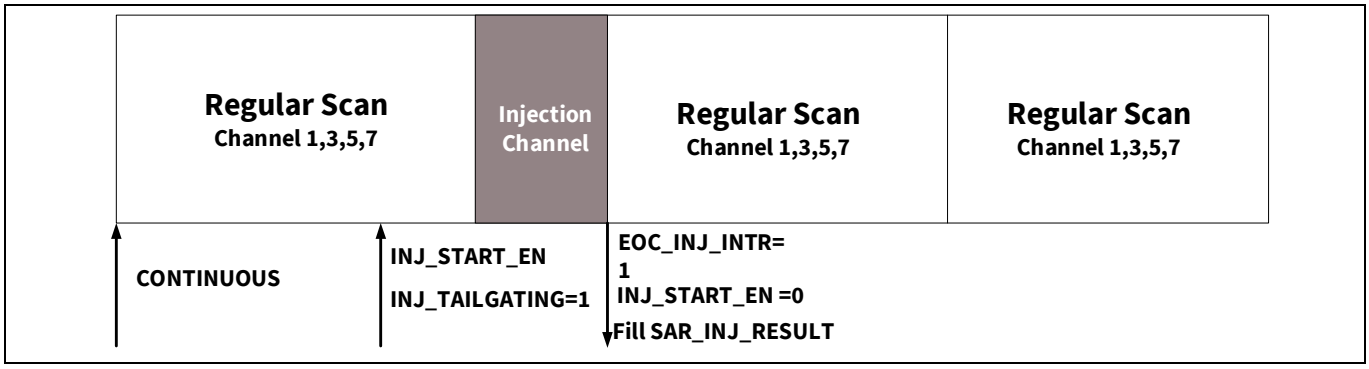



**Figure 22-14. Injection channel enabled with tailgating**

## 22.4.5 Interrupt

An interrupt can be generated on different events:

- End of scan – When scanning is complete for all the enabled channels.
- Overflow – When the result register is updated before the previous result is read.
- Collision – When a new trigger is received while the SAR ADC is still processing the previous trigger.
- Injection end of conversion – When the injection channel is converted.
- Range detection – When the channel result meets the threshold value.
- Saturation detection – When the channel result is equal to the minimum or maximum value of the set resolution.

This section describes each interrupt in detail. These interrupts have an interrupt mask in the SAR_INTR_MASK register. By making the interrupt mask low, the corresponding interrupt source is ignored. The SAR interrupt is generated if the interrupt mask bit is high and the corresponding interrupt source is pending.

When servicing an interrupt, the interrupt service routine (ISR) clears the interrupt source by writing a ‘1’ to the interrupt bit after reading the data.

The SAR_INTR_MASKED register is the logical AND between the interrupts sources and the interrupt mask. This register provides a convenient way for the firmware to determine the source of the interrupt.

For verification and debug purposes, a set bit (such as EOS_SET in the SAR_INTR_SET register) is used to trigger each interrupt. This action allows the firmware to generate an interrupt without the actual event occurring.

### 22.4.5.1 End-of-scan interrupt (EOS_INTR)

After completing a scan, the end-of-scan interrupt (EOS_INTR) is raised. Firmware should clear this interrupt after picking up the data from the RESULT registers. Optionally, the EOS_INTR can also be sent out on the GPIO by setting the EOS_DSI_OUT_EN bit in SAR_SAMPLE_CTRL [31]. The EOS_INTR signal is maintained for two system clock cycles. These cycles coincide with the data_valid signal for the last channel of the scan (if selected).

EOS_INTR can be masked by making the EOS_MASK bit 0 in the SAR_INTR_MASK register. EOS_MASKED bit of the SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a ‘1’ to EOS_SET bit in SAR_INTR_SET register can set the EOS_INTR, which is intended for debug and verification.

### 22.4.5.2 Overflow interrupt

If a new scan completes and the hardware tries to set the EOS_INTR and EOS_INTR is still high (firmware does not clear it fast enough), then an overflow interrupt (OVERFLOW_INTR) is generated by the hardware. This usually means that the firmware is unable to read the previous results before the current scan completes. In this case, the old data is overwritten.

OVERFLOW_INTR can be masked by making the OVERFLOW_MASK bit 0 in SAR_INTR_MASK register. OVER-

FLOW_MASKED bit of SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks, which is for firmware convenience. Writing a '1' to the OVERFLOW_SET bit in SAR_INTR_SET register can set OVERFLOW_INTR, which is intended for debug and verification.

### 22.4.5.3 Collision interrupt

It is possible that a new trigger is generated while the SARSEQ is still busy with the scan started by the previous trigger. Therefore, the scan for the new trigger is delayed until after the ongoing scan is completed. It is important to notify the firmware that the new sample is invalid. This is done through the collision interrupt, which is raised any time a new trigger, other than the continuous trigger, is received.

There are three collision interrupts: for the firmware trigger (FW_COLLISION_INTR), for the external trigger (DSI_COLLISION_INTR), and for the injection channel (INJ_COLLISION_INTR). These interrupts allow the firmware to identify which trigger collided with an ongoing scan.

When the external trigger is used in level mode, the DSI_COLLISION_INTR will never be set. The three collision interrupts can be masked by making the corresponding bit '0' in the SAR_INTR_MASK register. The

corresponding bit in the SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a '1' to the corresponding bit in SAR_INTR_SET register can set the collision interrupt, which is intended for debug and verification.

### 22.4.5.4 Injection end-of-conversion interrupt (INJ_EOC_INTR)

After completing a conversion for the injection channel, the injection end-of-conversion interrupt is raised (INJ_EOC_INTR). The firmware clears this interrupt after picking up the data from the INJ_RESULT register.

Note that if the injection channel is tailgating a scan, the EOS_INTR is raised in parallel to starting the injection channel conversion. The injection channel is not considered part of the scan.

INJ_EOC_INTR can be masked by making the INJ_EOC_MASK bit '0' in the SAR_INTR_MASK register. The INJ_EOC_MASKED bit of SAR_INTR_MASKED register is the logic AND of the interrupt flags and the interrupt masks. Writing a '1' to the INJ_EOC_SET bit in SAR_INTR_SET register can set INJ_EOC_INTR, which is intended for debug and verification.

### 22.4.5.5 Range detection interrupts

Range detection interrupt flag can be set after averaging, alignment, and sign extension (if applicable). This means its not required to wait for the entire scan to complete to determine whether a channel conversion is over-range. The threshold values need to have the same data format as the result data.

Range detection interrupt for a specified channel can be masked by setting the SAR_RANGE_INTR_MASK register specified bit to '0'. Register SAR_RANGE_INTR_MASKED reflects a bitwise AND between the interrupt request and mask registers. If the value is not zero, then the SAR interrupt signal to the NVIC is high. SAR_RANGE_INTR_SET can be used for debug/verification. Write a '1' to set the corresponding bit in the interrupt request register; when read, this register reflects the interrupt request register. There is a range detect interrupt for each channel (RANGE_INTR and INJ_RANGE_INTR).

### 22.4.5.6 Saturate detection interrupts

The saturation detection is always applied to every conversion. This feature detects if a sample value is equal to the minimum or maximum value for the specific resolution and sets a maskable interrupt flag for the corresponding channel. This action allows the firmware to take action, such as discarding the result, when the SAR ADC saturates. The sample value is tested right after conversion, before averaging. This means that the interrupt is set while the averaged result in the data register is not equal to the minimum or maximum.

When a 10-bit or 8-bit resolution is selected for the channel, saturate detection is done on 10-bit or 8-bit data. Saturation interrupt flag is set immediately to enable a fast response to saturation, before the full scan and averaging. Saturation detection interrupt for specified channel can be masked by setting the SAR_SATURATE_INTR_MASK register specified bit to ‘0’. SAR_SATURATE_INTR_MASKED register reflects a bit-wise AND between the interrupt request and mask registers. If the value is not zero, then the SAR interrupt signal to the NVIC is high. SAR_SARTURATE_INTR_SET can be used for debug/verification. Write a '1' to set the corresponding bit in the interrupt request register; when read, this register reflects the interrupt request register.

### 22.4.5.7 Interrupt cause overview

INTR_CAUSE register contains an overview of all the pending SAR interrupts. It allows the ISR to determine the cause of the interrupt. The register consists of a mirror copy of SAR_INTR_MASKED. In addition, it has two bits that aggregate the range and saturate detection interrupts of all channels. It includes a logical OR of all the bits in RANGE_INTR_MASKED and SATURATE_INTR_MASKED registers (does not include INJ_RANGE_INTR and INJ_SATURATE_INTR).

## 22.4.6 Trigger

The three possible ways to trigger a scan are:

A firmware or one-shot trigger is generated when the firmware writes to the FW_TRIGGER bit of the SAR_START_CTRL register. After the scan is completed, the SARSEQ clears the FW_TRIGGER bit and goes back to idle mode waiting for the next trigger. The FW_TRIGGER bit is cleared immediately after the SAR is disabled.

An external trigger can be TCPWM outputs, CTBm comparator outputs, low-power comparator outputs, and SAR ADC’s end-of-sampling and end-of-conversion signals. Hardware trigger is enabled by writing '1' to the DSI_TRIGGER_EN bit of the SAR_SAMPLE_CTRL register. Signal for the trigger is selected using the PERI_TR_GROUP1_TR_OUT_CTL0 register in EZ-PD™ PMG1-S3 MCU.

**Table 22-5. Hardware trigger source selection in EZ-PD™ PMG1-S3 MCU**

| PERI_TR_GROUP1_TR_OUT_CTL0[6:0] | Trigger source |
|---|---|
| 0 | Hardwired to 0 (firmware trigger) |
| 1 | TCPWM 0 Overflow |
| 2 | TCPWM 1 Overflow |
| 3 | TCPWM 2 Overflow |
| 4 | TCPWM 3 Overflow |
| 5 | TCPWM 4 Overflow |
| 6 | TCPWM 0 Compare Match |
| 7 | TCPWM 1 Compare Match |
| 8 | TCPWM 2 Compare Match |
| 9 | TCPWM 3 Compare Match |
| 10 | TCPWM 4 Compare Match |
| 11 | TCPWM 0 Underflow |

**Table 22-5. Hardware trigger source selection in EZ-PD™ PMG1-S3 MCU** (continued)

| PERI_TR_GROUP1_TR_OUT_CTL0[6:0] | Trigger source |
|---|---|
| 12 | TCPWM 1 Underflow |
| 13 | TCPWM 2 Underflow |
| 14 | TCPWM 3 Underflow |
| 15 | TCPWM 4 Underflow |
| 16 | SAR ADC Sample Done (sdone) Signal |
| 17 | SAR ADC End of Conversion (eoc) Signal |
| 18 | CTBm Comparator 0 Output |
| 19 | CTBm Comparator 1 Output |
| 20 | LPCOMP 0 Output |
| 21 | LPCOMP 1 Output |

A continuous trigger is activated by setting the CONTINUOUS bit in SAR_SAMPLE_CTRL register. In this mode, after completing a scan the SARSEQ starts the next scan immediately; therefore, the SARSEQ is always BUSY. As a result, all other triggers are essentially ignored. Note that FW_TRIGGER will still get cleared by hardware on the next completion.

The three triggers are mutually exclusive, although there is no hardware requirement. If an external trigger coincides with a firmware trigger, the external trigger is handled first and a separate scan is done for the firmware trigger (and a collision interrupt is set). When an external trigger coincides with a continuous trigger, both triggers are effectively handled at the same time (a collision interrupt may be set for the external trigger).

For firmware continuous trigger, it takes only one SAR ADC clock cycle before the sequencer tells the SAR ADC to start sampling (provided the sequencer is idle). For the external trigger, it depends on the trigger configuration setting.

### 22.4.6.1 External trigger configuration

- Synchronization

If the incoming external trigger signal is not synchronous to the AHB clock, the signal needs to be synchronized by double flopping it (default). However, if the trigger signal is already synchronized with the AHB clock, then these two flops can be bypassed. The configuration bit, DSI_SYNC_TRIGGER in the SAR_SAMPLE_CTRL register, controls the double flop bypass. DSI_SYNC_TRIGGER affects the trigger width (TW) and trigger interval (TI) requirement of the pulse trigger signal.

- Trigger level

The trigger can either be a pulse or a level; this is indicated by the configuration bit, DSI_TRIGGER_LEVEL in the SAR_SAMPLE_CTRL register. If it is a level, then the SAR starts new scans for as long as the trigger signal remains high. When the trigger signal is a pulse input, a positive edge detected on the trigger signal triggers a new scan.

- Transmission time

After the 'dsi_trigger' is raised, it takes some transmission time before the SAR ADC is told to start sampling. With different DSI_SYNC_TRIGGER and DSI_TRIGGER_LEVEL configuration, the transmission time is different; **Table 22-6** shows the maximum time. Two trigger pulse intervals should be longer than the transmission time, otherwise, the second trigger is ignored.

When the SAR is disabled (ENABLED=0), the trigger is ignored.

**Table 22-6. External trigger maximum time**

| Maximum External_TRIGGER transmission time | Bypass sync DSI_SYNC_TRIGGER=0 | Enable sync DSI_SYNC_TRIGGER=1 (by default) |
|---|---|---|
| Pulse trigger: DSI_TRIGGER_LEVEL=0 (by default) | 1 clk_sys+2 clk_sar | 3 clk_sys+2 clk_sar |
| Level Trigger: DSI_TRIGGER_LEVEL=1 | 2 clk_sar | 2 clk_sys+2 clk_sar |

**Table 22-7. Trigger signal requirement**

| Trigger specification | Requirement |
|---|---|
| Trigger Width (TW) | TW should be greater enough so that a trigger can be locked. If DSI_SYNC_TRIGGER=1, TW ≥ 2 clk_sys cycle. If DSI_SYNC_TRIGGER=0, TW ≥ 1 SAR clock cycle. |
| Trigger interval (TI) | Trigger interval should be longer than the transmission time (as specified in Table 22-6); otherwise, the second trigger pulse will be ignored. |

### 22.4.7 SAR ADC status

The current SAR status can be observed through the BUSY and CUR_CHAN fields in the SAR_STATUS register. The BUSY bit is high whenever the SAR is busy sampling or converting a channel; the CUR_CHAN [4:0] bits indicate the number of the current channel being sampled (channel 16 indicates the injection channel). SW_VREF_NEG bit indicates the current switch status that shorts NEG with VREF input.

CHAN_WORK_VALID in the CHAN_WORK_VALID register will be set if the WORK data that was sampled during the last scan is valid. When CHAN_RESULT_VALID is set in the CHAN_RESULT_VALID register, indicating that the RESULT data is valid, then the corresponding CHAN_WORK_VALID bit is cleared. The CUR_AVG_ACCU and CUR_AVG_CNT fields in the SAR_AVG_STAT register indicate the current averaging accumulator contents and the current sample counter value for averaging (counts down).

The SAR_MUX_SWITCH_STATUS register gives the current switch status of MUX_SWITCH0 register. These status registers help to debug SAR behavior.

### 22.4.8 Low-power mode

The current consumption of the SAR ADC can be divided into two parts: SAR ADC core and SARREF. There are several methods to reduce the power consumption of the SAR ADC core. The easiest way is to reduce the trigger frequency; that is, reduce the number of conversions per second. Another option is to use a lower resolution for channels that do not need high accuracy. This action shortens the conversion by up to four out of 18 cycles (for 8-bit resolution and minimum sample time). In addition, the SAR ADC offers the ICONT_LV[1:0] configuration bits, which control overall power of the SAR ADC. Maximum clock rates for each power setting should be observed.

**Table 22-8. ICONT_LV for low power consumption**

| ICONT_LV[1:0] | Relative power of SAR ADC core [%] | Maximum frequency [MHz] | Minimum sample time [cycles] | Maximum sample speed (at 12-bit) [ksps] |
|---|---|---|---|---|
| 0 | 100 | 18 | 4 | 1000 |
| 1 | 50 | 9 | 3 | 529 |
| 2 | 133 | 18 | 4 | 1000 |
| 3 | 25 | 4.5 | 2 | 281 |

**12-bit SAR ADC**

In addition to controlling the power of the SAR ADC core, the power consumed by VREF buffer (if used) can also be configured. Note that for full VDDA range (1.7 V to 5.5 V) operation without external bypass capacitor, the VREF buffer should be operated in 2x power mode. However, the maximum sample rate supported without external bypass capacitor remains at 100 ksps. For a 1-Msps sample rate, an external bypass capacitor and an 18-MHz clock are required. See **Table 22-9** for details.

**Table 22-9. SAR VREF power options**

| PWR_CTRL _VREF [1:0] | External bypass capacitor required | Relative VREF power [%] | Maximum frequency [MHz] | Minimum sample time [cycles] | Maximum sample speed (at 12-bit) [ksps] | VDDA range |
|---|---|---|---|---|---|---|
| 0 | Yes | 100 | 18 | 4 | 1000 | 1.7 V - 5.5 V |
| 0 | No | 100 | 1.6 | 2 | 100 | 2.7 V - 5.5 V |
| 2 | No | 200 | 1.6 | 2 | 100 | 1.7 V - 5.5 V |
| 1 or 3 | Invalid setting - Should not be used | | | | | |

Using an external VREF eliminates the need for the VREF buffer and bypass capacitor, which in turn reduces overall power consumption of the SAR ADC block.

## 22.4.9 System operation

After the SAR analog is enabled by setting the ENABLED bit (SAR_CTRL [31]), follow these steps to start ADC conversions with the SARSEQ:

1. Set SARMUX analog routing (pin/signal selection) via sequencer/firmware
2. Set the global SARSEQ conversion configurations
3. Configure each channel source (such as pin address)
4. Enable the channels
5. Set the trigger type
6. Set interrupt masks
7. Start the trigger source
8. Retrieve data after each end of conversion interrupt
9. Perform injection conversions if needed

Use registers to configure the SAR ADC; this is the most common usage. Detailed register bit definition is available in the EZ-PD™ PMG1-S3 MCU register TRM.

### 22.4.9.1 SARMUX analog routing

There are two ways to control the SARMUX analog routing: sequencer and firmware.

#### Sequencer control

It is essential that the appropriate hardware control bits in MUX_SWITCH_HW_CTRL register and the firmware control bits in MUX_SWITCH0 register are both set to ‘1’. Ensure that SWITCH_DISABLE=0; setting SWITCH_DISABLE disables sequencer control.

With sequencer control, the pin or internal signal a channel converts is specified by the combination of port and pin address. The PORT_ADDR bits are SAR_CHANx_CONFIG [6:4] and PIN_ADDR bits are SAR_CHANx_CONFIG [2:0]. **Table 22-10** shows the PORT_ADDR and PIN_ADDR setup with corresponding SARMUX selection.

**Table 22-10. PORT_ADDR and PIN_ADDR**

| PORT_ADDR | PIN_ADDR | Description |
|---|---|---|
| 0 | 0..7 | 8 dedicated pins of the SARMUX |
| 1 | X | sarbus0[a)] |
| 1 | X | sarbus1[a] |
| 7 | 0 | Temperature sensor |
| 7 | 2 | AMUXBUS-A |
| 7 | 3 | AMUXBUS-B |

a) sarbus0 and sarbus1 connect to the output of the CTBm block, which contains opamp0/1. See the **"Continuous Time Block mini (CTBm)"** on page 237 for more information. When PORT_ADDR=1, sarbus0 connects to positive terminal of SAR ADC regardless of the value of PIN_ADDR; sarbus1 can only connect to the negative terminal of SAR ADC when differential mode is enabled and PORT_ADDR=1.

For differential conversion, the negative terminal connection is dependent on the positive terminal connection, which is defined by PORT_ADDR and PIN_ADDR. By setting DIFFERENTIAL_EN, the channel will do a differential conversion on the even/odd pin pair specified by the pin address with PIN_ADDR [0] ignored. P.0/P.1, P.2/P.3, P.4/P.5, P.6/P.7 are valid differential pairs for sequencer control. More flexible analog can be implemented by firmware.

For single-ended conversions, NEG_SEL (SAR_CTRL [11:9]) is intended to decide which signal is connected to negative input. In differential mode, these bits are ignored. Negative input choice affects the input voltage range and effective resolution. See **"Negative input selection"** on page 211 for details. The options include: VSSA, VREF, or an external input from any of the eight pins with SARMUX connectivity. To connect negative input to VREF, an additional bit, SAR_HW_CTRL_NEGVREF (SAR_CTRL[13]) must be set, because the MUX_SWITCH_HW_CTRL register does not have that hardware control bit.

### Firmware control

By default, the SARMUX operates in firmware control. VPLUS (positive) and VMINUS (negative) inputs of SAR ADC can be controlled separately by setting the appropriate bits in SAR_MUX_SWITCH0 [29:0]. Clear appropriate bits in the hardware switch control register (SAR_MUX_SWITCH_HW_CTR[n]=0). Otherwise, hardware control method (sequencer) will control the SARMUX analog routing.

SAR_CTRL register bit SWITCH_DISABLE is used to disable SAR sequencer from enabling routing switches. Note that firmware control mode can always close switches independent of this bit value; however, it is recommended to set it to '1'.

NEG_SEL (SAR_CTRL [11:9]) decides which signal is connected to the negative terminal (vminus) of SAR ADC in single ended mode. In differential mode, these bits are ignored. In single-ended mode, when using sequencer control, you must set these bits. When using firmware control, NEG_SEL is ignored and SAR_MUX_SWITCH0 should be set to control the negative input. A special case is when SAR_MUX_SWITCH0 does not connect internal VREF to vminus; then, set NEG_SEL to '7'.

Negative input choice affects the input voltage range, SNR, and effective resolution. See **"Negative input selection"** on page 211 for details.

### 22.4.9.2 Global SARSEQ configuration

A number of conversion options that apply to all channels are configured globally. In several cases, the channel configuration has bits to choose what parts of the global configuration to use.

SAR_CTRL, SAR_SAMPLE_CTRL, SAR_SAMPLE01, SAR_SAMPLE23, SAR_RANGE_THES, and SAR_RANGE_COND are all global configuration registers. Typically, these configurations should not be modified while a scan is in progress. If configuration settings that are in use are changed, the results are undefined. Configuration settings that are not currently in use can be changed without affecting the ongoing scan.

**Table 22-11. Global configuration registers**

| Configurations | Control registers | Detailed reference |
|---|---|---|
| Reference selection | SAR_CTRL[6:4] | **"Reference options"** on page 220 |
| Signed/unsigned selection | SAR_SAMPLE_CTRL [3:2] | **"Result data format"** on page 210 |
| Data left/right alignment | SAR_SAMPLE_CTRL [1] | **"Result data format"** on page 210 |
| Negative input selection in single-ended mode | SAR_CTRL[11:9] | **"Negative input selection"** on page 211 |
| Resolution | SAR_SAMPLE_CTRL[0][a)] | **"Resolution"** on page 211 |
| Acquisition time | SAR_SAMPLE_TIME01 [25:0]<br>SAR_SAMPLE_TIME32 [25:0] | **"Acquisition time"** on page 212 |
| Averaging count | SAR_SAMPLE_CTRL[7:4] | **"Averaging"** on page 223 |
| Range detection | SAR_RANGE_THRES [31:0]<br>SAR_RANGE_COND [31:30] | **"Range detection"** on page 223 |

a) The alternate resolution should be enabled by the SAR_RESOLUTION bit in the SAR_CHAN_CONFIG register. If the alternate resolution is not enabled, the ADC operates at 12-bits of resolution, irrespective of the resolution set by the SAR_SAMPLE_CTRL register.

### 22.4.9.3 Channel configurations

Channel configuration includes:

- Differential or single-ended mode selection
- Global configuration selection: sample time, resolution, averaging enable
- DSI output enable

As a general rule, the channel configurations should only be updated between scans (same as global configurations). However, if a channel is not enabled for the ongoing scan, then the configuration for that channel can be changed freely without affecting the ongoing scan. If this rule is violated, the results are undefined. The channels that enable themselves are the only exception to this rule; enabled channels can be changed during the on-going scan, and it will be effective in the next scan. Changing the enabled channels may change the sample rate.

**Table 22-12. Channel configuration registers**

| Configurations | Registers | Detailed Reference |
|---|---|---|
| Single-ended/differential | SAR_CHANx_CONFIG [8] | **"Single-ended and differential mode"** on page 209 |
| Acquisition time selection | SAR_CHANx_CONFIG [13:12] | **"Acquisition time"** on page 212 |
| Resolution selection | SAR_CHANx_CONFIG [9] | **"Resolution"** on page 211 |
| Average enable | SAR_CHANx_CONFIG [10] | **"Averaging"** on page 223 |

SUB_RESOLUTION (SAR_SAMPLE_CTRL[0]) can choose which alternate resolution will be used, either 8-bit or 10 bit. Resolution (SAR_CHANx_CONFIG [9]) can determine whether default resolution 12-bit or alternate resolution is used. When averaging is enabled, the SUB_RESOLUTION is ignored; the resolution will be fixed to the maximum 12-bit.

**Table 22-13. Resolution**

| Average | SUB_RESOLUTION SAR_SAMPLE_CTRL[0] | Register mode resolution SAR_CHANx_CONFIG [9] | Channel resolution |
|---|---|---|---|
| OFF | 0 | 1 | 8-bit |
| OFF | 1 | 1 | 10-bit |
| OFF | 0 | 0 | 12-bit |
| OFF | 1 | 0 | 12-bit |
| ON | X | X | 12-bit |

### 22.4.9.4 Channel enables

A CHAN_EN register is available to individually enable each channel. All enabled channels are scanned when the next trigger happens. After a trigger, the channel enables can immediately be updated to prepare for the next scan. This action does not affect the ongoing scan. Note that this is an exception to the rule; all other configurations (global or channel) should not be changed while a scan is in progress.

### 22.4.9.5 Interrupt masks

There are six interrupt sources; all have an interrupt mask:

- End-of-scan interrupt
- Overflow interrupt
- Collision interrupt
- Injection end-of-conversion interrupt
- Range detection interrupt
- Saturate detection interrupt

Each interrupt has an interrupt request register (INTR, SATURATE_INTR, RANGE_INTR), a software interrupt set register (INTR_SET, SATURATE_INTR_SET,RANGE_INTR_SET), an interrupt mask register (INTR_MASK, SATURATE_INTR_MASK, RANGE_INTR_MASK), and an interrupt re-quest masked result register (INTR_MASKED, SATURATE_INTR_MASKED, RANGE_INTR_MASKED). An interrupt cause register is also added to have an overview of all the currently pending SAR interrupts and allows the ISR to determine the interrupt cause by just reading this register.

See **“Interrupt”** on page 225 for details.

### 22.4.9.6 Trigger

The three ways to start an A/D conversion are:

- Firmware trigger: SAR_START_CTRL [0]
- External trigger: dsi_trigger
- Continuous trigger: SAR_SAMPLE_CTRL [16]

See **“Trigger”** on page 227 for details.

### 22.4.9.7 Retrieve data after each interrupt

Make sure you read the data from the result register after each scan; otherwise, the data may change because of the next scan's configuration. The 16-bit data registers are used to implement double buffering for up to eight channels (injection channel do not have double buffer). Double buffering means that there is one working register and one result register for each channel.

Data is written to the working register immediately after sampling this channel. It is then copied to the result register from the working register after all enabled channels in this scan have been sampled.

The CHAN_WORK_VALID bit is set after the corresponding WORK data is valid, that is, it was already sampled during the current scan. Corresponding CHAN_RESULT_VALID is set after completed scan. When CHAN_RESULT_VALID is set, the corresponding CHAN_WORK_VALID bit is cleared. For firmware convenience, bit [31] in SAR_CHAN_WORK register is the mirror bit of the corresponding bit in SAR_CHAN_WORK_VALID register. Bit[29], bit [30], and bit[31] in SAR_CHAN_RESULT are the mirror bits of the corresponding bit in SAR_SATURATE_INTR, SAR_RANGE_INTR, and SAR_CHAN_RESULT_VALID registers. Note that the interrupt bits mirrored here are the aw (unmasked) interrupt bits. It helps firmware to check if the data is valid by just reading the data register.

### 22.4.9.8 Injection conversions

Injection channel can be triggered by setting the start bit INJ_START_EN (INJ_CHAN_CONFIG [31]). To prevent the collision of regular automatic scan, it is recommended to enable tailgating by setting INJ_CHAN_CONFIG [30]. When it is enabled, INJ_START_EN will enable the injection channel to be scanned at the end of next scan of regular channels. See **“Injection channel”** on page 224 for details.

## 22.4.10 Temperature sensor configuration

One on-chip temperature sensor is available for temperature sensing and temperature-based calibration. Differential conversions are not available for temperature sensors (conversion result is undefined). Therefore, always use it in single-ended mode. The reference is from internal 1.2 V.

A pin or signal can be routed to the SAR ADC in three ways. **Table 22-14** lists the methods to route temperature sensors to SAR ADC. Setting the MUX_FW_TEMP_VPLUS bit (SAR_MUX_SWITCH0[17]) can enable the temperature sensor and connect its output to VPLUS of SAR ADC; clearing this bit disables temperature sensor by cutting its bias current.

**Table 22-14. Route temperature to SAR ADC**

| Control methods | Setup |
|---|---|
| Sequencer | DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8])<br>VREF_SEL = 0 (SAR_CTRL[6:4])<br>PORT_ADDR = 7 (SAR_CHANx_CONFIG[6:4])<br>PIN_ADDR = 0 (SAR_CHANx_CONFIG[2:0])<br>SWITCH_DISABLE = 0 (SAR_CTRL[30])<br>SAR_MUX_SWITCH0[16] = 1<br>SAR_MUX_SWITCH0[17] = 1<br>SAR_MUX_SWITCH_HW_CTRL[16]= 1<br>SAR_MUX_SWITCH_HW_CTRL[17]= 1<br>NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a)] |
| Firmware | DIFFERENTIAL_EN = 0 (SAR_CHANx_CONFIG[8])<br>VREF_SEL = 0 (SAR_CTRL[6:4])<br>SWITCH_DISABLE = 1 (SAR_CTRL[30])<br>SAR_MUX_SWITCH0[16] = 1<br>SAR_MUX_SWITCH0[17] = 1<br>SAR_MUX_SWITCH_HW_CTRL[16]= 0<br>SAR_MUX_SWITCH_HW_CTRL[17]= 0<br>NEG_SEL = 0 (SAR_CTRL [11:9]) override to 0[a] |

a) For temperature sensor, override NEL_SEG (SAR_CTRL [11:9]) to '0'.

## 22.5 Registers

**Table 22-15. SAR ADC registers**

| Name | Offset | Qty. | Width | Description |
|---|---|---|---|---|
| SAR_CTRL | 0x0000 | 1 | 32 | Global configuration register<br>Analog control register |
| SAR_SAMPLE_CTRL | 0x0004 | 1 | 32 | Global configuration register<br>Sample control register |
| SAR_SAMPLE_TIME01 | 0x0010 | 1 | 32 | Global configuration register<br>Sample time specification ST0 and ST1 |
| SAR_SAMPLE_TIME23 | 0x0014 | 1 | 32 | Global configuration register<br>Sample time specification ST2 and ST3 |
| SAR_RANGE_THRES | 0x0018 | 1 | 32 | Global range detect threshold register |
| SAR_RANGE_COND | 0x001C | 1 | 32 | Global range detect mode register |
| SAR_CHAN_EN | 0x0020 | 1 | 32 | Enable bits for the channels |
| SAR_START_CTRL | 0x0024 | 1 | 32 | Start control register (firmware trigger) |
| SAR_CHAN_CONFIG | 0x0080 | 8 | 32 | Channel configuration register |
| SAR_CHAN_WORK | 0x0100 | 8 | 32 | Channel working data register |
| SAR_CHAN_RESULT | 0x0180 | 8 | 32 | Channel result data register |
| SAR_CHAN_WORK_VALID | 0x0200 | 1 | 32 | Channel working data register valid bits |
| SAR_CHAN_RESULT_VALID | 0x0204 | 1 | 32 | Channel result data register valid bits |
| SAR_STATUS | 0x0208 | 1 | 32 | Current status of internal SAR registers (for debug) |
| SAR_AVG_STAT | 0x020C | 1 | 32 | Current averaging status (for debug) |

**Table 22-15. SAR ADC registers** (continued)

| Name | Offset | Qty. | Width | Description |
|---|---|---|---|---|
| SAR_INTR | 0x0210 | 1 | 32 | Interrupt request register |
| SAR_INTR_SET | 0x0214 | 1 | 32 | Interrupt set request register |
| SAR_INTR_MASK | 0x0218 | 1 | 32 | Interrupt mask register |
| SAR_INTR_MASKED | 0x021C | 1 | 32 | Interrupt masked request register: If the value is not zero, then the SAR interrupt signal to the NVIC is high. When read, this register reflects a bit-wise AND between the interrupt request and mask registers |
| SAR_SATURATE_INTR | 0x0220 | 1 | 32 | Saturate interrupt request register |
| SAR_SATURATE_INTR_SET | 0x0224 | 1 | 32 | Saturate interrupt set request register |
| SAR_SATURATE_INTR_MASK | 0x0228 | 1 | 32 | Saturate interrupt mask register |
| SAR_SATURATE_INTR_MASKED | 0x022C | 1 | 32 | Saturate interrupt masked request register |
| SAR_RANGE_INTR | 0x0230 | 1 | 32 | Range detect interrupt request register |
| SAR_RANGE_INTR_SET | 0x0234 | 1 | 32 | Range detect interrupt set request register |
| SAR_RANGE_INTR_MASK | 0x0238 | 1 | 32 | Range detect interrupt mask register |
| SAR_RANGE_INTR_MASKED | 0x023C | 1 | 32 | Range interrupt masked request register |
| SASR_INTR_CAUSE | 0x0240 | 1 | 32 | Interrupt cause register |
| SAR_INJ_CHAN_CONFIG | 0x0280 | 1 | 32 | Injection channel configuration register |
| SAR_INJ_RESULT | 0x0290 | 1 | 32 | Injection channel result register |
| SAR_MUX_SWITCH0 | 0x0300 | 1 | 32 | SARMUX firmware switch controls |
| SAR_MUX_SWITCH_CLEAR0 | 0x0304 | 1 | 32 | SARMUX firmware switch control clear |
| SAR_MUX_SWITCH_HW_CTRL | 0x0340 | 1 | 32 | SARMUX switch hardware control |
| SAR_MUX_SWITCH_STATUS | 0x0348 | 1 | 32 | SARMUX switch status |
| SAR_PUMP_CTRL | 0x0380 | 1 | 32 | Switch pump control |

# 23 Continuous Time Block mini (CTBm)

The Continuous Time Block mini (CTBm) provides discrete operational amplifiers (opamps) inside the chip for use in continuous-time signal chains. Each CTBm block includes a switch matrix for input/output configuration, two identical opamps, which are also configurable as two comparators, a charge pump inside each opamp, and a digital interface for comparator output routing, switch controls, and interrupts. EZ-PD™ PMG1-S3 MCU have one CTBm block, which can be operational in Deep-Sleep power mode.

## 23.1 Features

The opamps in the EZ-PD™ PMG1-S3 MCU CTBm block have the following features:

- Discrete, high-performance, and highly configurable on-chip amplifiers
- Programmable power, bandwidth, compensation, and output drive strength
- 1-mA or 10-mA selectable output current drive capability
- 6-MHz gain bandwidth for 20-pF load
- Less than 1-mV offset with trim
- Support for opamp follower mode
- Comparator mode with optional 10-mV hysteresis
- Buffer/pre-amplifier for SAR inputs
- Support in Deep-Sleep device power mode

## 23.2 Block diagram

**Figure 23-1** shows the block diagram for the CTBm block available in EZ-PD™ PMG1-S3 MCU devices.

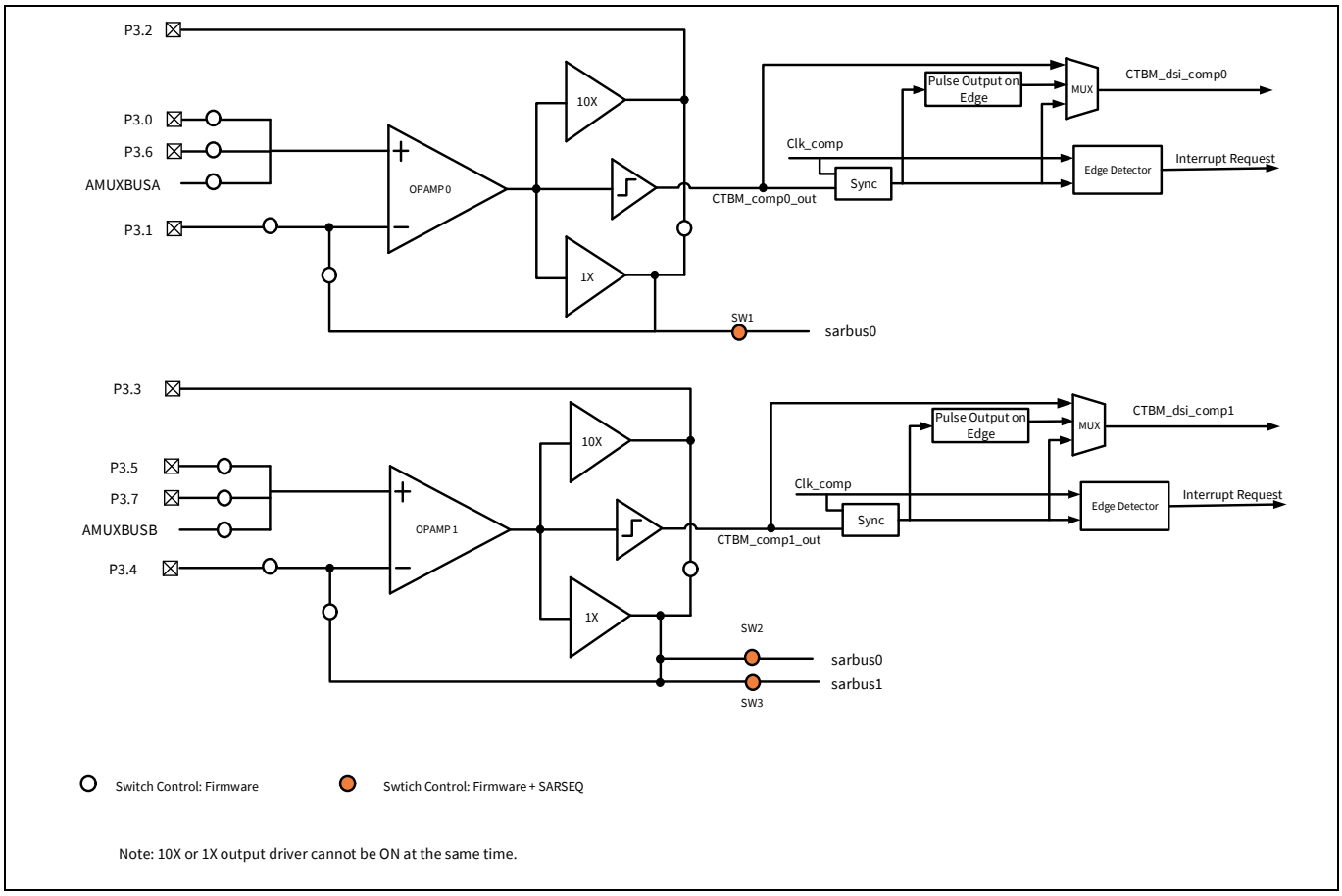



**Figure 23-1. CTBm block diagram**

## 23.3 How it works

As the block diagram shows, CTBm has two identical opamps. Each opamp has one input and three output stages, all of which share a common input stage, as shown in **Figure 23-1**; only one of them can be selected at a time. The output stage can be operated as Class-A(1X), Class-AB(10X), or comparator. The other configurable features are power and speed, compensation, and switch routing control.

To use the CTBm block, the first step is to set up external components (such as resistors), if required. Then, enable the block by setting the CTB_CTRL [31] bit. To have almost rail-to-rail input range and minimal distortion common mode input, there is one charge pump inside each opamp. The charge pump can be enabled by setting the CTBM_OA_RES0_CTRL [11] bit for opamp0 and CTBM_OA_RES1_CTRL [11] bit for opamp1.

After enabling the opamps and charge pumps, follow these steps to set up the amplifier:

1. Configure power mode
2. Configure output strength
3. Configure compensation
4. Configure input switch
5. Configure output switch, especially when opamp output needs to be connected to SAR ADC

Follow these steps to set up a comparator:

1. Configure the power mode
2. Configure the input switch
3. Configure the comparator output circuitry, as required - interrupt generation, output, and so on
4. Configure hysteresis and enable the comparator

### 23.3.1 Power mode configuration

The opamp can operate in three power modes – low, medium, and high. CTBm adjusts the power consumed by adjusting the reference currents coming into the opamp. Power modes are configured using the PWR_MODE bits [1:0] in CTB-M_OA_RESx_CTRL. The slew rate and gain bandwidth are maximum in high-power mode and minimum in low-power mode. Note that power mode configuration also affects the maximum output drive capability $_{(IOUT)}$ in 1X mode. See **Table 23-1** for details.

### 23.3.2 Output strength configuration

The output driver of each opamp can be configured to internal driver (Class A/1X driver) or external driver (Class AB/10X driver). 1X and 10X drivers are mutually exclusive – they cannot be active at the same time. 1X output driver is suited to drive smaller on-chip capacitive and resistive loads at higher speeds. The 10X output driver is useful for driving large off-chip capacitive and resistive loads. The 1X driver output is routed to sarbus 0/1 and 10X driver output is routed to an external pin. Each driver mode has a low, medium, or high power mode, as shown in **Table 23-1**.

**Table 23-1. Output driver versus power mode**

| Power mode IOUT drive capability | CTBM_OA_RESx_CTRL[1:0] | | | |
|---|---|---|---|---|
| | 00 (Disable) | 01 (Low) | 10 (Medium) | 11 (High) |
| External Driver (10X) | Off | 10 mA | 10 mA | 10 mA |
| Internal Driver (1X) | Off | 100 µA | 400 µA | 1 mA |

The CTBM_OA_RESx_CTRL[2] bit is used to select between the 10X and 1X output capability (0: 1X, 1: 10X). If the output of the opamp is connected to the SAR ADC, it is recommended to choose the 1X output driver. If the output of the opamp is connected to an external pin, then, choose the 10X output driver. In special instances, to connect the output to an external pin with 1X output driver or an internal load (for example, SAR ADC) with 10X output driver, set CTBM_OAx_SW [21] to '1'. However, Infineon does not guarantee performance in this case.

Table 23-2 summarizes the bits used to configure the opamp output drive strength and power modes.

**Table 23-2. Output strength and power mode configuration in CTBm registers**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| CTBM_CTB_CTRL[31] | ENABLE | CTBM power mode selection<br>0: CTBM is disabled<br>1: CTBM is enabled |
| CTBM_OA_RES0_CTRL [11] | OA0_PUMP_EN | Opamp0 pump enable bit<br>0: Opamp0 pump is disabled<br>1: Opamp0 pump is enabled |
| CTBM_OA_RES1_CTRL [11] | OA1_PUMP_EN | Opamp1 pump enable bit<br>0: Opamp1 pump is disabled<br>1: Opamp1 pump is enabled |
| CTBM_OA_RES0_CTRL [1:0] | OA0_PWR_MODE | Opamp0 power mode select bits<br>00: Opamp0 is OFF<br>01: Opamp0 is in low power mode<br>10: Opamp0 is in medium power mode<br>11: Opamp0 is in high power mode |
| CTBM_OA_RES1_CTRL [1:0] | OA1_PWR_MODE | Opamp1 power mode select bits<br>00: Opamp1 is OFF<br>01: Opamp1 is in low power mode<br>10: Opamp1 is in medium power mode<br>11: Opamp1 is in high power mode |
| CTBM_OA_RES0_CTRL [2] | OA0_DRIVE_STR_SEL | Opamp0 output drive strength select bits<br>0: Opamp0 output drive strength is 1X<br>1: Opamp0 output drive strength is 10X |
| CTBM_OA_RES1_CTRL [2] | OA1_DRIVE_STR_SEL | Opamp1 output drive strength select bits<br>0: Opamp1 output drive strength is 1X<br>1: Opamp1 output drive strength is 10X |

### 23.3.3 Compensation

Each opamp also has a programmable compensation capacitor block, which allows optimizing the stability of the opamp performance based on output load. The compensation of each opamp is controlled by the respective CTBM_OAx_COMP_TRIM register, as explained in Table 23-3.

**Table 23-3. Opampx (Opamp0 or Opamp1) compensation bits in CTBm**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| CTBM_OAx_COMP_TRIM[1:0] | OAx_COMP_TRIM | Opampx compensation trim bits<br>00: No compensation<br>01: Minimum compensation, high speed, and low stability<br>10: Medium compensation, balanced speed, and stability<br>11: Maximum compensation, low speed, and high stability |

### 23.3.4 Switch control

The CTBm has many switches to configure the opamp input and output. Most of them are controlled by configuring CTBm registers (CTBM_OA0_SW, CTBM_OA1_SW), except three switches, which are used to connect the output of opamps to SAR ADC through sarbus0 and sarbus1. They must be controlled by SAR ADC registers, and CTBm registers.

Switches can be closed by setting the corresponding bit in register CTBM_OAx_SW; clearing them will cause the corresponding switches to open. To open the switch, write '1' to CTBM_OAx_SW_CLEAR, which clears the corresponding bit in CTB-M_OAx_SW. See PMG1 Family EZ-PD™ PMG1-S3 MCU registers TRM for details on the switches and the connections they enable.

#### 23.3.4.1 Input configuration

Positive and negative input to the operational amplifier can be selected from several options through analog switches. These switches serve to connect the opamp inputs from the external pins or AMUX buses, or to form a local feedback loop (for buffer function). Each opamp has a switch connecting to one of the two AMUXBUS line: Opamp0 connects to AMUXBUS-A and Opamp1 connects to AMUXBUS-B.

*Note: Only one switch should be closed for the positive and negative input paths; otherwise, different input source may short together.*

- Positive input: Both opamp0 and opamp1 have three positive input options through analog switches: two external pins and one AMUXBUS line. See **Table 23-4** for details.

**Table 23-4. Positive input selection**

| | Positive input | Switch control bit | Description |
|---|---|---|---|
| Opamp0 | AMUXBUSA | CTBM_OA0_SW [0] | 0: open 1: close switch |
| | P3.0 | CTBM_OA0_SW [2] | 0: open 1: close switch |
| | P3.6 | CTBM_OA0_SW [3] | 0: open 1: close switch |
| Opamp1 | AMUXBUSB | CTBM_OA1_SW [0] | 0: open 1: close switch |
| | P3. 5 | CTBM_OA1_SW [1] | 0: open 1: close switch |
| | P3.7 | CTBM_OA1_SW [4] | 0: open 1: close switch |

- Negative input: Both opamp0 and opamp1 have two negative input options through analog switches: one external pin or output feedback, which is controlled by the CTBM_OAx_SW register. **Table 23-5** shows the control bits.

**Table 23-5. Negative input selection**

| | Negative input | Switch control bit | Description |
|---|---|---|---|
| Opamp0 | P3.1 | CTBM_OA0_SW [8] | 0: open 1: close switch |
| | Opamp0 output feedback through 1X output driver | CTBM_OA0_SW [14] | 0: open 1: close switch |
| Opamp1 | P3.4 | CTBM_OA1_SW [8] | 0: open 1: close switch |
| | Opamp1 output feedback through 1X output driver | CTBM_OA1_SW [14] | 0: open 1: close switch |

### 23.3.4.2 Output configuration

Each opamp's output is connected directly to a fixed pin; no additional setup is needed. Optionally, it can be connected to sarbus0 or sarbus1 through three switches (SW1/2/3). The opamp0 output can be connected to sarbus0 and opamp1 can be connected to sarbus0 or sarbus1. sarbus0 and sarbus1 are intended to connect opamp output to the SAR ADC input mux. The three output routing switches to sarbus are controlled by SAR ADC registers, and CTBm register together; the other switches can be controlled only by CTBm register.

The following truth tables (**Table 23-6**, **Table 23-7**, and **Table 23-8**) show the control logic of the three switches. PORT_ADDR, PIN_ADDR, and DIFFERENTIAL_EN are from SAR_CHANx_CONFIG [6:4], SAR_CHANx_CONFIG [2:0], and SAR_CHANx_CONFIG [2:0], respectively. Either PORT_ADDR =0 or PIN_ADDR = 0 will set SW[n]=0. CTBM_SW_HW_C-TRL bit [2] or [3] should be set when using the SAR register to control switches. CTBM_OAx_SW[18]/[19] can mask the other control bits – if CTBM_OAx_SW[18]/[19] = 0, SW[n] = 0.

The CTBM__SW_STATUS [30:28] register gives the current switch status of SW1/2/3.

**Table 23-6. Truth table of SW1 control logic**

| PORT_ADDR | PIN_ADDR | CTBM_SW_HW_CTRL[2] | CTBM_OA0_SW[18] | SW1 |
|---|---|---|---|---|
| X | X | X | 0 | 0 |
| X | 0 | 1 | 1 | 0 |
| 0 | X | 1 | 1 | 0 |
| X | X | X | 1 | 1 |
| X | X | 0 | 1 | 1 |
| 1 | 2 | X | 1 | 1 |

**Table 23-7. Truth table of SW2 control logic**

| DIFFERENTIAL_EN | PORT_ADDR | PIN_ADDR | CTBM_SW_HW_CT RL[3] | CTBM_OA0_SW[18] | SW2 |
|---|---|---|---|---|---|
| X | X | X | X | 0 | 0 |
| X | X | 0 | 1 | 1 | 0 |
| X | 0 | X | 1 | 1 | 0 |
| 1 | X | X | X | 1 | 0 |
| X | X | X | 0 | 1 | 1 |
| X | X | X | X | 1 | 1 |
| 0 | 1 | 3 | X | 1 | 1 |

**Table 23-8. Truth table of SW3 control logic**

| DIFFERENTIAL_EN | PORT_ADDR | PIN_ADDR | CTBM_SW_HW_CT RL[3] | CTBM_OA0_SW[18] | SW3 |
|---|---|---|---|---|---|
| X | X | X | X | 0 | 0 |
| X | X | 0 | 1 | 1 | 0 |
| X | 0 | X | 1 | 1 | 0 |
| 0 | X | X | X | 1 | 0 |
| X | X | X | 0 | 1 | 1 |
| X | X | X | X | 1 | 1 |
| 1 | 1 | 2 | X | 1 | 1 |

### 23.3.4.3 Comparator mode

Each opamp can be configured as a comparator by setting the respective CTBM_OA_RESx_CTRL[4] bit. Note that enabling the comparator completely disables the compensation capacitors and shuts down the Class A (1X) and Class AB (10X) output drivers. The comparator has the following features:

- Optional 10-mV input hysteresis
- Configurable power/speed
- Optional comparator output synchronization
- Offset trimmed to less than 1 mV
- Configurable edge detection (rising/falling/both/disable)

### 23.3.4.4 Comparator configuration

The hysteresis of 10 mV ±5 percent can be enabled in one direction (low to high). Input hysteresis can be enabled by setting CTBM_OA_RESx_CTRL[5]. The two comparators also have three power modes: low, medium, and high, controlled by setting CTBM_OA_RESx_CTRL [1:0]. Power modes differ in response time and power consumption; power consumption is maximum in fast mode and minimum in ultra-low-power mode. Exact specifications for power consumption and response time are provided in the datasheet.

The synchronization of the comparator output with the system AHB clock can be configured in CTBM_OA_RESx_CTRL[6].

The output state of comparator0 and comparator1 are stored in CTBM_COMP_STAT[0] and CTBM_COMP_STAT[16], respectively.

**Table 23-9** summarizes various bits used to configure the comparator mode in the CTBm block.

**Table 23-9. Comparator mode and configuration register settings**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| CTBM_OA_RESyx_CTRL[4] | OAx_COMP_EN | Opampx comparator enable bit<br>0: Comparator mode is disabled in opampx<br>1: Comparator mode is enabled in opampx |
| CTBM_OA_RESx_CTRL[5] | OAx_HYST_EN | Opampx Comparator hysteresis enable bit<br>0: Hysteresis is disabled in opampx<br>1: Hysteresis is enabled in opampx |
| CTBM_OA_RESx_CTRL[6] | OAx_BYPASS_DSI_SYNC | Opampx bypass comparator output synchronization for DSI (trigger) output<br>0: Synchronize (level or pulse)<br>1: Bypass |
| CTBM_OA_RESx_CTRL[7] | OAx_DSI_LEVEL | Opampx comparator DSI (trigger) output synchronization level<br>0: Pulse<br>1: Level |

### 23.3.4.5 Comparator interrupt

The comparator output is connected to an edge detector block, which is used to detect the edge (disable/rising/falling/both) that generates interrupt. It can be configured by the CTBM_OA_RESx_CTRL[9:8] bits.

Each comparator has a separate IRQ. CTBM_INTR [0] is for comparator0 IRQ, CTBM_INTR [1] is for comparator1 IRQ. Though each comparator have different IRQ bits, they all share a single CTBM ISR mapped in the CPU NVIC. See the **"Interrupts"** on page 38 for details. You can check which comparator(s) triggered the ISR by polling the CTBMx_INTR bits.

Each interrupt has an interrupt mask bit in the CTBM_INTR_MASK register. By setting the interrupt mask low, the corresponding interrupt source is ignored. The CTBm comparator interrupt to the NVIC will be raised if logic AND of the interrupt flags in CTBM_INTR registers and the corresponding interrupt masks in CTBM_INTR_MASK register is 1.

Writing a '1' to the CTBM_INTR bit [1:0] can clear corresponding interrupt.

For firmware convenience, the intersection (logic AND) of the interrupt flags and the interrupt masks is also made available in the CTBM_INTR_MASKED register.

For verification and debug purposes, a set bit is provided for each interrupt in the CTBM_INTR_SET register. This bit allows the firmware to raise the interrupt without a real comparator switch event.

### 23.3.4.6 Deep-Sleep mode operation

In Deep-Sleep mode, the block that provides the bias current, reference voltage, and IMO clock is turned off. As a result, the CTBm functionality, which relies on the bias current and IMO clock for its operation is not available. See the **"Power modes"** on page 95 for details on various power modes and blocks available in each mode. To support the functionality of the CTBm during deep sleep, an alternate bias current is generated by a special block called Deep-Sleep Amplifier Bias (DSAB) block. This current allows the opamps in the CTBm to be functional in Deep-Sleep mode.

**Figure 23-2** shows the architecture of the DSAB block. This block receives the Active mode bias current as input. It outputs the bias current that is fed to the opamp bias circuitry. In Active mode, the DSAB block acts similar to a pass-through block and routes the bias current from the input to the output. In Deep-Sleep mode, if enabled, the DSAB generates the alternate bias current, attenuates the output to a user-selected value, and provides the bias current for the CTBm at its output. If the DSAB block is disabled, the output is always connected to the input bias current and the alternate bias current is not generated during deep sleep. The opamps will not be functional in Deep-Sleep mode, if the DSAB block is disabled. The ENABLED bit [31] of the PASS_DSAB_DSAB_CTRL register enables/disables the block; the CURRENT_SEL bits [5:0] selects the output bias current value. The value selected is CURRENT_SEL × 0.075 µA (±5 percent). The SEL_OUT bit[8] is used to control the selection between the two bias currents, which can be routed to the CTBm bias. **Table 23-10** summarizes the bit configuration settings of the PASS_DSAB_DSAB_CTRL register.

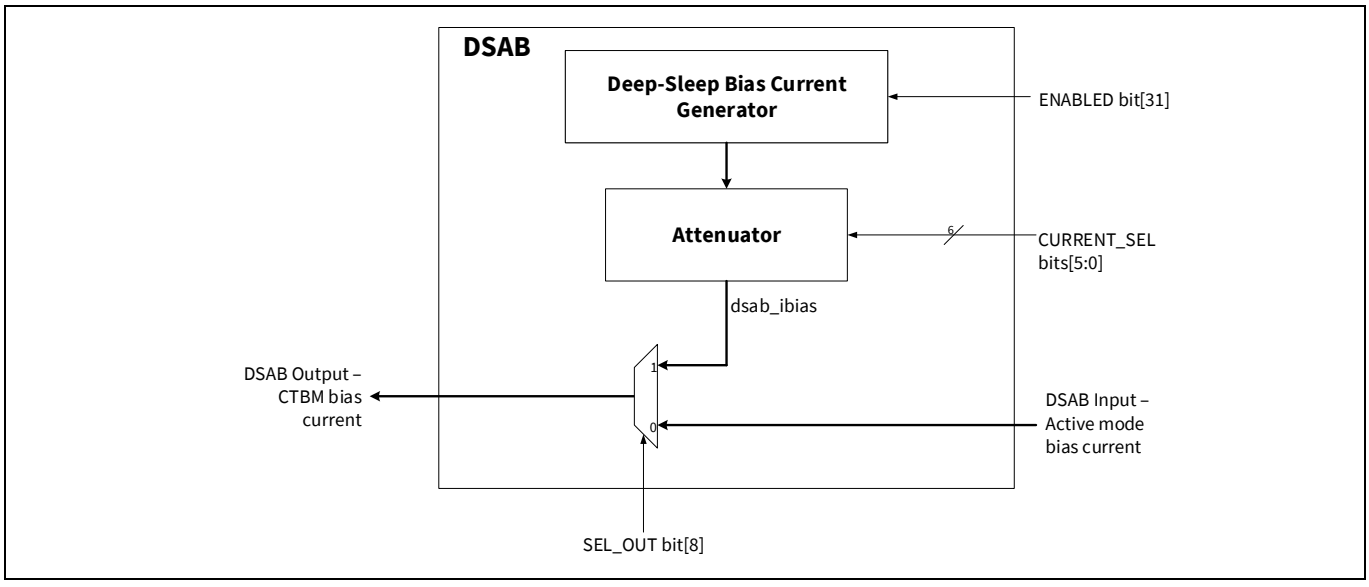



**Figure 23-2. Deep-Sleep amplifier bias block diagram**

### Continuous Time Block mini (CTBm)

This feature is useful in designs that require the opamp-based circuitry to remain active in low-power modes, such as Deep-Sleep, to save power. For instance, in a battery-operated system (such as a heart-rate monitor) that requires always-on opamps, substantial power savings can be achieved if the rest of the chip can go into Deep-Sleep mode and only wake up as needed. Note that the bias current provided by the DSAB block does not meet the accuracy and stability of the Active mode bias current. In addition, the DSAB does not generate an alternate clock. As a result, none of the switch or opamp-related charge pumps are activated. Consequently, the highest input common-mode voltage of the opamps is limited to approximately $V_{DDA}$ – 1.3 V. In addition, because of the unavailability of switch pumps (required for analog switches when operating below 3.3 V), the on-resistance of the analog switches increase beyond normal specification as the supply voltage drops below 3.3 V. It is justifiable for the analog switches to have higher on-resistance as long as the signal speeds are low. Thus, VDDA can go as low as ~2.8 V before the analog switches become too resistive. It will eventually set the lowest-possible supply voltage. However, it is recommended to use VDDA of 3.3 V or greater when using opamps in Deep-Sleep mode. See the **device datasheet** for opamp specifications during Deep-Sleep mode.

To enable the opamps in Deep-Sleep mode, set the DEEPS-LEEP_ON bit [30] of the CTBM_CTB_CTRL register. This bit enables both the opamps of the CTBM during deep sleep. The deep-sleep operation of the CTBm also requires the DSAB block to be enabled.

**Table 23-10. DSAB and CTBM Deep-Sleep configuration register settings**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| PASS_DSAB_DSAB_CTRL [5:0] | CURRENT_SEL | Current selection for the dsab_ibias; dsab_ibias = CURRENT_SEL × 0.075 µA (±5%) |
| PASS_DSAB_DSAB_CTRL [8] | SEL_OUT | CTBm bias current selection<br>0: Bypass DSAB and use active mode bias current<br>1: Use dsab_ibias as the CTBm bias current |
| PASS_DSAB_DSAB_CTRL [31] | ENABLED | Enable/disable DSAB bias generator<br>0: DSAB block is disabled and the CTBm bias current is connected to the Active mode bias current<br>1: DSAB block is enabled and the CTBm bias current is controlled by the SEL_OUT signal |
| CTBMx_CTBM_CTB_CTRL [30] | DEEPSLEEP_ON | Enable/disable the CTBMx functionality in Deep-Sleep mode<br>0: Enabled<br>1: Disabled |

## 23.4 Registers

**Table 23-11. CTBm registers**

| Name | Description |
|---|---|
| CTBM_CTRL | Global CTBm block enable |
| CTBM_OA_RES0_CTRL | Opamp0 control register |
| CTBM_OA_RES1_CTRL | Opamp1 control register |
| CTBM_COMP_STAT | Comparator status |
| CTBM_INTR | Interrupt request register |
| CTBM_INTR_SET | Interrupt request set register |
| CTBM_INTR_MASK | Interrupt request mask |
| CTBM_INTR_MASKED | Interrupt request masked |
| CTBM_OA0_SW | Opamp0 switch control |
| CTBM_OA0_SW_CLEAR | Opamp0 switch control clear |
| CTBM_OA1_SW | Opamp1 switch control |
| CTBM_OA1_SW_CLEAR | Opamp1 switch control clear |
| CTBM_SW_HW_CTRL | CTBm hardware control enable |
| CTBM_SW_STATUS | CTBm bus switch control status |
| CTBM_OA0_OFFSET_TRIM | Opamp0 trim control |
| CTBM_OA0_SLOPE_OFFSET_TRIM | Opamp0 trim control |
| CTBM_OA0_COMP_TRIM | Opamp0 trim control |
| CTBM_OA1_OFFSET_TRIM | Opamp1 trim control |
| CTBM_OA1_SLOPE_OFFSET_TRIM | Opamp1 trim control |
| CTBM_OA1_COMP_TRIM | Opamp1 trim control |
| PASS_DSAB_DSAB_CTRL | DSAB control register |
| PASS_DSAB_TRIM | IBIAS trim register |

# 24 Low-power comparator

EZ-PD™ PMG1-S3 MCU devices have two low-power comparators. These comparators can perform fast analog signal comparison in all system power modes. Refer to the **"Power modes"** on page 95 for details on various device power modes. The positive and negative inputs can be connected to dedicated GPIO pins or to AMUXBUS-A/AMUXBUS-B. The comparator output can be read by the CPU through a status register, used as an interrupt or wakeup source or routed to a GPIO.

## 24.1 Features

EZ-PD™ PMG1-S3 MCU comparators have the following features:

- Configurable positive and negative inputs
- Programmable power and speed
- Ultra low-power mode support (<4 µA)
- Optional 10-mV input hysteresis
- Low-input offset voltage (<4 mV after trim)
- Wakeup source in Deep-Sleep mode

## 24.2 Block diagram

**Figure 24-1** shows the block diagram for the low-power comparator.

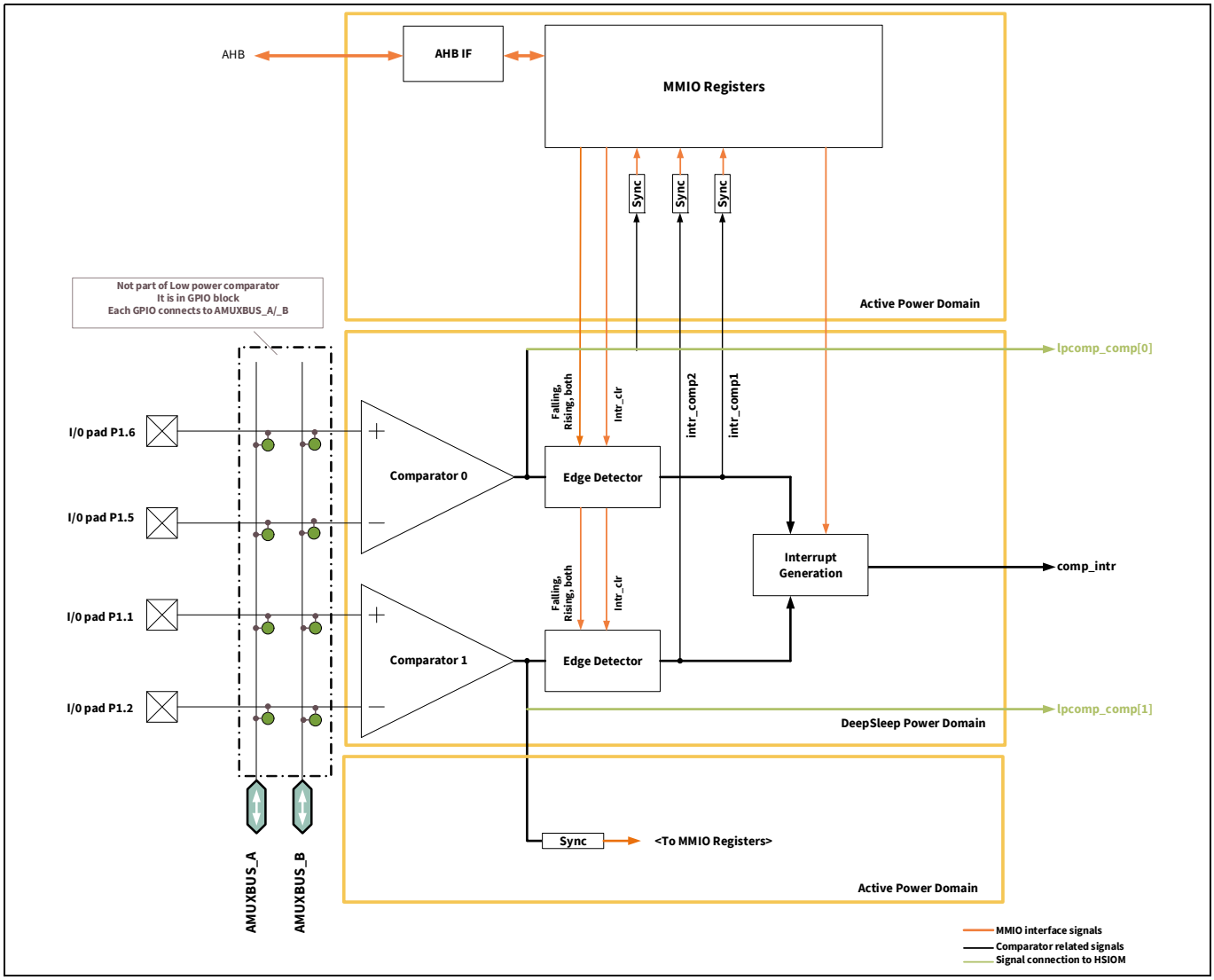



**Figure 24-1. Low-power comparator block diagram**

## 24.3 How it works

The following sections describe the operation of the EZ-PD™ PMG1-S3 MCU low-power comparator, including input configuration, power and speed mode, output and interrupt configuration, hysteresis, wake up from low-power modes, comparator clock, and offset trim.

### 24.3.1 Input configuration

Inputs to the comparators can be as follows:

- Both positive and negative inputs from dedicated input pins.
- Both positive and negative inputs from any pin through AMUXBUS (not available in Deep-Sleep mode).
- One input from an external pin and another input from an internally-generated signal. Both inputs can be connected to either positive or negative inputs of the comparator. The internally-generated signal is connected to the comparator input through the analog AMUXBUS.
- Both positive and negative inputs from internally-generated signals. The internally-generated signals are connected to the comparator input through AMUXBUS-A/AMUXBUS-B.

From **Figure 24-1**, note that P0.0 and P0.1 connect to positive and negative inputs of Comparator 0; P0.2 and P0.3 connect to the inputs of Comparator 1. Also, note that the AMUXBUS nets do not have a direct connection to the comparator inputs. Therefore, the comparator connection is routed to the AMUXBUS nets through the corresponding input pin. These input pins will not be available for other purposes when using AMUXBUS for comparator connections.

They should be left open in designs that use AMUXBUS for comparator input connection. Note that AMUXBUS connections are not available in Deep-Sleep mode. If Deep-Sleep operation is required, the low-power comparator must be connected to the dedicated pins. This restriction also includes routing of any internally-generated signal, which uses the AMUXBUS for the connection. See the **"I/O system"** on page 77 for more details on connecting the GPIO to AMUXBUS A/B or setting up the GPIO for comparator input.

### 24.3.2 Output and interrupt configuration

The output of Comparator0 and Comparator1 are available in the OUT1 bit [6] and OUT2 bit [14], respectively, in the LPCOMP_CONFIG register (**Table 24-1**). The comparator outputs are synchronized to SYSCLK before latching them to the OUTx bits in the LPCOMP_CONFIG register. The output of each comparator is connected to a corresponding edge detector block. This block determines the edge that triggers the interrupt. The edge selection and interrupt enable is configured using the INTTYPE1 bits [5:4] and INTTYPE2 bits [13:12] in the LPCOMP_CONFIG register. Using the INTTYPEx bits, the interrupt type can be selected to disabled, rising edge, falling edge, or both edges, as described in **Table 24-1**.

Each comparator's output can be routed directly to a GPIO pin through the HSIOM. The comparator outputs are available as Deep-Sleep source 2 connection in the HSIOM. See **"High-speed I/O matrix"** on page 81 for details on HSIOM. For details on the pins that support the low-power comparator output, refer to the **device datasheet**. The output on these pins are direct output from the comparator and are not synchronized. Because they act as Deep-Sleep source for the pins, the comparator output is available in Deep-Sleep power mode as well.

During an edge event, the comparator will trigger an interrupt (intr_comp1/intr_comp2 signals in **Figure 24-1**). The interrupt request is registered in the COMP1 bit [0] and COMP2 bit [1] of the LPCOMP_INTR register for Comparator0 and Comparator1, respectively. Both Comparator0 and Comparator1 share a common interrupt (comp_intr signal in **Figure 24-1**), which is a logical OR of the two interrupts and mapped as the low-power comparator block's interrupt in the CPU NVIC. Refer to the **"Interrupts"** on page 38 for details. If both the comparators are used in a design, the COMP1 and/or COMP2 bits of the LPCOMP_INTR register need to be read in the interrupt service routine to know which one triggered the interrupt. Alternatively, COMP1_MASK bit [0] and COMP2_MASK bit [1] of the LPCOMP_INTR_MASK register can be used to mask the Comparator0 and Comparator1 interrupts to the CPU. Only the masked interrupts will be serviced by the CPU. After the interrupt is processed, the interrupt should be cleared by writing a '1' to the COMP1 and COMP2 bits of the LPCOMP_INTR register in firmware. If the interrupt is not cleared, the next compare event will not trigger an interrupt and the CPU will not be able to process the event.

The LPCOMP interrupt (comp1_intr/comp2_intr) is synchronous with SYSCLK. Clearing comp1_intr/comp2_intr are all synchronous.

LPCOMP_INTR_SET register bits [1:0] can be used to assert an interrupt for software debugging.

In Deep-Sleep mode, the wakeup interrupt controller (WIC) can be activated by a comparator edge event, which then wakes up the CPU. Thus, the LPCOMP has the capability to monitor a specified signal in low-power modes.

**Table 24-1. Output and interrupt configuration in LPCOMP_CONFIG register**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| LPCOMP_CONFIG[6] | OUT1 | Current/Instantaneous output value of Comparator0 |
| LPCOMP_CONFIG[14] | OUT2 | Current/Instantaneous output value of Comparator1 |
| LPCOMP_CONFIG[5:4] | INTTYPE1 | Sets on which edge Comparator0 will trigger an IRQ<br>00: Disabled<br>01: Rising Edge<br>10: Falling Edge<br>11: Both rising and falling edges |
| LPCOMP_CONFIG[13:12] | INTTYPE2 | Sets on which edge Comparator1 will trigger an IRQ<br>00: Disabled<br>01: Rising Edge<br>10: Falling Edge<br>11: Both rising and falling edges |
| LPCOMP_INTR[0] | COMP1 | Comparator0 Interrupt: hardware sets this interrupt when Comparator0 triggers. Write a '1' to clear the interrupt |
| LPCOMP_INTR[1] | COMP2 | Comparator2 Interrupt: hardware sets this interrupt when Comparator1 triggers. Write a '1' to clear the interrupt |
| LPCOMP_INTR_SET[0] | COMP1 | Write a '1' to trigger the software interrupt for Comparator0 |
| LPCOMP_INTR_SET[1] | COMP2 | Write a 1 to trigger the software interrupt for Comparator1 |

### 24.3.3 Power mode and speed configuration

The low-power comparators can operate in three power modes:

- Fast
- Slow
- Ultra low-power

The power or speed setting for Comparator0 is configured using MODE1 bits [1:0] in the LPCOMP_CONFIG register. The power or speed setting for Comparator1 is configured using MODE2 bits [9:8] in the same register. The power consumption and response time vary depending on the selected power mode; power consumption is highest in fast mode and lowest in ultra-low-power mode, response time is fastest in fast mode and slowest in ultra-low-power mode. Refer to the **device datasheet** for specifications for the response time and power consumption for various power settings.

The comparators are enabled/disabled using ENABLE1 bit [7] and ENABLE2 bit [15] in the LPCOMP_CONFIG register, as described in **Table 24-2**.

*Note: The output of the comparator may glitch when the power mode is changed while comparator is enabled. To avoid this, disable the comparator before changing the power mode.*

**Table 24-2. Comparator power mode selection bits MODE1 and MODE2**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| LPCOMP_CONFIG[1:0] | MODE1 | Compartor0 power mode selection<br>00: Slow operating mode (uses less power)<br>01: Fast operating mode (uses more power)<br>10: Ultra low-power operating mode (uses lowest possible power) |
| LPCOMP_CONFIG[9:8] | MODE2 | Compartor1 power mode selection<br>00: Slow operating mode (uses less power)<br>01: Fast operating mode (uses more power)<br>10: Ultra low-power operating mode (uses lowest possible power) |
| LPCOMP_CONFIG[7] | ENABLE1 | Comparator0 enable bit<br>0: Disables Comparator0<br>1: Enables Comparator0 |
| LPCOMP_CONFIG[15] | ENABLE2 | Comparator1 enable bit<br>0: Disables Comparator1<br>1: Enables Comparator1 |

### 24.3.4 Hysteresis

For applications that compare signals close to each other and slow changing signals, hysteresis helps to avoid oscillations at the comparator output when the signals are noisy. For such applications, a fixed 10-mV hysteresis may be enabled in the comparator block.

The 10-mV hysteresis level is enabled/disabled by using the HYST1 bit [2] and HYST2 bit [10] in the LPCOMP_CONFIG register, as described in **Table 24-3**.

**Table 24-3. Hysteresis control bits HYST1 and HYST2**

| Register[Bit_Pos] | Bit_Name | Description |
|---|---|---|
| LPCOMP_CONFIG[2] | HYST1 | Enable/Disable 10 mV hysteresis to Comparator0<br>- 0: Enable Hysteresis<br>- 1: Disable Hysteresis |
| LPCOMP_CONFIG[10] | HYST2 | Enable/Disable 10 mV hysteresis to Comparator1<br>- 0: Enable Hysteresis<br>- 1: Disable Hysteresis |

### 24.3.5 Wakeup from low-power modes

The comparator is operational in the device's low-power modes, including Sleep and Deep-Sleep modes. The comparator output interrupt can wake the device from Sleep and Deep-Sleep modes. The comparator should be enabled in the LPCOMP_CONFIG register, the INTTYPEx bits in the LPCOMP_CONFIG register should not be set to disabled, and the INTR_MASKx bit should be set in the LPCOMP_INTR_MASK register for the corresponding comparator to wake the device from low-power modes. Comparisons involving AMUXBUS connections are not available in DeepSleep mode.

In the Deep-Sleep power mode, a compare event on either Comparator0 or Comparator1 output will generate a wakeup interrupt. The INTTYPEx bits in the LPCOMP_CONFIG register should be configured, as required, for the corresponding comparator to wake the device from low-power modes.

The mask bits in the LPCOMP_INTR_MASK register is used to select whether one or both of the comparator's interrupt is serviced by the CPU.

### 24.3.6 Comparator clock

The comparator uses the system main clock SYSCLK as the clock for interrupt synchronization.

### 24.3.7 Offset trim

The comparator offset is trimmed at the factory to less than 4.0 mV. The trim is a two-step process, trimmed first at common mode voltage equal to 0.1 V, then at common mode voltage equal to VDD –0.1 V. Offset voltage is guaranteed to be less than 10.0 mV over the input voltage range of 0.1 V to VDD –0.1 V. For normal operation, further adjustment of trim values is not recommended.

If a tighter trim is required at a specific input common mode voltage, a trim may be performed at the desired input common mode voltage. The comparator offset trim is performed using the LPCOMP_TRIM1/2/3/4 registers. LPCOMP_TRIM1 and LPCOMP_TRIM2 are used to trim comparator 0. LPCOMP_TRIM3 and LPCOMP_TRIM4 are used to trim comparator 1. The bit fields that change the trim values are TRIMA bits [4:0] in LPCOMP_TRIM1 and LPCOMP_TRIM3, and TRIMB bits [3:0] in LPCOMP_TRIM2 and LPCOMP_TRIM4. TRIMA bits are used to coarse tune the offset; TRIMB bits are used to fine tune. The use of TRIMB bits for offset correction is restricted to slow mode of comparator operation.

Any standard comparator offset trim procedure can be used to perform the trimming. The following method can be used to improve the offset at a given reference/common mode voltage input.

1. Short the comparator inputs externally and connect the voltage reference, Vref, to the input.
2. Set up the comparator for comparison, turn off hysteresis, and check the output.
3. If the output is high, the offset is positive. Otherwise, the offset is negative. Follow these steps to tune the offset:
   a) Tune the TRIMA bits[4:0] until the output switches direction. TRIMA bits[3:0] control the amount of offset and TRIMA bit[4] controls the polarity of offset ('1' indicates positive offset and '0' indicates negative offset).
   b) When the tuning of TRIMA bits is complete, tune the TRIMB bits[3:0] until the output switches direction again. The TRIMB bit tuning is valid only for slow mode of comparator operation. TRIMB bit[3] controls the polarity of offset. Increasing TRIMB bits [2:0] reduces the offset.

## 24.4 Registers

**Table 24-4. Low-power comparator registers**

| Register | Function |
|---|---|
| LPCOMP_ID | Includes the information of LPCOMP controller ID and revision number |
| LPCOMP_CONFIG | LPCOMP configuration register |
| LPCOMP_INTR | LPCOMP interrupt register |
| LPCOMP_INTR_SET | LPCOMP interrupt set register |
| LPCOMP_TRIM1 | Trim fields for comparator 0 |
| LPCOMP_TRIM2 | Trim fields for comparator 0 |
| LPCOMP_TRIM3 | Trim fields for comparator 1 |
| LPCOMP_TRIM4 | Trim fields for comparator 1 |

# 25 Temperature sensor

EZ-PD™ PMG1-S3 MCU has an on-chip temperature sensor that is used to measure the internal die temperature. The sensor consists of a transistor connected in diode configuration.

## 25.1 Features

The temperature sensor has the following features:

- ± 5° celsius accuracy over temperature range –40°C to +85°C
- 0.5° celsius/LSB resolution (without amplification) when using a 12-bit SAR ADC with a 1.2-V reference
- 10 µs settling time

## 25.2 How it works

The temperature sensor consists of a single bipolar junction transistor (BJT) in the form of a diode. Its base-to-emitter voltage (VBE) has a strong dependence on temperature at a constant collector current and zero collector-base voltage. This property is used to calculate the die temperature by measuring the VBE of the transistor using SAR ADC, as shown in **Figure 25-1**.

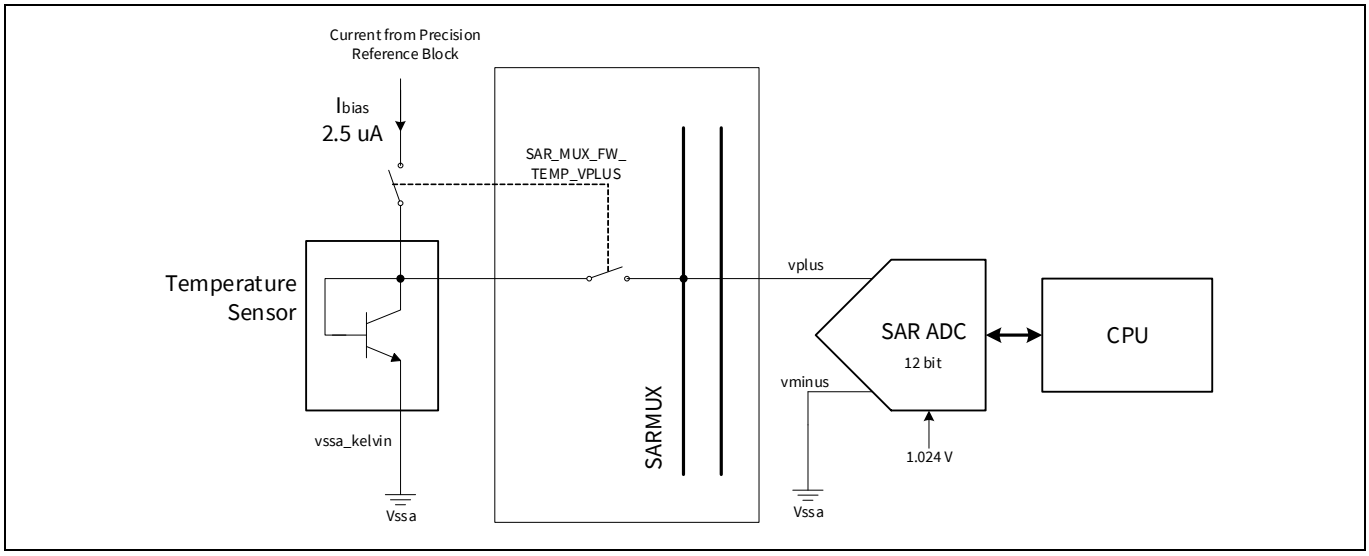



**Figure 25-1. Temperature sensing mechanism**

The analog output from the sensor (VBE) is measured using the SAR ADC. Die temperature in °C can be calculated from the ADC results as given in the following equation:

$$\mathrm{np} = (\mathrm{A} \times \mathrm{SAR}_{\mathrm{out}} + 2^{10} \times \mathrm{B}) + \mathrm{T}_{\mathrm{adj}} \quad (25.1)$$

- Temp is the slope compensated temperature in °C represented as Q16.16 fixed point number format.
- 'A' is the 16-bit multiplier constant. The value of A is determined using the EZ-PD™ PMG1-S3 MCU family characterization data of two point slope calculation. It is calculated as given in the following equation.

$$\mathrm{A} = (\text{signed int})\left(2^{16}\left(\frac{100°\mathrm{C} - (-40°\mathrm{C})}{\mathrm{SAR}_{100°\mathrm{C}} - \mathrm{SAR}_{-40°\mathrm{C}}}\right)\right) \quad (25.2)$$

Where,

SAR100C = ADC counts at 100°C

### Temperature sensor

SAR–40C = ADC counts at –40 °C

Constant 'A' is stored in a register SFLASH_SAR_TEMP_MULTIPLIER.

- 'B' is the 16-bit offset value. The value of B is determined on a per die basis by taking care of all the process variations and the actual bias current (Ibias) present in the chip. It is calculated as given in the following equation.

$$B = (\text{unsigned int})\left(2^6 \times 100°C - \left(\frac{A \times SAR_{100C}}{2^{10}}\right)\right) \quad (25.3)$$

Where,

SAR100C = ADC counts at 100°C

Constant 'B' is stored in a register SFLASH_SAR_TEMP_OFFSET.

- $T_{adjust}$ is the slope correction factor in °C. The temperature sensor is corrected for dual slopes using the slope correction factor. It is evaluated based on the result obtained without slope correction, that is, evaluating $T_{initial}$ = (A×$SAR_{out}$ + 210 ×B). If it is greater than the center value (15°C), then $T_{adjust}$ is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{100°C - 15°C} \times (100°C \times 2^{16} - T_{initial})\right) \quad (25.4)$$

If less than center value, then $T_{adjust}$ is given by the following equation.

$$T_{adjust} = \left(\frac{0.5°C}{40°C + 15°C} \times (40°C \times 2^{16} - T_{initial})\right) \quad (25.5)$$

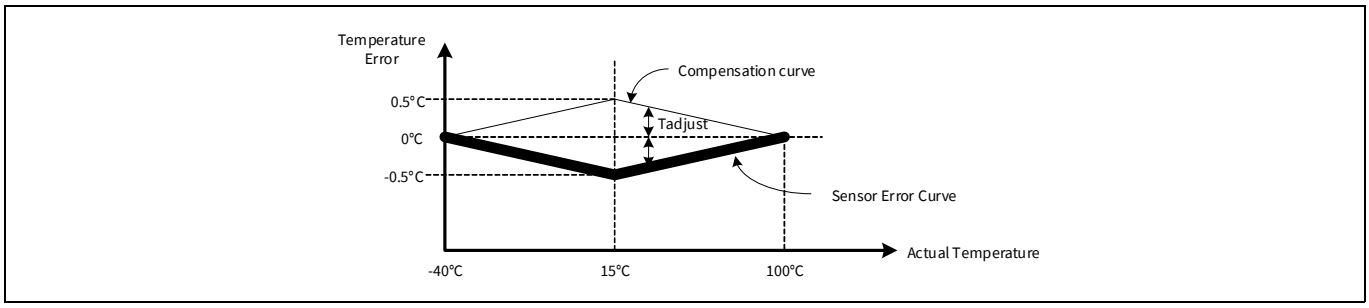



**Figure 25-2. Temperature error compensation**

*Note: A and B are 16-bit constants stored in flash during factory calibration. Note that these constants are valid only when the SAR ADC is running at 12-bit resolution with a 1.2-V reference.*

## 25.3 Temperature sensor configuration

The temperature sensor output is routed to the positive input of SAR ADC via dedicated switches, which can be controlled by sequencer, or firmware. See the **"12-bit SAR ADC"** on page 207 for details on how to read the temperature sensor output using the ADC.

## 25.4 Algorithm

1. Enable the SARMUX and SAR ADC.
2. Configure SAR ADC in single-ended mode with VNEG = VSS, VREF = 1.2 V, 12-bit resolution, and right-aligned result.
3. Enable the temperature sensor.
4. Get the digital output from the SAR ADC.
5. Fetch ‘A’ from SFLASH_SAR_TEMP_MULTIPLIER and ‘B’ from SFLASH_SAR_TEMP_OFFSET.
6. Calculate the die temperature using the linear equation (Equation 25-1).

For example, let A = 0xBC4B and B = 0x65B4. Assume that the output of SAR ADC (VBE) is 0x595 at a given temperature.

Firmware does the following calculations:

a) Multiply A and VBE: 0xBC4B × 0x595 = (–17333)10 × (1429)10 = (–24768857)10
b) Multiply B and 1024: 0x65B4 × 0x400 = (26036)10 × (1024)10 = (26660864)10
c) Add the result of steps 1 and 2 to get Tinitial: (–24768857)10 + (26660864)10 = (1892007)10 = 0x1CDEA7
d) Calculate Tadjust using Tinitial value: Tinitial is the upper 16 bits multiplied by 216, that is, 0x1C00 = (1835008)10. It is greater than 15°C (0x1C - upper 16 bits). Use Equation 4 to calculate Tadjust. It comes to 0x6C6C = (27756)10
e) Add Tadjust to Tinitial: (1892007)10 + (27756)10 = (1919763)10 = 0x1D4B13
f) The integer part of temperature is the upper 16 bits = 0x001D = (29)10
g) The decimal part of temperature is the lower 16 bits = 0x4B13 = (0.19219)10
h) Combining the result of steps f and g, Temp = 29.19219°C ~ 29.2°C

## 25.5 Registers

**Table 25-1. Temperature sensor registers**

| Name | Description |
|---|---|
| SAR_MUX_SWITCH0 | This register has the SAR_MUX_FW_TEMP_VPLUS field to connect the temperature sensor to the SAR MUX terminal |
| SAR_MUX_SWITCH_STATUS | This register provides the status of the temperature sensor switch connection to SAR MUX |
| SFLASH_SAR_TEMP_MULTIPLIER | Multiplier constant 'A' as defined in **Equation 25.2 on page 252** |
| SFLASH_SAR_TEMP_OFFSET | Constant 'B' as defined in **Equation 25.3 on page 253** |

# 26 CAPSENSE™

The CAPSENSE™ system can measure the self-capacitance of an electrode or the mutual capacitance between a pair of electrodes. In addition to capacitive sensing, the CAPSENSE™ system can function as an ADC to measure voltage on any GPIO pin that supports the CAPSENSE™ functionality.

The CAPSENSE™ touch sensing method in EZ-PD™ PMG1-S3 MCU, which senses self-capacitance, is known as CAPSENSE™ Sigma Delta (CSD). Similarly, the mutual-capacitance sensing method is known as CAPSENSE™ Cross-point (CSX). The CSD and CSX touch sensing methods provide the industry's best-in-class signal-to-noise ratio (SNR), high touch sensitivity, low-power operation, and superior EMI performance.

CAPSENSE™ touch sensing is a combination of hardware and firmware techniques. Therefore, use the CAPSENSE™ component provided by the ModusToolbox™ software IDE to implement CAPSENSE™ designs. See the **PSoC™ 4 and PSoC™ 6 MCU CAPSENSE™ design guide** for more details.

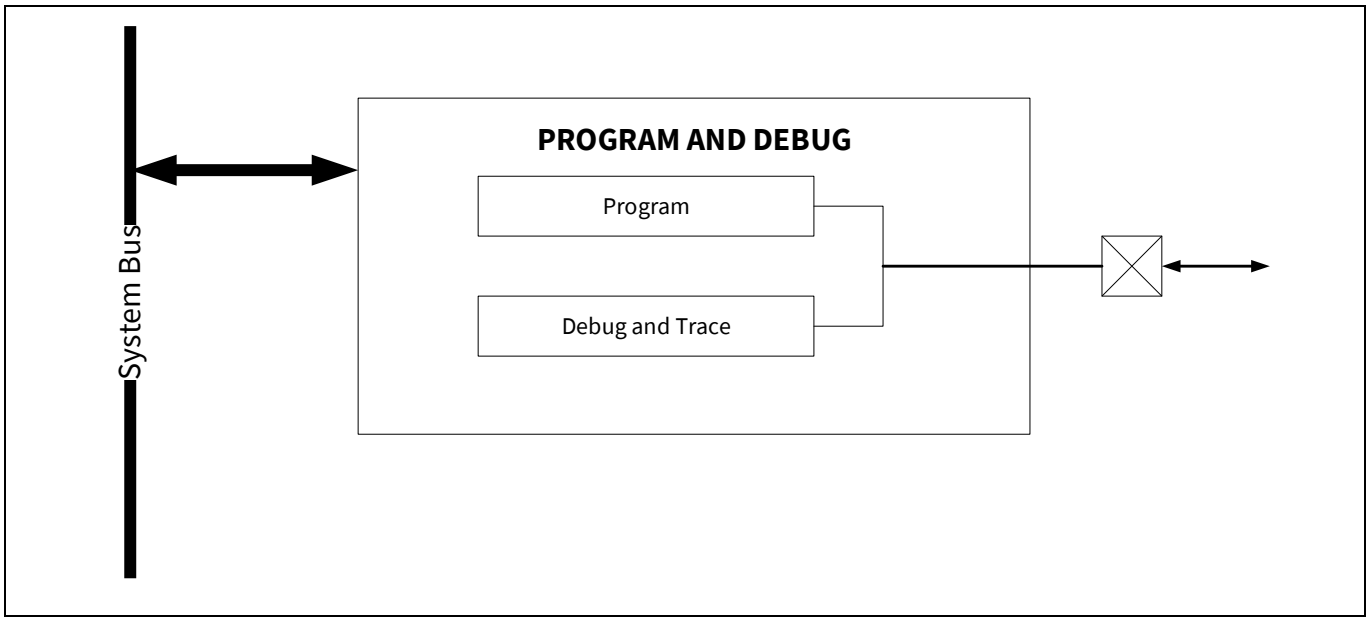
# Section H: Program and debug

This section encompasses the following chapters:

- **"Program and debug Interface"** on page 257
- **"Nonvolatile memory Programming"** on page 266

## Top level architecture

### Program and debug block diagram



# 27 Program and debug Interface

The EZ-PD™ PMG1-S3 MCU Program and Debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be a Infineon-supplied programmer and debugger, or a third-party device that supports EZ-PD™ PMG1-S3 MCU programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and EZ-PD™ PMG1-S3 MCU.

## 27.1 Features

- Programming and debugging through the SWD interface
- Four hardware breakpoints and two hardware watchpoints while debugging
- Read and write access to all memory and registers in the system while debugging, including the Cortex-M0+ register bank when the core is running or halted

## 27.2 Functional description

**Figure 27-1** shows the block diagram of the program and debug interface in EZ-PD™ PMG1-S3 MCU. The Cortex-M0+ debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the "host", communicates with the DAP of the EZ-PD™ PMG1-S3 MCU "target" using the two pins of the SWD interface – the bidirectional data pin (SWDIO) and the host-driven clock pin (SWDCK). The SWD physical port pins (SWDIO and SWDCK) communicate with the DAP through the high-speed I/O matrix (HSIOM). See the **"I/O system"** on page 77 for details on HSIOM.

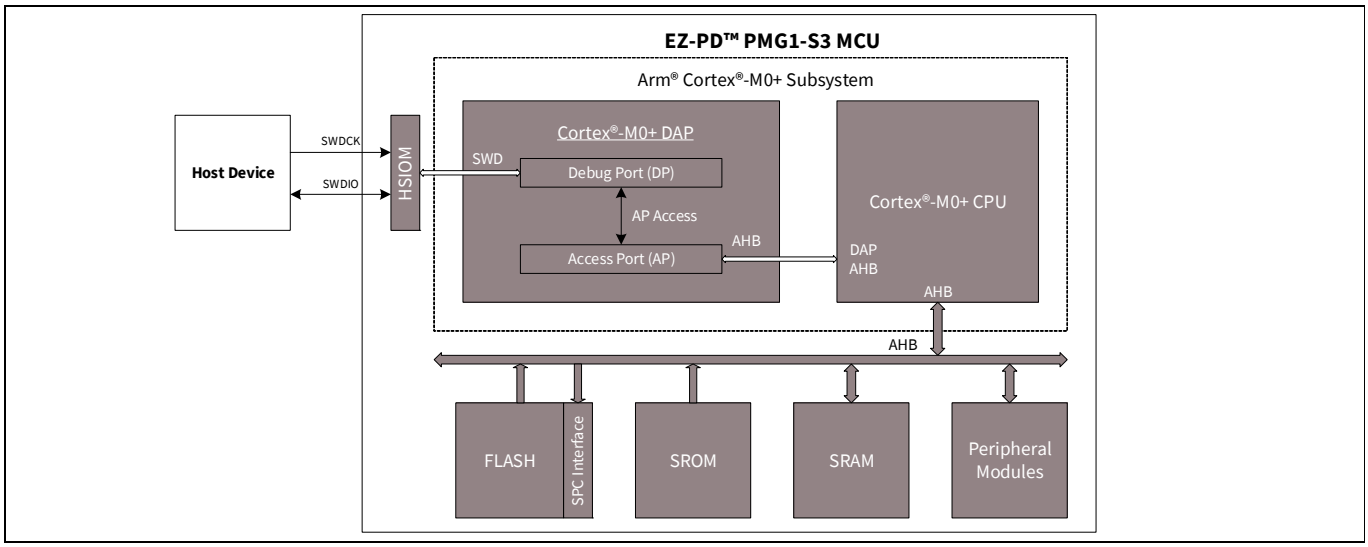



**Figure 27-1. Program and debug interface**

The DAP communicates with the Cortex®-M0+ CPU using the Arm®-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside EZ-PD™ PMG1-S3 MCU, which facilitates memory and peripheral register access by the AHB master. EZ-PD™ PMG1-S3 MCU has two AHB masters – Arm® CM0+ CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

## 27.3 Serial wire debug (SWD) interface

EZ-PD™ PMG1-S3 MCU's Cortex®-M0+ supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete data transfer (one SWD packet) requires 46 clocks and consists of three phases:

- **Host packet request phase** – The host issues a request to the EZ-PD™ PMG1-S3 MCU target.
- **Target acknowledge response phase** – The EZ-PD™ PMG1-S3 MCU target sends an acknowledgement to the host.
- **Data transfer phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

**Figure 27-2** shows the timing diagrams of read and write SWD packets.

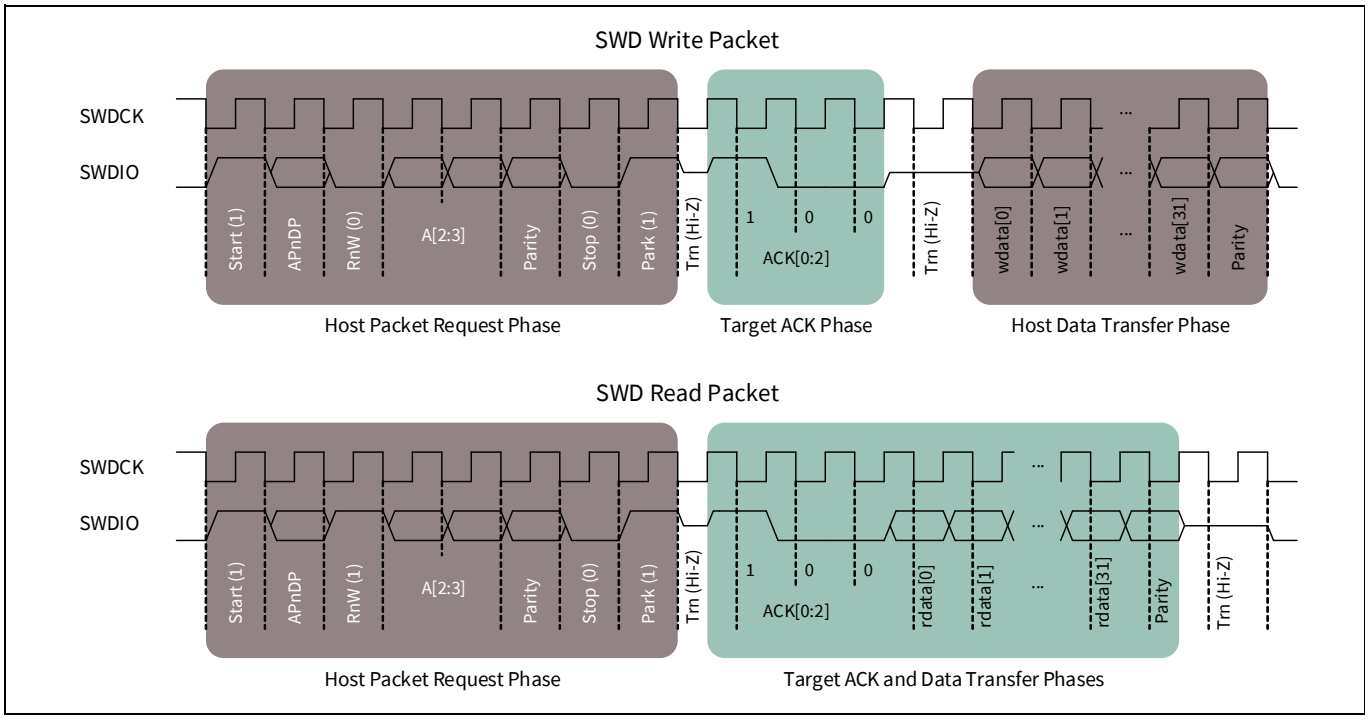



**Figure 27-2. SWD write and read packet timing diagrams**

The sequence to transmit SWD read and write packets are as follows:

1. Host packet request phase: SWDIO driven by the host
   a) The start bit initiates a transfer; it is always logic 1.
   b) The "AP not DP" (APnDP) bit determines whether the transfer is an AP access – 1b1 or a DP access – 1b0.
   c) The "Read not Write" bit (RnW) controls which direction the data transfer is in. 1b1 represents a 'read from' the target, or 1b0 for a 'write to' the target.
   d) The Address bits (A[3:2]) are register select bits for AP or DP, depending on the APnDP bit value. See **Table 27-3** and **Table 27-4** for definitions.

*Note: Address bits are transmitted with the LSB first.*

e) The parity bit contains the parity of APnDP, RnW, and ADDR bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.
f) If the parity bit is not correct, the header is ignored by EZ-PD™ PMG1-S3 MCU; there is no ACK response (ACK = 3b111). The programming operation should be aborted and retried again by following a device reset.
g) The stop bit is always logic 0.
h) The park bit is always logic 1.

2. Target acknowledge response phase: SWDIO driven by the target
   a) The ACK[2:0] bits represent the target to host response, indicating failure or success, among other results. See **Table 27-2** for definitions.

*Note: ACK bits are transmitted with the LSB first.*

3. Data transfer phase: SWDIO driven by either target or host depending on direction
   a) The data for read or write is written to the bus, LSB first.
   b) The data parity bit indicates the parity of the data read or written. It is an even parity; this means when XORed with the data bits, the result will be 0.

If the parity bit indicates a data error, corrective action should be taken. For a read packet, if the host detects a parity error, it must abort the programming operation and restart. For a write packet, if the target detects a parity error, it generates a FAULT ACK response in the next packet.

According to the SWD protocol, the host can generate any number of SWDCK clock cycles between two packets with SWDIO low. It is recommended to generate three or more dummy clock cycles between two SWD packets if the clock is not free-running or to make the clock free-running in IDLE mode.

The SWD interface can be reset by clocking the SWDCK line for 50 or more cycles with SWDIO high. To return to the idle state, clock the SWDIO low once.

### 27.3.1 SWD timing details

The SWDIO line is written to and read at different times depending on the direction of communication. The host drives the SWDIO line during the Host Packet Request Phase and, if the host is writing data to the target, during the Data Transfer phase as well. When the host is driving the SWDIO line, each new bit is written by the host on falling SWDCK edges, and read by the target on rising SWDCK edges. The target drives the SWDIO line during the Target Acknowledge Response Phase and, if the target is reading out data, during the Data Transfer Phase as well. When the target is driving the SWDIO line, each new bit is written by the target on rising SWDCK edges, and read by the host on falling SWDCK edges.

**Table 27-1** and **Figure 27-2** illustrate the timing of SWDIO bit writes and reads.

**Table 27-1. SWDIO bit write and read timing**

| SWD packet phase | SWDIO edge | |
|---|---|---|
| | **Falling** | **Rising** |
| Host packet request | Host write | Target read |
| Host data transfer | | |
| Target ack response | Host read | Target write |
| Target data transfer | | |

### 27.3.2 ACK details

The acknowledge (ACK) bit-field is used to communicate the status of the previous transfer. OK ACK means that previous packet was successful. A WAIT response requires a data phase. For a FAULT status, the programming operation should be aborted immediately. **Table 27-2** shows the ACK bit-field decoding details.

**Table 27-2. SWD transfer ACK response decoding**

| Response | ACK[2:0] |
|---|---|
| OK | 3b001 |
| WAIT | 3b010 |
| FAULT | 3b100 |
| NO ACK | 3b111 |

Details on WAIT and FAULT response behaviors are as follows:

- For a WAIT response, if the transaction is a read, the host should ignore the data read in the data phase. The target does not drive the line and the host must not check the parity bit as well.
- For a WAIT response, if the transaction is a write, the data phase is ignored by the EZ-PD™ PMG1-S3 MCU. But, the host must still send the data to be written to complete the packet. The parity bit corresponding to the data should also be sent by the host.
- For a WAIT response, it means that the EZ-PD™ PMG1-S3 MCU is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received. If it fails, then the programming operation should be aborted and retried again.
- For a FAULT response, the programming operation should be aborted and retried again by doing a device reset.

### 27.3.3 Turnaround (Trn) period details

There is a turnaround period between the packet request and the ACK phases, as well as between the ACK and the data phases for host write transfers, as shown in **Figure 27-2**. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor EZ-PD™ PMG1-S3 MCU should drive the SWDIO line during the Trn period.

## 27.4 Cortex®-M0+ debug and access port (DAP)

The Cortex®-M0+ program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex®-M0+ DAP-AHB interface. Typically, the DP registers are used for a one time configuration or for error detection purposes, and the AP registers are used to perform the programming and debugging operations. Complete architecture details of the DAP is available in the **Arm® debug interface v5 architecture specification**.

### 27.4.1 Debug port (DP) registers

Table 27-3 shows the Cortex®-M0+ DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the **Arm® debug interface v5 architecture specification** for details on all of the DP registers.

**Table 27-3. Main debug port (DP) registers**

| Register | APnDP | Address A[3:2] | RnW | Full name | Register functionality |
|---|---|---|---|---|---|
| ABORT | 0 (DP) | 2b00 | 0 (W) | AP Abort Register | This register is used to force a DAP abort and to clear the error and sticky flag conditions. |
| IDCODE | 0 (DP) | 2b00 | 1 (R) | Identification Code Register | This register holds the SWD ID of the Cortex®-M0+ CPU, which is 0x0BB11477. |
| CTRL/STAT | 0 (DP) | 2b01 | X (R/W) | Control and Status Register | This register allows control of the DP and contains status information about the DP. |
| SELECT | 0 (DP) | 2b10 | 0 (W) | AP Select Register | This register is used to select the current AP. In EZ-PD™ PMG1-S3 MCU, there is only one AP, which interfaces with the DAP AHB. |
| RDBUFF | 0 (DP) | 2b11 | 1 (R) | Read Buffer Register | This register holds the result of the last AP read operation. |

### 27.4.2 Access port (AP) registers

Table 27-4 lists the main Cortex®-M0+ AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

**Table 27-4. Main access port (AP) registers**

| Register | APnDP | Address A[3:2] | RnW | Full name | Register functionality |
|---|---|---|---|---|---|
| CSW | 1 (AP) | 2b00 | X (R/W) | Control and Status Word Register (CSW) | This register configures and controls accesses through the memory access port to a connected memory system (which is the EZ-PD™ PMG1-S3 MCU memory map) |
| TAR | 1 (AP) | 2b01 | X (R/W) | Transfer Address Register | This register is used to specify the 32-bit memory address to be read from or written to |
| DRW | 1 (AP) | 2b11 | X (R/W) | Data Read and Write Register | This register holds the 32-bit data read from or to be written to the address specified in the TAR register |

## 27.5 Programming the EZ-PD™ PMG1-S3 MCU device

EZ-PD™ PMG1-S3 MCU is programmed using the following sequence. Refer to the **CYPMxxxx Programming specifications** for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in EZ-PD™ PMG1-S3 MCU.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

### 27.5.1 SWD port acquisition

#### 27.5.1.1 Primary and secondary SWD pin pairs

The first step in device programming is to acquire the SWD port in EZ-PD™ PMG1-S3 MCU. EZ-PD™ PMG1-S3 MCU devices have two different pairs of pins that can be used as SWDCK and SWDIO. The primary pair is active by default, but the selection can be changed to the secondary pins by firmware. Refer to the **device datasheet** for information on SWD pins.

#### 27.5.1.2 SWD port acquire sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After removing the XRES signal, the host must send an SWD connect sequence for the device within the acquire window to connect to the SWD interface in the DAP. The pseudo code for the sequence is given here.

Code 1. SWD port acquire pseudo code

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute Arm's connection sequence to acquire SWD-port
do
{
       SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
       ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register

}while ((ack != OK) && time_elapsed < 5 ms); //retry connection until OK ACK or timeout

if (time_elapsed >= 5 ms) return FAIL; //check for acquire time out

if (ID != CM0P_ID) return FAIL; //confirm SWD ID of Cortex-M0+ CPU. (0x0BB11477)
```

In this pseudo code, SWD_LineReset() is the standard Arm® command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (5 ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex®-M0+ DAP.

### 27.5.2 SWD programming mode entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudo code for entering the programming mode are detailed in the **CYPMxxxx Programming specifications** document.

### 27.5.3 SWD programming routines executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in **"Nonvolatile memory Programming"** on page 266. The exact sequence of calling the programming routines is given in the **CYPMxxxx Programming specifications** document.

## 27.6 EZ-PD™ PMG1-S3 MCU SWD debug interface

Cortex®-M0+ DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex®-M0+ debug.

See the **Armv6-M architecture reference manual** for complete details on the debug architecture.

### 27.6.1 Debug control and configuration registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the **Armv6-M architecture reference manual** for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0P_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0P_DFSR) – This register describes the reason a debug event has occurred. This includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0P_DCRSR) – This register is used to select the general-purpose register in the Cortex-M0+ CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0P_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0P_DCRSR register.
- Debug Exception and Monitor Control Register (CM0P_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

### 27.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0+ DAP in EZ-PD™ PMG1-S3 MCU supports up to four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction in the Cortex-M0+. The BPU has two types of registers.

- The breakpoint control register (CM0P_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0+ DAP in EZ-PD™ PMG1-S3 MCU).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0P_BP_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

### 27.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. Trace is not supported by the Cortex-M0+ in EZ-PD™ PMG1-S3 MCU. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows.

- The watchpoint compare (CM0P_DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint mask (CM0P_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0P_DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event has occurred.
- The watchpoint comparator PC sample register (CM0P_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, noninvasive profiling of the program counter register.

### 27.6.4 Debugging the EZ-PD™ PMG1-S3 MCU device

The host debugs the target EZ-PD™ PMG1-S3 MCU device by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0+ DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target EZ-PD™ PMG1-S3 MCU device is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the correct CM0+ DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the **"I/O system"** on page 77 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

Debugging the target device is also affected by the overall device protection setting, which is explained in the **"Device security"** on page 104. Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep-Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

## 27.7 Registers

**Table 27-5. Program and debug interface registers**

| Register name | Description |
|---|---|
| CM0P_DHCSR | Debug Halting Control and Status Register |
| CM0P_DFSR | Debug Fault Status Register |
| CM0P_DCRSR | Debug Core Register Selector Register |
| CM0P_DCRDR | Debug Core Register Data Register |
| CM0P_DEMCR | Debug Exception and Monitor Control Register |
| CM0P_BP_CTRL | Breakpoint control register |
| CM0P_BP_COMPx | Breakpoint Compare Register |
| CM0P_DWT_COMPx | Watchpoint Compare Register |
| CM0P_DWT_MASKx | Watchpoint Mask Register |
| CM0P_DWT_FUNCTIONx | Watchpoint Function Register |
| CM0P_DWT_PCSR | Watchpoint Comparator PC Sample Register |

# 28 Nonvolatile memory Programming

Nonvolatile memory programming refers to the programming of flash memory in the EZ-PD™ PMG1-S3 MCU device. This chapter explains the different functions that are part of device programming, such as erase, write, program, and checksum calculation. Infineon-supplied programmers and other third-party programmers can use these functions to program the EZ-PD™ PMG1-S3 MCU device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

## 28.1 Features

- Supports programming through the debug and access port (DAP) and Cortex®-M0+ CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex®-M0+ CPU

## 28.2 Functional description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0+ CPU requests the system call by writing the function opcode and parameters to the System Performance Controller Interface (SPCIF) input registers, and then requesting the SROM to execute the function. Based on the function opcode, the System Performance Controller (SPC) executes the corresponding system call from SROM and updates the SPCIF status register. The DAP or the CPU should read this status register for the pass/fail result of the function execution. As part of function execution, the code in SROM interacts with the SPCIF to do the actual flash programming operations.

EZ-PD™ PMG1-S3 MCU flash is programmed using a Program Erase Program (PEP) sequence. The flash cells are all programmed to a known state, erased, and then the selected bits are programmed. This increases the life of the flash by balancing the stored charge. When writing to flash the data is first copied to a page latch buffer. The flash write functions are then used to transfer this data to flash. External programmers program the flash memory in EZ-PD™ PMG1-S3 MCU using the SWD protocol by sending the commands to the DAP. The programming sequence for the EZ-PD™ PMG1-S3 MCU device with an external programmer is given in the **CYPMxxxx Programming specifications**. Flash memory can also be programmed by the CM0+ CPU by accessing the relevant registers through the AHB interface. This type of programming is typically used to update a portion of the flash memory as part of a bootload operation, or other application requirements, such as updating a lookup table stored in the flash memory. All write operations to flash memory, whether from the DAP or from the CPU, are done through the SPCIF.

*Note: It can take as much as 20 milliseconds to write to flash. During this time, the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources (see the **"Reset system"** on page 102) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.*

## 28.3 System call implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: Two 8-bit parameters are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows:
  key1 = 0xB6
  key2 = 0xD3 + Opcode
  The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, additional parameters may be required depending on the specific function being called.
- Return values: Some system calls also return a value on completion of their execution, such as the silicon ID or a checksum.
- Completion status: Each system call returns a 32-bit status that the CPU or DAP can read to verify success or determine the reason for failure.

## 28.4 Blocking and non-blocking system calls

System call functions can be categorized as blocking or non-blocking based on the nature of their execution. Blocking system calls are those where the CPU cannot execute any other task in parallel other than the execution of the system call. When a blocking system call is called from a process, the CPU jumps to the code corresponding in SROM. When the execution is complete, the original thread execution resumes. Non-blocking system calls allow the CPU to execute some other code in parallel and communicate the completion of interim system call tasks to the CPU through an interrupt.

Non-blocking system calls are only used when the CPU initiates the system call. The DAP will only use system calls during the programming mode and the CPU is halted during this process. The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

Because the CPU cannot execute code from flash while doing an erase or program operation on the flash, the non-blocking system calls can only be called from a code executing out of SRAM. If the non-blocking functions are called from flash memory, the result is undefined and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

The System Performance Controller (SPC) is the block that generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called from SRAM, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure that the subsequent steps in the system call are completed. Because the CPU can execute code only from the SRAM when a non-blocking write or program operation is being done, the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. The Resume Non-Blocking function call done in the SPC ISR is called once in a non-blocking program operation and thrice in a non-blocking write operation.

The pseudo code for using a non-blocking write system call and executing user code out of SRAM is given later in this chapter.

### 28.4.1 Performing a system call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, additional parameters) are:
   a) Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b) Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits are not cleared after one second, the operation should be considered a failure and aborted without executing the following steps. Note that unlike the DAP, the CPU application code cannot poll these bits during system call execution. This is because the CPU executes code out of the SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.
4. Check the completion status: After the PRIVILEGED and SYSCALL_REQ bits are cleared to indicate completion of the system call, the CPUSS_SYSARG register should be read to check for the status of the system call. If the 32-bit value read from the CPUSS_SYSARG register is 0xAXXXXXXX (where 'X' denotes don't care hex values), the system call was successfully executed. For a failed system call, the status code is 0xF00000YY where YY indicates the reason for failure. See **Table 28-2** for the complete list of status codes and their description.
5. Retrieve the return values: For system calls that return values such as silicon ID and checksum, the CPU or DAP should read the CPUSS_SYSREG and CPUSS_SYSARG registers to fetch the values returned.

## 28.5 System calls

Table 28-1 lists all the system calls supported in EZ-PD™ PMG1-S3 MCU along with the function description and availability in device protection modes. See the "Device security" on page 104 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

**Table 28-1. List of system calls**

| System call | Description | DAP access | | | CPU access |
|---|---|---|---|---|---|
| | | Open | Protected | Kill | |
| Silicon ID | Returns the device Silicon ID, Family ID, and Revision ID | ✓ | ✓ | – | ✓ |
| Load Flash Bytes | Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 256 bytes | ✓ | – | – | ✓ |
| Write Row | Erases and then programs a row of flash with data in the page latch buffer | ✓ | – | – | ✓ |
| Program Row | Programs a row of flash with data in the page latch buffer | ✓ | – | – | ✓ |
| Erase All | Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area | ✓ | – | – | – |
| Checksum | Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash | ✓ | ✓ | – | ✓ |
| Write Protection | This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0) | ✓ | ✓ | – | – |
| Non-Blocking Write Row | Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access | – | – | – | ✓ |
| Non-Blocking Program Row | Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access | – | – | – | ✓ |
| Resume Non-Blocking | Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access | – | – | – | ✓ |

### 28.5.1 Silicon ID

This function returns a 12-bit family ID, 16-bit silicon ID, and an 8-bit revision ID, and the current device protection mode. These values are returned to the CPUSS_SYSARG and CPUSS_SYSREQ registers. Parameters are passed through the CPUSS_SYSARG and CPUSS_SYSREQ registers.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD3 | Key2 |
| Bits [31:16] | 0x0000 | Not used |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0000 | Silicon ID opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | Silicon ID Lo | See the **CYPMxxxx Programming specifications** for Silicon ID values for different part numbers |
| Bits [15:8] | Silicon ID Hi | |
| Bits [19:16] | Minor Revision Id | See the **CYPMxxxx Programming specifications** for these values |
| Bits [23:20] | Major Revision Id | |
| Bits [27:24] | 0xXX | Not used (don't care) |
| Bits [31:28] | 0xA | Success status code |
| CPUSS_SYSREQ register | | |
| Bits [11:0] | Family ID | Family ID is 0xC0 for EZ-PD™ PMG1 MCU family |
| Bits [15:12] | Chip Protection | See the **"Device security"** on page 104 |
| Bits [31:16] | 0xXXXX | Not used |

### 28.5.2 Configure clock

This function initializes the clock necessary for flash programming and erasing operations. This API is used to ensure that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz prior to calling the flash write and flash erase APIs. The flash write and erase APIs will exit without acting on the flash and return the “Invalid Pump Clock Frequency” status if the IMO is the source of the charge pump clock and is not 48 MHz.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xE8 | Key2 |
| Bits [31:16] | 0xXXXX | Don’t care |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0015 | Configure clock opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don’t care) |

### 28.5.3 Load flash bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 128 bytes. Data is loaded into the page latch buffer starting at the location specified by the “Byte Addr” input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS_SYSARG register. Note that the starting parameter address should be a word-aligned address.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address – 32’hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD7 | Key2 |
| Bits [23:16] | Byte Addr | Start address of page latch buffer to write data<br>0x00 – Byte 0 of latch buffer<br>0xFF – Byte 255 of latch buffer |
| Bits [31:24] | Flash Macro Select | 0x00 - Flash Macro 0<br>0x01 - Flash Macro 1 |
| SRAM Address- 32’hYY + 0x04 | | |
| Bits [7:0] | Load Size | Number of bytes to be written to the page latch buffer.<br>0x00 – 1 byte<br>0xFF – 256 bytes |
| Bits [15:8] | 0xXX | Don’t care parameter |
| Bits [23:16] | 0xXX | Don’t care parameter |
| Bits [31:24] | 0xXX | Don’t care parameter |
| SRAM Address- From (32’hYY + 0x08) to (32’hYY + 0x08 + Load Size) | | |
| Byte 0 | Data Byte [0] | First data byte to be loaded |
| . | . | . |
| . | . | . |
| Byte (Load size –1) | Data Byte [Load size –1] | Last data byte to be loaded |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32’hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0004 | Load Flash Bytes opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don't care) |

### 28.5.4 Write row

This function erases and then programs the addressed row of flash with the data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The parameters for this function are stored in SRAM. The start address of the stored parameters is written to the CPUSS_SYSARG register. This function clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write protected.

Note that the SROM does not modify, enable, or disable any clock during any flash operation. Refer to the CLK_IMO_CONFIG register in the EZ-PD™ PMG1-S3 MCU registers TRM for more information.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD8 | Key2 |
| Bits [31:16] | Row ID | Row number to write<br>0x0000 – Row 0<br>0x03FF - Row 1023 |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0005 | Write Row opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don't care) |

### 28.5.5 Program row

This function programs the addressed row of the flash, with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The row must be in an erased state before calling this function. This clears the page latch buffer contents after the row is programmed.

Usage requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address: 32’hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xD9 | Key2 |
| Bits [31:16] | Row ID | Row number to program<br>0x0000 – Row 0<br>0x03FF - Row 1023 |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32’hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0006 | Program Row opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don’t care) |

### 28.5.6 Erase all

This function erases all the user code in the flash main arrays and the row-level protection data in supervisory flash row 0.

Usage requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. This API can be called only from the DAP in the programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address: 32’hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDD | Key2 |
| Bits [31:16] | 0xXXXX | Don’t care |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32’hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x000A | Erase All opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:0] | 0xXXXXXXX | Not used (don’t care) |

### 28.5.7 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDE | Key2 |
| Bits [31:16] | Row ID | Selects the flash row number on which the checksum operation is done<br>Row number – 16 bit flash row number or<br>0x8000 – Checksum is performed on entire flash memory |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x000B | Checksum opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:24] | 0xX | Not used (don’t care) |
| Bits [23:0] | Checksum | 24-bit checksum value of the selected flash region |

### 28.5.8 Write protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by eight. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

Then, the Write Protection function is called, which programs the flash protection bytes from the page latch to be the corresponding flash macro's supervisory row. In flash macro zero, which also stores the device protection settings, the device level protection setting is passed as a parameter in the CPUSS_SYSARG register.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xE0 | Key2 |
| Bits [23:16] | Device Protection Byte | Parameter applicable only for Flash Macro 0<br>0x01 – OPEN mode<br>0x02 – PROTECTED mode<br>0x04 – KILL mode |
| Bits [31:24] | Flash Macro Select | 0x00 - Flash Macro 0<br>0x01 - Flash Macro 1 |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x000D | Write Protection opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code |
| Bits [27:24] | 0xX | Not used (don't care) |
| Bits [23:0] | 0x000000 | |

### 28.5.9 Non-blocking write row

This function is used when a flash row needs to be written by the CM0+ CPU in a non-blocking manner, so that the CPU can execute code from SRAM while the write operation is being done. **"Blocking and non-blocking system calls"** on page 267 describes non-blocking system calls in detail.

The non-blocking write row system call has three phases: Pre-program, Erase, Program. Pre-program is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all of the bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code from SRAM. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After the completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz.

*Note:* *The device firmware must not attempt to put the device to sleep during a non-blocking write row. This will reset the page latch buffer and the flash will be written with all zeroes.*

Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0+ CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDA | Key2 |
| Bits [31:16] | Row ID | Row number to write<br>0x0000 – Row 0<br>0x03FF - Row 1023 |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0007 | Non-Blocking Write Row opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits[31:0] | 32'hYY | SRAM addressed programmed by user in case of success. Will return status code in case of error. |

### 28.5.10 Non-blocking program row

This function is used when a flash row needs to be programmed by the CM0+ CPU in a non-blocking manner, so that the CPU can execute code from the SRAM when the program operation is being done. **"Blocking and non-blocking system calls"** on page 267 describes non-blocking system calls in detail. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

Unlike the Non-Blocking Write Row system call, the Program system call only has a single phase. Therefore, the Resume Non-Blocking function only needs to be called once from the SPC interrupt when using the Non-Blocking Program Row system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0+ CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDB | Key2 |
| Bits [31:16] | Row ID | Row number to write<br>0x0000 – Row 0<br>0x03FF - Row 1023 |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0008 | Non-Blocking Program Row opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits[31:0] | 32'hYY | SRAM addressed programmed by user in case of success. Will return status code in case of error. |

### 28.5.11 Resume non-blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are available in **"Blocking and non-blocking system calls"** on page 267.

**Parameters**

| Address | Value to be written | Description |
|---|---|---|
| SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address) | | |
| Bits [7:0] | 0xB6 | Key1 |
| Bits [15:8] | 0xDC | Key2 |
| Bits [31:16] | 0xXXXX | Don't care. Not used by SROM |
| CPUSS_SYSARG register | | |
| Bits [31:0] | 32'hYY | 32-bit word-aligned address of the SRAM that stores the first function parameter (key1) |
| CPUSS_SYSREQ register | | |
| Bits [15:0] | 0x0009 | Resume Non-Blocking opcode |
| Bits [31:16] | 0x8000 | Set SYSCALL_REQ bit |

**Return**

| Address | Return value | Description |
|---|---|---|
| CPUSS_SYSARG register | | |
| Bits [31:28] | 0xA | Success status code in the case of the last resume call which completes the program or write operation. |
| Bits [31:0] | 32'hYY | SRAM addressed programmed by user in case of intermediate resume operations. |

## 28.6 System call status

At the end of every system call, a status code is written over the arguments in the CPUSS_SYSARG register. A success status is 0xAXXXXXXX, where X indicates don't care values or return data in the case of the system calls that return a value. A failure status is indicated by 0xF00000XX, where XX is the failure code.

**Table 28-2. System call status codes**

| Status code (32-bit value in CPUSS_SYSARG register) | Description |
|---|---|
| AXXXXXXXh | Success – The "X" denotes a don't care value, which has a value of '0' returned by the SROM, unless the API returns parameters directly to the CPUSS_SYSARG register. |
| F0000001h | Invalid Chip Protection Mode – This API is not available during the current chip protection mode. |
| F0000003h | Invalid Page Latch Address – The address within the page latch buffer is either out of bounds or the size provided is too large for the page address. |
| F0000004h | Invalid Address – The row ID or byte address provided is outside of the available memory. |
| F0000005h | Row Protected – The row ID provided is a protected row. |
| F0000007h | Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API. |
| F0000008h | Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other API's may be called. |
| F0000009h | System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume. |
| F000000Ah | Checksum Zero Failed – The calculated checksum was not zero. |
| F000000Bh | Invalid Opcode – The opcode is not a valid API opcode. |
| F000000Ch | Key Opcode Mismatch – The opcode provided does not match key1 and key2. |
| F000000Eh | Invalid Start Address – The start address is greater than the end address provided. |
| F0000012h | Invalid Pump Clock Frequency – IMO must be set to 48 MHz and HF clock source to the IMO clock source before flash write/erase operations. |

## 28.7 Non-blocking system call pseudo code

This section contains pseudo code to demonstrate how to set up a non-blocking system call and execute code out of SRAM during the flash programming operations.

```
#define REG(addr)(*((volatile uint32 *) (addr)))
#define CM0P_ISER_REG REG( 0xE000E100 )
#define CPUSS_CONFIG_REG REG( 0x40100000 )
#define CPUSS_SYSREQ_REG REG( 0x40100004 )
#define CPUSS_SYSARG_REG REG( 0x40100008 )
#define ROW_SIZE              256
//Variable to keep track of how many times SPC ISR is triggered
__ram int iStatusInt = 0x00;

__flash int main(void)
{
   DoUserStuff();
   //CM0+ interrupt enable bit for spc interrupt enable
   CM0P_ISER_REG |= 0x00004000;
   //Set CPUSS_CONFIG.VECT_IN_RAM because SPC ISR should be in SRAM
   CPUSS_CONFIG_REG |= 0x00000001;
   //Call non-blocking write row API
   NonBlockingWriteRow();
   //End Program
   while(1);
}
__sram void SpcIntHandler(void)
{
   /* Call Resume API */
   // Write key1, key2 parameters to SRAM
   REG( 0x20002000 ) = 0x0000DCB6;
   //Write the address of key1 to the CPUSS_SYSARG reg
   CPUSS_SYSARG_REG = 0x20002000;
   //Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
   //register and assert the sysreq bit
   CPUSS_SYSREQ_REG = 0x80000009;
   iStatusInt ++; // Number of times the ISR has triggered
}
 __sram void NonBlockingWriteRow(void)
 {
    int iter;
    /*Load the Flash page latch with data to write*/
    //Write key1, key2, byte address,
    //and macro sel parameters to SRAM
    REG( 0x20002000 ) = 0x0000D7B6;
    //Write load size param (128 bytes) to SRAM
    REG( 0x20002004 ) = 0x0000007F;
    for(i = 0; i < ROW_SIZE/4; i += 1);
    {
        REG( 0x20002008 + i*4 ) = 0xDADADADA;
    }
    //Write the address of the key1 param to CPUSS_SYSARG reg
    CPUSS_SYSARG_REG = 0x20002000;
    //Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
```

```
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000004;
        /*Perform Non-Blocking Write Row on Row 200 as an example */
        //Write key1, key2, row id to SRAM
        //row id = 0xC8 -> which is row 200
        REG( 0x20002000 ) = 0x00C8DAB6;
        //Write the address of the key1 param to CPUSS_SYSARG reg
        CPUSS_SYSARG_REG = 0x20002000;
        //Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000007;
        //Execute user code until iStatusInt equals 3 to signify
        //3 SPC interrupts have happened. This should be 1 in case
        // of non-blocking program System Call
        while( iStatusInt != 0x03 )
        {
            DoOtherUserStuff();
        }
        //Get the success or failure status of System Call
        syscall_status = CPUSS_SYSARG_REG;
    }
```

In the code, the CM0+ exception table is configured to be in SRAM by writing 0x01 to the CPUSS_CONFIG register. The SRAM exception table should have the vector address of the SPC interrupt as the address of the *SpcIntHandler()* function, which is also defined to be in SRAM. See the **“Interrupts”** on page 38 for details on configuring the CM0+ exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

# References

- EZ-PD™ PMG1-S3 MCU registers TRM
- **EZ-PD™ PMG1-S3 MCU datasheet**
- **PSoC™ 4 and PSoC™ 6 MCU CAPSENSE™ design guide**

# Terminology

The section explains the terminology used in this technical reference manual. The terms are characterized in ***bold, italic font*** throughout the text of this manual.

**A**

| | |
|---|---|
| ***accumulator*** | In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU). |
| ***active high*** | 1. A logic signal having its asserted state as the logic 1 state.<br>2. A logic signal having the logic 1 state as the higher voltage of the two states. |
| ***active low*** | 1. A logic signal having its asserted state as the logic 0 state.<br>2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic. |
| ***address*** | The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored. |
| ***algorithm*** | A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation. |
| ***ambient temperature*** | The temperature of the air in a designated area, particularly the area surrounding the device. |
| ***analog*** | See ***analog signals***. |
| ***analog blocks*** | The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more. |
| ***analog output*** | An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0. |
| ***analog signals*** | A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time. |
| ***analog-to-digital (ADC)*** | A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation. |
| ***AND*** | See *Boolean Algebra*. |

## Terminology

| | |
|---|---|
| ***API (Application Programming Interface)*** | A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications. |
| ***array*** | An array, also known as a vector or list, is one of the simplest data structures in computer programming. Arrays hold a fixed number of equally-sized data elements, generally of the same data type. Individual elements are accessed by index using a consecutive range of integers, as opposed to an associative array. Most high-level programming languages have arrays as a built-in data type. Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers; for example, by a group of two integers. One- and two-dimensional arrays are the most common. Also, an array can be a group of capacitors or resistors connected in some common form. |
| ***assembly*** | A symbolic representation of the machine language of a specific processor. Assembly language is converted to machine code by an assembler. Usually, each line of assembly code produces one machine instruction, though the use of macros is common. Assembly languages are considered low-level languages; where as C is considered a high-level language. |
| ***asynchronous*** | A signal whose data is acknowledged or acted upon immediately, irrespective of any clock signal. |
| ***attenuation*** | The decrease in intensity of a signal as a result of absorption of energy and of scattering out of the path to the detector, but not including the reduction due to geometric spreading. Attenuation is usually expressed in dB. |

### B

| | |
|---|---|
| ***bandgap reference*** | A stable voltage reference design that matches the positive temperature coefficient of $V_T$ with the negative temperature coefficient of $V_{BE}$, to produce a zero temperature coefficient (ideally) reference. |
| ***bandwidth*** | 1. The frequency range of a message or information processing system measured in hertz.<br>2. The width of the spectral region over which an amplifier (or absorber) has substantial gain (or loss); it is sometimes represented more specifically as, for example, full width at half maximum. |
| ***bias*** | 1. A systematic deviation of a value from a reference value.<br>2. The amount by which the average of a set of values departs from a reference value.<br>3. The electrical, mechanical, magnetic, or other force (field) applied to a device to establish a reference level to operate the device. |
| ***bias current*** | The constant low-level DC current that is used to produce a stable operation in amplifiers. This current can sometimes be changed to alter the bandwidth of an amplifier. |

## Terminology

| | |
|---|---|
| ***binary*** | The name for the base 2 numbering system. The most common numbering system is the base 10 numbering system. The base of a numbering system indicates the number of values that may exist for a particular positioning within a number for that system. For example, in base 2, binary, each position may have one of two values (0 or 1). In the base 10, decimal, numbering system, each position may have one of ten values (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). |
| ***bit*** | A single digit of a binary number. Therefore, a bit may only have a value of '0' or '1'. A group of 8 bits is called a byte. Because the EZ-PD™ PMG1-S3 MCU's M8CP is an 8-bit microcontroller, the devices's native data chunk size is a byte. |
| ***bit rate (BR)*** | The number of bits occurring per unit of time in a bit stream, usually expressed in bits per second (bps). |
| ***block*** | 1. A functional unit that performs a single function, such as an oscillator.<br>2. A functional unit that may be configured to perform one of several functions, such as a digital block or an analog block. |
| ***Boolean Algebra*** | In mathematics and computer science, Boolean algebras or Boolean lattices, are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the set theoretic operations union, intersection, and complement. Boolean algebra also defines a set of theorems that describe how Boolean equations can be manipulated. For example, these theorems are used to simplify Boolean equations, which will reduce the number of logic elements needed to implement the equation.<br>The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR, and NOT. In describing circuits, NAND (NOT AND), NOR (NOT OR), XNOR (exclusive NOT OR), and XOR (exclusive OR) may also be used. Mathematicians often use + (for example, A+B) for OR and • for AND (for example, A*B) (in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated (for example, ~A, A_, !A). |
| ***break-before-make*** | The elements involved go through a disconnected state entering ('break") before the new connected state ("make"). |
| ***broadcast net*** | A signal that is routed throughout the microcontroller and is accessible by many blocks or systems. |
| ***buffer*** | 1. A storage area for data that is used to compensate for a speed difference, when transferring data from one device to another. Usually refers to an area reserved for I/O operations, into which data is read, or from which data is written.<br>2. A portion of memory set aside to store data, often before it is sent to an external device or as it is received from an external device.<br>3. An amplifier used to lower the output impedance of a system. |

## Terminology

| | |
|---|---|
| ***bus*** | 1. A named connection of nets. Bundling nets together in a bus makes it easier to route nets with similar routing patterns.<br>2. A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, address[7:0].<br>3. One or more conductors that serve as a common connection for a group of related devices. |
| ***byte*** | A digital storage unit consisting of 8 bits. |

### C

| | |
|---|---|
| ***C*** | A high-level programming language. |
| ***capacitance*** | A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads. |
| ***capture*** | To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file. |
| ***chaining*** | Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another. |
| ***checksum*** | The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a pre-determined value. |
| ***clear*** | To force a bit/register to a value of logic ‘0’. |
| ***clock*** | The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks. |
| ***clock generator*** | A circuit that is used to generate a clock signal. |
| ***CMOS*** | The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor. |
| ***comparator*** | An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements. |
| ***compiler*** | A program that translates a high-level language, such as C, into machine language. |
| ***configuration*** | In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance. |
| ***configuration space*** | In EZ-PD™ PMG1-S3 MCU devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’. |

## Terminology

| | |
|---|---|
| ***crowbar*** | A type of over-voltage protection that rapidly places a low-resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value. |
| ***CPUSS*** | CPU subsystem |
| ***crystal oscillator*** | An oscillator in which the frequency is controlled by a piezoelectric crystal. Typically a piezoelectric crystal is less sensitive to ambient temperature than other circuit components. |
| ***cyclic redundancy check (CRC)*** | A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression. |

### D

| | |
|---|---|
| ***data bus*** | A bi-directional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions. |
| ***data stream*** | A sequence of digitally encoded signals used to represent information in transmission. |
| ***data transmission*** | Sending data from one place to another by means of signals over a channel. |
| ***debugger*** | A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory. |
| ***dead band*** | A period of time when neither of two or more signals are in their active state or in transition. |
| ***decimal*** | A base-10 numbering system, which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 (called digits) together with the decimal point and the sign symbols + (plus) and - (minus) to represent numbers. |
| ***default value*** | Pertaining to the pre-defined initial, original, or specific setting, condition, value, or action a system will assume, use, or take in the absence of instructions from the user. |
| ***device*** | The device referred to in this manual is the EZ-PD™ PMG1-S3 MCU device, unless otherwise specified. |
| ***die*** | An non-packaged integrated circuit (IC), normally cut from a wafer. |
| ***digital*** | A signal or function, the amplitude of which is characterized by one of two discrete values: '0' or '1'. |
| ***digital blocks*** | The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI. |

## Terminology

| | |
|---|---|
| ***digital logic*** | A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system. |
| ***digital-to-analog (DAC)*** | A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation. |
| ***direct access*** | The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data. |
| ***duty cycle*** | The relationship of a clock period *high time* to its *low time*, expressed as a percent. |

### E

| | |
|---|---|
| ***External Reset (XRES_N)*** | An active high signal that is driven into the EZ-PD™ PMG1-S3 MCU device. It causes all operation of the CPU and blocks to stop and return to a pre-defined state. |

### F

| | |
|---|---|
| ***falling edge*** | A transition from a logic 1 to a logic 0. Also known as a negative edge. |
| ***feedback*** | The return of a portion of the output, or processed portion of the output, of a (usually active) device to the input. |
| ***filter*** | A device or process by which certain frequency components of a signal are attenuated. |
| ***firmware*** | The software that is embedded in a hardware device and executed by the CPU. The software may be executed by the end user, but it may not be modified. |
| ***flag*** | Any of various types of indicators used for identification of a condition or event (for example, a character that signals the termination of a transmission). |
| ***Flash*** | An electrically programmable and erasable, *volatile* technology that provides users with the programmability and data storage of EPROMs, plus in-system erasability. Nonvolatile means that the data is retained when power is off. |
| ***Flash bank*** | A group of flash ROM blocks where flash block numbers always begin with ‘0’ in an individual flash bank. A flash bank also has its own block level protection information. |
| ***Flash block*** | The smallest amount of flash ROM space that may be programmed at one time and the smallest amount of flash space that may be protected. A flash block holds 256 bytes. A flash block is also called as a flash row. |
| ***flip-flop*** | A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until it is made to change to the other state by application of the corresponding signal. |
| ***frequency*** | The number of cycles or events per unit of time, for a periodic function. |

## G

***gain*** — The ratio of output current, voltage, or power to input current, voltage, or power, respectively. Gain is usually expressed in dB.

***gate***

1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients.
2. One of many types of combinational logic elements having at least two inputs (for example, AND, OR, NAND, and NOR (also see *Boolean Algebra*)).

***ground***

1. The electrical neutral line having the same potential as the surrounding earth.
2. The negative side of DC power supply.
3. The reference point for an electrical system.
4. The conducting paths between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

## H

***hardware*** — A comprehensive term for all of the physical parts of a computer or embedded system, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.

***hardware reset*** — A reset that is caused by a circuit, such as a POR, watchdog reset, or external reset. A hardware reset restores the state of the device as it was when it was first powered up. Therefore, all registers are set to the POR value as indicated in register tables throughout this document.

***hexadecimal*** — A base 16 numeral system (often abbreviated and called hex), usually written using the symbols 0-9 and A-F. It is a useful system in computers because there is an easy mapping from four bits to a single hex digit. Thus, one can represent every byte as two consecutive hexadecimal digits. Compare the binary, hex, and decimal representations:

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

So the decimal numeral 79 whose binary representation is 0100 1111b can be written as 4Fh in hexadecimal (0x4F).

## Terminology

| | |
|---|---|
| ***high time*** | The amount of time the signal has a value of '1' in one period, for a periodic digital signal. |

### I

| | |
|---|---|
| ***$I^2C$*** | A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). $I^2C$ is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. $I^2C$ uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode. |
| ***idle state*** | A condition that exists whenever user messages are not being transmitted, but the service is immediately available for use. |
| ***impedance*** | 1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit.<br>1. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit. |
| ***input*** | A point that accepts data, in a device, process, or channel. |
| ***input/output (I/O)*** | A device that introduces data into or extracts data from a system. |
| ***instruction*** | An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly. |
| ***instruction mnemonics*** | A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV. |
| ***integrated circuit (IC)*** | A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor. |
| ***interface*** | The means by which two systems or devices are connected and interact with each other. |
| ***interrupt*** | A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed. |
| ***interrupt service routine (ISR)*** | A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution. |

## J

***jitter***

1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams.
1. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

## L

***latency*** — The time or delay that it takes for a signal to pass through a given circuit or network.

***least significant bit (LSb)*** — The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.

***least significant byte (LSB)*** — The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.

***Linear Feedback Shift Register (LFSR)*** — A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.

***load*** — The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).

***logic function*** — A mathematical function that performs a digital operation on digital data and returns a digital value.

***lookup table (LUT)*** — A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.

***low time*** — The amount of time the signal has a value of ‘0’ in one period, for a periodic digital signal.

***low-voltage detect (LVD)*** — A circuit that senses VDDD and provides an interrupt to the system when VDDD falls below a selected threshold.

## M

***M8CP*** — An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a EZ-PD™ PMG1-S3 MCU device by interfacing to the flash, SRAM, and register space.

## Terminology

| | |
|---|---|
| ***macro*** | A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total. |
| ***mask*** | 1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference.<br>2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems. |
| ***master device*** | A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*. |
| ***microcontroller*** | An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor. |
| ***mnemonic*** | A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction. |
| ***mode*** | A distinct method of operation for software or hardware. For example, the Digital EZ-PD™ PMG1-S3 MCU block may be in either counter mode or timer mode. |
| ***modulation*** | A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator. |
| ***Modulator*** | A device that imposes a signal on a carrier. |
| ***MOS*** | An acronym for metal-oxide semiconductor. |
| ***most significant bit (MSb)*** | The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb. |
| ***most significant byte (MSB)*** | The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB. |

## Terminology

***multiplexer (mux)***

1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

### N

***NAND***

See *Boolean Algebra*.

***negative edge***

A transition from a logic 1 to a logic 0. Also known as a falling edge.

***net***

The routing between devices.

***nibble***

A group of four bits, which is one-half of a byte.

***noise***

1. A disturbance that affects a signal and that may distort the information carried by the signal.
1. The random variations of one or more characteristics of any entity such as voltage, current, or data.

***NOR***

See *Boolean Algebra*.

***NOT***

See *Boolean Algebra*.

### O

***OR***

See *Boolean Algebra*.

***oscillator***

A circuit that may be crystal controlled and is used to generate a clock frequency.

***output***

The electrical signal or signals which are produced by an analog or digital block.

### P

***parallel***

The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.

***parameter***

Characteristics for a given block that have either been characterized or may be defined by the designer.

***parameter block***

A location in memory where parameters for the SSC instruction are placed prior to execution.

***parity***

A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).

## Terminology

| | |
|---|---|
| ***path*** | 1. The logical sequence of instructions executed by a computer.<br>1. The flow of an electrical signal through a circuit. |
| ***pending interrupts*** | An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled. |
| ***phase*** | The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees). |
| ***pin*** | A terminal on a hardware component. Also called lead. |
| ***pinouts*** | The pin number assignment: the relation between the logical inputs and outputs of the EZ-PD™ PMG1-S3 MCU device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names. |
| ***port*** | A group of pins, usually eight. |
| ***positive edge*** | A transition from a logic 0 to a logic 1. Also known as a rising edge. |
| ***posted interrupts*** | An interrupt that is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts. |
| ***Power On Reset (POR)*** | A circuit that forces the EZ-PD™ PMG1-S3 MCU device to reset when the voltage is below a pre-set level. This is one type of *hardware reset*. |
| ***program counter*** | The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed. |
| ***protocol*** | A set of rules. Particularly the rules that govern networked communications. |
| ***pulse*** | A rapid change in some characteristic of a signal (for example, phase or frequency), from a baseline value to a higher or lower value, followed by a rapid return to the baseline value. |
| ***pulse width modulator (PWM)*** | An output in the form of duty cycle which varies as a function of the applied measure. |

### R

| | |
|---|---|
| ***RAM*** | An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in. |
| ***register*** | A storage device with a specific capacity, such as a bit or byte. |
| ***reset*** | A means of bringing a system back to a know state. See *hardware reset* and *software reset*. |

## Terminology

| | |
|---|---|
| ***resistance*** | The resistance to the flow of electric current measured in ohms for a conductor. |
| ***revision ID*** | A unique identifier of the EZ-PD™ PMG1-S3 MCU device. |
| ***ripple divider*** | An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to $2^n - 1$. |
| ***rising edge*** | *See positive edge*. |
| ***ROM*** | An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in. |
| ***routine*** | A block of code, called by another block of code, that may have some general or frequent use. |
| ***routing*** | Physically connecting objects in a design according to design rules set in the reference library. |
| ***runt pulses*** | In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable. |

### S

| | |
|---|---|
| ***sampling*** | The process of converting an analog signal into a series of digital values or reversed. |
| ***schematic*** | A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer. |
| ***seed value*** | An initial value loaded into a linear feedback shift register or random number generator. |
| ***serial*** | 1. Pertaining to a process in which all events occur one after the other.<br>2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel. |
| ***set*** | To force a bit/register to a value of logic 1. |
| ***settling time*** | The time it takes for an output signal or value to stabilize after the input has changed from one value to another. |
| ***shift*** | The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12. |
| ***shift register*** | A memory storage device that sequentially shifts a word either left or right to output a stream of serial data. |

## Terminology

| | |
|---|---|
| ***sign bit*** | The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity. |
| ***signal*** | A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse. |
| ***silicon ID*** | A unique identifier of the EZ-PD™ PMG1-S3 MCU silicon. |
| ***skew*** | The difference in arrival time of bits transmitted at the same time, in parallel transmission. |
| ***slave device*** | A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device. |
| ***software*** | A set of computer programs, procedures, and associated documentation about the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed. |
| ***software reset*** | A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not EZ-PD™ PMG1-S3 MCU blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at flash address 0x0000. |
| ***SRAM*** | An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device. |
| ***SROM*** | An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash. |
| ***stack*** | A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off. |
| ***stack pointer*** | A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell. |
| ***state machine*** | The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences. |
| ***sticky*** | A bit in a register that maintains its value past the time of the event that caused its transition, has passed. |
| ***stop bit*** | A signal following a character or block that prepares the receiving device to receive the next character or block. |

## Terminology

| | |
|---|---|
| ***switching*** | The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network. |
| ***switch phasing*** | The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The EZ-PD™ PMG1-S3 MCU SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed. |
| ***synchronous*** | 1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.<br>1. A system whose operation is synchronized by a clock signal. |

### T

| | |
|---|---|
| ***tap*** | The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider. |
| ***terminal count*** | The state at which a counter is counted down to zero. |
| ***threshold*** | The minimum value of a signal that can be detected by the system or sensor under consideration. |
| ***Thumb-2*** | The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions. |
| ***transistors*** | The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers. |
| ***tristate*** | A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*. |

### U

| | |
|---|---|
| ***UART*** | A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits. |
| ***user*** | The person using the EZ-PD™ PMG1-S3 MCU device and reading this manual. |

## Terminology

| | |
|---|---|
| ***user modules*** | Pre-build, pre-tested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital EZ-PD™ PMG1-S3 MCU Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function. |
| ***user space*** | The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program. |

### V

| | |
|---|---|
| ***V*DDD** | A name for a power net meaning “voltage drain.” The most positive power supply signal. Usually 5 or 3.3 volts. |
| ***volatile*** | Not guaranteed to stay the same value or level when not in scope. |
| $V_{ss}$ | A name for a power net meaning “voltage source.” The most negative power supply signal. |

### W

| | |
|---|---|
| ***watchdog timer*** | A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time. |
| ***waveform*** | The representation of a signal as a plot of amplitude versus time. |

### X

| | |
|---|---|
| ***XOR*** | See *Boolean Algebra*. |

**Trademarks**
All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2022-03-04**
**Published by**
**Infineon Technologies AG**
**81726 Munich, Germany**

**© 2022 Infineon Technologies AG.**
**All Rights Reserved.**

**Do you have a question about any aspect of this document?**
**Go to www.infineon.com/support**

**Document reference**
**002-33922 Rev. *A**

**IMPORTANT NOTICE**
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").
With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.
In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.
The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

**WARNINGS**
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.